Appunti di Ricerca Operativa

2023/2024

Prefazione

La Ricerca Operativa è un campo in continua evoluzione, il cui impatto sulle realtà aziendali ed organizzative è in costante crescita. L'insegnamento di questa disciplina, ed in particolare delle sue basi metodologiche ed algoritmiche, è quindi indispensabile nei corsi che mirano a formare molte figure con elevate capacità tecnologiche e manageriali, ad esempio—ma non solamente—nei corsi di Laurea in Informatica, Matematica, Ingegneria e materie affini.

Queste dispense sono state sviluppate dai docenti di Ricerca Operativa del Dipartimento di Informatica dell'Università di Pisa per il supporto a diversi corsi afferenti all'area della Ricerca Operativa, sia di base che avanzati, in molti diversi curricula della stessa Università, quali i Corsi di Laurea in Informatica, Informatica Applicata, Matematica, Ingegneria Gestionale, Ingegneria Elettronica, ed Ingegneria delle Telecomunicazioni, ed i Corsi di Laurea Magistrale in Informatica, Informatica e Networking, Data Science and Business Informatics, ed Ingegneria Gestionale. Inoltre, le dispense sono state adottate anche in Corsi di Laurea di altre Università italiane. Le dispense coprono sia gli aspetti di base del curriculum in Ricerca Operativa (problemi e modelli di ottimizzazione, Programmazione Lineare, algoritmi su grafi) che aspetti più avanzati, in particolare legati alla soluzione di problemi di Ottimizzazione Combinatoria $\mathcal{NP}$-hard. Ciò non esaurisce certamente lo spettro delle metodologie della Ricerca Operativa, né delle sue possibili applicazioni; in particolare non vengono affrontate le tematiche relative alla presenza di elementi nonlineari nei modelli di ottimizzazione, e non vengono discusse in dettaglio le molteplici possibili applicazioni delle metodologie descritte in campi quali i trasporti e la logistica, le telecomunicazioni, l'energia, l'economia e la finanza, l'informatica, l'intelligenza artificiale, la scienze fisiche e biologiche, e molti altri.

Queste dispense sono il frutto di un lavoro collettivo, svolto nel corso di molti anni da diverse persone, in forme e ruoli diversi. In particolare, hanno collaborato alla stesura di questo documento Giorgio Gallo, Stefano Pallottino, Maria Grazia Scutellà, Antonio Frangioni, Giancarlo Bigi, e Luca Mencarelli. Un aiuto particolare alla stesura e al miglioramento delle dispense è stato dato da Paola Cappanera e Maria Paola Scaparra. Molte altre persone, tra cui molti studenti dei corsi di Ricerca Operativa all'interno dei corsi di Laurea e di Laurea Magistrale dell'Università di Pisa, hanno contributo a queste dispense segnalando errori e suggerendo miglioramenti; a tutti loro va il ringraziamento degli estensori. Ogni errore ed imprecisione rimasta nel testo è esclusivamente responsabilità degli autori; segnalazioni a tal proposito sono caldamente benvenute.

Indice

Capitolo 1

Problemi e Modelli

La *Ricerca Operativa* ha come oggetto lo studio e la messa a punto di metodologie per la soluzione di *problemi decisionali*. I problemi affrontati nell'ambito della Ricerca Operativa sono tipicamente quelli in cui bisogna prendere *decisioni* sull'uso di risorse disponibili in quantità limitata in modo da rispettare un insieme assegnato di condizioni (vincoli) e massimizzando il "beneficio" ottenibile dall'uso delle risorse stesse. La Ricerca Operativa considera quindi, in generale, tutte le metodologie utili a migliorare l'efficacia delle decisioni; ciò significa, in linea di principio, considerare tutte le fasi del *processo decisionale* che porta a prenderle. In modo molto schematico, queste fasi sono:

1. individuazione del problema;
2. analisi della realtà e raccolta dei dati;
3. costruzione del modello;
4. determinazione di una o più soluzioni;
5. analisi dei risultati ottenuti.

Questi punti non devono essere visti come strettamente sequenziali, in quanto ciascuno dei punti è fortemente correlato con quelli che lo precedono e seguono. La stessa raccolta dei dati presuppone un'idea (magari embrionale) sul tipo di modello che sarà costruito, e la scelta del modello è spesso funzionale alle esigenze della fase successiva, in quanto il modello deve ammettere approcci risolutivi in grado di determinare soluzioni in tempo compatibili con le esigenze del processo decisionale (se la decisione va presa entro domani, non posso aspettare una settimana per avere la risposta). Viceversa, in un processo decisionale reale è frequente il caso in cui una delle fasi richieda modifiche dei risultati ottenuti in una fase precedente. Ad esempio, nella costruzione del modello può emergere l'esigenza di nuovi dati in aggiunta a quelli già raccolti. Oppure, la determinazione delle soluzioni costringe a rivedere il modello in quanto il costo computazionale della sua soluzione si rivela essere troppo elevato. O ancora, l'analisi dei risultati ottenuti mostra che il modello non cattura in modo adeguato la realtà che dovrebbe rappresentare, e quindi porta alla sua modifica.

La Ricerca Operativa ha quindi sia un vastissimo ambito di applicazione, che, in linea di principio, la necessità di utilizzare o sviluppare metodologie molto diversificate che vanno dall'analisi dei sistemi produttivi e sociali, alla capacità di raccogliere dati statisticamente significativi (possibilmente estraendoli da una grande mole di dati scarsamente strutturati), fino allo sviluppo di metodologie per costruire modelli efficaci di porzioni della realtà che possano essere risolti efficientemente. Questi ultimi due aspetti, corrispondenti ai punti (3) e (4), sono quelli sui quali si concentrano queste dispense; il materiale qui presentato si quindi situa in particolare nell'area di studio che prende il nome di *Programmazione Matematica*. La scelta è dettata dal fatto che la costruzione di modelli e la loro soluzione algoritmica sono le fasi che più si prestano ad una rigorosa trattazione matematica, e quindi quelle più adatte a studenti in materie tecniche e scientifiche. In ogni caso, a questi argomenti sono dirette gran parte delle metodologie messe a punto nell'ambito della Ricerca Operativa.

È opportuno però sottolineare come queste dispense non presentino di certo un panorama esaustivo della Ricerca Operativa. In primo luogo, infatti, la Programmazione Matematica è solo una delle

componenti, per quanto spesso necessaria ed importante, di quella che è più in generale la *Scienza delle Decisioni*. In secondo luogo, ovviamente, i contenuti di queste dispense riguardano solo alcuni tra i più basilari risultati del settore. L'enorme mole di ricerche svolte in questo campo ha determinato un corpus di risultati, peraltro in continua espansione, di cui queste note possono fornire solo un primissimo assaggio, cercando ove possibile di far intravedere *in nuce* tematiche che possono essere sviluppate in profondità ed ampiezza grandemente superiori. Infatti, queste dispense sono state pensate per corsi di base di Ricerca Operativa. Molti altri aspetti rilevanti non sono neanche accennati in questo materiale, tra i quali

- algoritmi e metodologie per modelli di Programmazione Matematica diversi, quali quelli con prevalenza di elementi non lineari (*programmazione non lineare*) oppure caratterizzati da forte incertezza sui dati (*ottimizzazione robusta e/o stocastica*);
- applicazioni a particolari ambiti operativi quali i *sistemi logistici*, le *reti di telecomunicazione*, oppure l'apprendimento automatico (Machine Learning/Artificial Intelligence);
- metodologie per tipi di modelli diversi da quelli considerati nella Programmazione Matematica, quali le *tecniche di simulazione* oppure la *teoria dei giochi*.

Alcuni di questi sono affrontati in corsi più avanzati dell'area di Ricerca Operativa.

È utile segnalare infine come la Ricerca Operativa sia una disciplina che ha profonde relazioni con moltissime altre, ed in modi diversi. Infatti, da una parte l'ottimizzazione ha un ruolo di rilievo in moltissimi ambiti tecnologici e scientifici, tra i quali ad esempio la statistica, l'analisi numerica, l'informatica, l'apprendimento automatico / intelligenza artificiale, la fisica, la chimica, l'economia e la finanza, diversi ambiti dell'ingegneria (energetica, informatica, gestionale, ...), e molti altri ancora. Ma dall'altra parte le metodologie dell'ottimizzazione traggono risultati da molti ambiti diversi della matematica (discreta, continua, logica, ...) e sono profondamente influenzate dalle evoluzioni scientifiche e tecnologiche relative a molte discipline quali, ancora, la statistica, l'analisi numerica, e molti settori dell'informatica (algoritmica, metodologie di programmazione, tecniche di programmazione parallela, ...). Vale la pena in particolare di sottolineare come sostanziali progressi nelle metodologie di ottimizzazione (ad esempio, lo sviluppo di metodologie per i problemi di ottimizzazione misti-interi, nonlineari e nonconvessi) siano storicamente stati ottenuti attraverso lo stimolo di particolari ambiti applicativi (ad esempio, l'ingegneria chimica) i cui specifici problemi hanno fortemente motivato la ricerca in quel particolare ambito, spesso condotta in prima persona da specialisti dell'applicazione stessa. In questo senso la Ricerca Operativa si qualifica come inerentemente interdisciplinare, ed è aperta a contributi di studiosi di molti campi diversi.

1.1 Modelli e Problemi

L'elemento centrale nel processo decisionale è il *modello*, una descrizione, in generale per mezzo di strumenti di tipo logico-matematico, della porzione di realtà di interesse. Si distinguono almeno tre classi principali di modelli:

- Nei *giochi*, la difficoltà di modellare in modo matematico il comportamento degli individui o dei gruppi di individui presenti nella realtà sotto esame viene superata introducendo direttamente l'uomo nel modello attraverso i giocatori, a ciascuno dei quali viene affidato un prefissato ruolo.
- Nei *modelli di simulazione* si cerca di descrivere nel modo più accurato possibile il comportamento del sistema che si vuole studiare per mezzo di relazioni matematiche; quindi si studia su calcolatore la sua risposta a sollecitazioni che vengono realizzate, in genere per mezzo di generatori di numeri pseudo casuali, in modo che siano il più possibile simili a quelle reali.
- Nei *modelli analitici* invece tutto il sistema sotto esame è descritto per mezzo di relazioni matematiche (o logiche) tra variabili che rappresentano gli elementi del sistema; quindi si cercano valori per tali variabili che soddisfino i vincoli e che massimizzino o minimizzino una *funzione obiettivo* opportunamente definita.

Nell'analizzare la realtà per mezzo di modelli non va mai dimenticato lo scarto esistente tra la realtà stessa ed il modello: le soluzioni di un modello sono in realtà sempre soluzioni della *rappresentazione* che abbiamo costruito del problema reale, ma "la mappa non è il mondo". È quindi sempre necessario prestare grande attenzione alla fondatezza del modello costruito, in quanto esso sarà sempre una descrizione molto limitata della realtà, ma dovrà rappresentare con ragionevole accuratezza almeno gli aspetti che interessano ai fini della soluzione del problema decisionale che si sta affrontando.

Una caratteristica fondamentale di ogni modello è quindi la sua capacità di fornire previsioni corrette sul comportamento della realtà modellata in determinate circostanze di interesse. Ma altrettanto importante è la sua *utilizzabilità operativa*: deve essere possibile raccogliere i dati che caratterizzano il modello e determinare le soluzioni in un tempo compatibile con le esigenze del processo decisionale corrispondente. Ciò può voler dire cose molto diverse, dai problemi di controllo ottimo in cui la risposta deve essere disponibile in millisecondi fino ai problemi di pianificazione di lungo termine in cui è possibile attendere settimane prima di avere una risposta. In ogni caso, però, esistono dei limiti (tempo e risorse) entro i quali la risposta deve essere ottenuta. Quest'ultimo è un punto assolutamente non banale, in quanto molti modelli sono, allo stato dell'arte, computazionalmente intrattabili (si veda ad esempio l'Appendice A). La "qualità" di un modello è quindi il risultato del (difficile) bilanciamento tra due necessità contrastanti: da una parte quella di tenere in conto di tutti gli elementi necessari ad una corretta descrizione dei fenomeni, e dall'altra quella di avere un modello "sufficientemente semplice" affinché sia possibile ottenere le risposte entro i limiti temporali e di risorse imposti dal processo decisionale. Questo è probabilmente lo snodo fondamentale di tutte le attività di creazione di modelli, ed è profondamente correlato con alcune delle domande più profonde riguardo alla capacità che gli uomini hanno di conoscere, prevedere e controllare il mondo che li circonda.

In queste dispense verranno trattati solamente modelli analitici, ed in particolare solo specifiche classi di questi. In questo contesto esiste una sostanziale equivalenza tra il concetto di modello e quello di *problema di ottimizzazione*; forniamo pertanto adesso alcune prime definizioni generali al riguardo. Per *problema* si intende una domanda espressa in termini generali, la cui risposta dipende da un certo numero di *parametri* e *variabili*. Un problema viene usualmente definito per mezzo di:

- una descrizione dei suoi parametri, in generale lasciati indeterminati;
- una descrizione delle proprietà che devono caratterizzare la risposta o *soluzione* desiderata.

Un'*istanza* di un dato problema (P) è quella particolare domanda che si ottiene specificando valori per tutti i parametri di (P).

Molto spesso un problema viene definito fornendo l'insieme F delle possibili risposte o soluzioni. Di tale insieme, detto *insieme ammissibile* o *regione ammissibile*, viene in generale data la struttura mediante i parametri da cui essa dipende; i suoi elementi vengono detti *soluzioni ammissibili*. Frequentemente F viene specificato indicando un insieme di "supporto" F' tale che $F \subset F'$, ed ulteriori condizioni (vincoli) che gli elementi di F devono soddisfare. In questo caso, si parla spesso degli elementi di $F' \setminus F$ come di *soluzioni non ammissibili*.

In un *problema di ottimizzazione*, sull'insieme ammissibile F viene definita una *funzione obiettivo*

$$c \,:\, F \,\to\, \mathbb{R}$$

che fornisce il costo o il beneficio associato ad ogni soluzione; la soluzione del problema è un elemento di F che rende minima, oppure massima, la funzione obiettivo. Un generico *problema di minimo* può essere scritto come

$$(P) \qquad \min \{\, c(\,x\,) \,:\, x \in F \,\} \ . \tag{1.1}$$

Sostituendo "min" con "max" in (1.1) si ottiene un *problema di massimo*. Chiamiamo

$$z(\,P\,) \;=\; \min \{\, c(\,x\,) \,:\, x \in F \,\}$$

il *valore ottimo* del problema. Una soluzione ammissibile $x^* \in F$ tale che $c(\,x^*\,) = z(\,P\,)$ (se esiste) è detta *soluzione ottima* per (P). Si noti che può esistere un numero arbitrario (anche infinito, se F lo è) di soluzioni ottime alternative: si pensi al caso estremo della funzione costante $c(\,x\,) = 0$ in

cui qualsiasi soluzione ammissibile è ottima. Nella pratica è possibile che esistano criteri diversi per valutare la bontà di una soluzione, e che quindi soluzioni diverse con lo stesso valore della funzione obiettivo non siano equivalenti. Ciò però corrisponde al fatto che le preferenze del decisore non sono state completamente rappresentate nella definizione di $c(\cdot)$. In queste dispense assumeremo invece che la funzione obiettivo rappresenti in modo esaustivo le preferenze del decisore, e che quindi tutte le soluzioni ottime alternative siano equivalenti. Esistono diverse ragioni per cui questa assunzione potrebbe non essere soddisfatta in situazioni reali, tra cui la difficoltà di estrarre rappresentazioni matematiche delle opinioni di decisori umani. Inoltre, i diversi criteri potrebbero non essere confrontabili tra di loro; ciò porta verso quella che è nota come *ottimizzazione vettoriale* o *multi-obiettivo*, che non può essere adeguatamente trattata in un corso di base.

Esempio 1.1. Un problema di equipartizione

Il problema di equipartizione corrisponde alla seguente domanda: dato un insieme di n numeri naturali, $N = \{ a_1 , a_2 , \dots , a_n \}$, qual'è il sottoinsieme S di N tale che la differenza in modulo tra la somma dei numeri in S e quella dei numeri in $N \setminus S$ è la più piccola possibile? Una formulazione matematica del problema è

$$(EQ) \quad \min \left\{ c(S) = \left| \sum_{a \in S} a - \sum_{a \in N \setminus S} a \right| \, : \, S \subseteq N \right\} \ . \tag{1.2}$$

In questo caso, F è l'insieme di tutti i sottoinsiemi di N o *insieme delle parti* di N, normalmente indicato come 2^N; infatti, l'unica condizione (vincolo) che una risposta (soluzione) deve soddisfare è di essere un sottoinsieme degli n numeri dati. Per questo problema, i *parametri* sono il numero "n" e gli n numeri "a_1, a_2, ..., a_n"; scegliendo ad esempio $n = 4$ e $N = \{ 7 , 3 , 4 , 6 \}$ si ottiene una particolare istanza del problema, in cui tutti i parametri sono specificati. Invece, S è la *variabile* del problema: $S = \{ 3 , 7 \}$ è una soluzione ottima per l'istanza considerata, in quanto $c(S) = 0$ ed ovviamente $z(EQ) \geq 0$. Si noti che $S = \{ 4 , 6 \}$ è banalmente una diversa soluzione ottima del problema; l'esempio illustra quindi il fatto, precedentemente anticipato, che la soluzione ottima (se esiste) non deve necessariamente essere unica. Come spesso accade, il numero di soluzioni ammissibili diverse del problema cresce quindi molto rapidamente con le dimensioni del problema, essendo in questo caso 2^n: in effetti, nonostante la sua apparente semplicità (EQ) è un problema $\mathcal{NP}$-hard, e non sono noti algoritmi in grado di risolverlo in generale in tempo polinomiale in n (anche se il problema è in effetti "uno dei più facili tra quelli difficili" e può ammettere algoritmi di risoluzione efficienti con opportune condizioni sugli a_i), il che si può pensare in prima battuta essere dovuto al fatto che "ci sono troppe soluzioni ammissibili da guardare per scegliere quella ottima".

La precedente descrizione del problema non è l'unica possibile. In effetti, la possibilità di esprimere un dato problema di ottimizzazione in forme diverse è importante, in quanto rappresentazioni appropriate possono permettere l'uso di tecniche risolutive più efficienti. Si parla in questo caso di *formulazioni equivalenti* del problema; data una formulazione, la costruzione di formulazioni equivalenti si dice *riformulazione*. Torneremo in seguito su questo concetto generale; in questo caso lo applichiamo in modo banale notando che il problema può alternativamente essere scritto utilizzando l'insieme degli *indici* $I = \{ 1 , 2 , \dots , n \}$ degli oggetti, ossia

$$(EQ) \quad \min \left\{ c(S) = \left| \sum_{i \in S} a_i - \sum_{i \in N \setminus} a_i \right| \, : \, S \subseteq I \right\} \ . \tag{1.3}$$

È ovvio come le due formulazioni (1.2) ed (1.3) siano equivalenti. Presentiamo la seconda principalmente per sottolineare come spesso nei problemi di ottimizzazione sono presenti molte entità che sono "diverse ma giocano sostanzialmente lo stesso ruolo", come nel nostro caso i diversi numeri a_i. In generale si tende spesso a preferire l'utilizzo, nelle descrizioni matematiche dei problemi, di *indici* che caratterizzano ciascuna di queste entità per identificarle, piuttosto che delle entità di per sé. In effetti, identificare gli *insiemi di indici* delle entità coinvolte nella descrizione del problema è spesso uno dei primi e fondamentali passi per costruire formulazioni dello stesso.

Un problema di ottimizzazione può essere indifferentemente codificato come problema di massimo o di minimo: infatti, è immediato verificare che il problema

$$(P') \qquad - \max \{ -c(x) \, : \, x \in F \}$$

è *equivalente* a (P), ossia $z(P') = z(P)$ (per questo il segno "$-$" nella definizione, altrimenti sarebbe $z(P') = -z(P)$) ed i due problemi hanno lo stesso insieme di soluzioni ottime.

Esempio 1.2. Un problema di impaccamento

Il problema del *Circle Packing* corrisponde alla seguente domanda: dato un quadrato di lato unitario ed un numero naturale n, qual è il massimo raggio r che possono avere n cerchi identici inscritti nel quadrato che non si intersecano tra di loro (tranne al più sul bordo)? La domanda può anche essere posta in modi diversi ma equivalenti, ad esempio qual è la minima dimensione di un quadrato che può contenere n cerchi di raggio unitario, oppure qual è il modo di localizzare n punti nel quadrato in modo che la minima distanza tra di essi sia massima. Di questo problema è

dimostrabile algebricamente quale sia la soluzione ottima per n piccolo (ad esempio $r = \sqrt{2}$ per $n = 2$, $r = \sqrt{6} - \sqrt{2}$ per $n = 3$, $r = 1/4$ per $n = 4$, ...), ma il problema è "difficile" per n arbitrario. In questo caso la descrizione dell'istanza è un semplice numero naturale, n. L'insieme ammissibile F sarebbe formalmente il sottoinsieme dei numeri reali $r \geq 0$ per cui è possibile costruire n cerchi opportuni, la soluzione ottima essendone evidentemente l'estremo destro. Ma *dimostrare* che un certo valore r è effettivamente ammissibile richiede in qualche modo di *esibire* tali cerchi. Pertanto, il primo passo è intanto definire l'insieme $I = \{ 1, 2, \ldots, n \}$ degli indici dei cerchi. Una possibile descrizione matematica della regione ammissibile comprende anche delle *variabili ausiliarie*, nella forma di n coppie $(x_i , y_i) \subseteq [0 , 1]^2$, $i \in I$, indicanti i centri dei cerchi. Insieme al singolo numero $r \in \mathbb{R}$ indicante il raggio, questo permette di costruire una *formulazione analitica* del problema:

$$(CP) \quad \max r \tag{1.4}$$

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \geq 2r \qquad i \in I \quad , \quad j = i + 1, \ldots, n \tag{1.5}$$

$$r \leq x_i \leq 1 - r \qquad i \in I \tag{1.6}$$

$$r \leq y_i \leq 1 - r \qquad i \in I \tag{1.7}$$

In tale formulazione si identificano diversi elementi. Innanzi tutto ovviamente la funzione obiettivo (1.4); si noti come in questo caso essa dipenda solamente dalla *variabile principale* o *strutturale* r, e non dalle *variabili ausiliarie*. Questo è comune, ma non significa certo che le variabili ausiliarie non siano comunque fondamentali per esprimere correttamente il problema (in questo caso, dimostrare il fatto che un certo valore di r sia oppure no ammissibile, che in effetti è esattamente il "cuore" del problema). Oltre alla funzione obiettivo si individuano poi *blocchi* (o *gruppi*, o *famiglie*) di vincoli, ciascuno dei quali tipicamente organizzato per esprimere una delle condizioni logiche fondamentali che caratterizzano l'ammissibilità di una soluzione. Descriviamo adesso e giustifichiamo ciascuno dei blocchi.

- Il blocco (1.5) contiene un vincolo per ciascuna possibile coppia (i , j), ed esprime la condizione che i centri dei due cerchi corrispondenti, (x_i , y_i) ed (x_j , y_j), devono essere a distanza almeno pari al doppio di r in modo tale che i cerchi non si intersechino, se non al più in un punto sulla circonferenza di entrambi. Poiché la relazione è simmetrica—vale per (i , j) se e solo se vale per (j , i)—è sufficiente imporla su una sola delle due coppie, che è possibile ad esempio selezionando solo quelle indicate; si sarebbe anche potuto introdurre un apposito insieme di (indici di) coppie ordinate $C = \{ (i , j) \in I \times I : i < j \}$, o equivalentemente l'insieme delle coppie non ordinate $O = \{ \{ i , j \} \in I \times I \}$.
- Il blocco (1.6) contiene un vincolo per ciascun cerchio, ed esprime la condizione che la coordinata x del centro deve essere sufficientemente interna al quadrato (a distanza almeno r dal bordo, su entrambi i lati) in modo che il cerchio sia completamente interno ad esso.
- Il blocco (1.7) esprime la condizione analoga al precedente per la coordinata y del centro dei cerchi.

Tali condizioni, nel loro insieme, assicurano che qualsiasi scelta di valori delle variabili che rispetta ciascuno dei vincoli corrisponda ad una soluzione ammissibile del problema. Di converso, data una qualsiasi soluzione ammissibile del problema (n cerchi di raggio r interni al quadrato unitario che non si intersecano se non sulle rispettive circonferenze) esiste chiaramente il modo di rappresentarla dando opportuni valori alle variabili della formulazione. Di conseguenza (1.4)–(1.7) è una formulazione corretta del problema. Come nel caso precedente sono possibili formulazioni diverse del problema, che però non discuteremo.

Il problema (CP) si classifica come *continuo*, in quanto il suo insieme ammissibile ha (in principio) un numero infinito *non enumerabile* di elementi diversi, a differenza del problema (EQ) che si classifica come *combinatorio* perchè ha un insieme *enumerabile* di soluzioni ammissibili diverse; l'insieme ammissibile di (EQ) è in effetti *finito*, ma il numero delle soluzioni cresce esponenzialmente con la dimensione del problema (n), e quindi diviene rapidamente enorme. Nella pratica queste due classi problemi, nonostante alcune differenze di rilievo, sono usualmente "ugualmente difficili".

Dato un problema di ottimizzazione (P), sono possibili tre casi:

i. Il problema è *vuoto*, ossia $F = \emptyset$; in questo caso si assume per convenzione $z(P) = +\infty$ ($-\infty$ per un problema di massimo). L'esistenza di problemi vuoti potrebbe a tutta prima parere paradossale, ma in generale, come vedremo, non è facile stabilire se un insieme F specificato attraverso una lista di condizioni (vincoli) contenga oppure no elementi.

ii. Il problema è *inferiormente illimitato* (*superiormente illimitato* per un problema di massimo), ossia comunque scelto un numero reale M esiste una soluzione ammissibile $x \in F$ tale che $c(x) \leq M$ ($\geq M$); in questo caso il valore ottimo è $z(P) = -\infty$ ($+\infty$).

iii. Il problema ha *valore ottimo finito* ed *ammette soluzione ottima*.

In realtà è matematicamente possibile anche un quarto caso: il problema ha *valore ottimo finito* $-\infty < z(P) < \infty$ ma *non ha soluzione ottima*, ossia non esiste nessun $x \in F$ tale che $c(x) = z(P)$. Un semplice esempio è dato dai problemi

$$\inf\{1/x : x \geq 0\} \qquad \text{e} \qquad \inf\{x : x > 0\}$$

che ammettono valore ottimo 0 ma nessuna soluzione ottima. Si noti come si è in questo caso utilizzata esplicitamente la notazione "inf", che corrisponde al fatto che qualsiasi insieme di numeri reali ha sempre un infimo (supremo), che però può non appartenervi. Questo può accadere in due modi, il primo essendo che l'insieme non è inferiormente (superiormente) limitato, per cui l'infimo (supremo) si indica convenzionalmente con $-\infty$ ($+\infty$), che non sono propriamente numeri reali ma *reali estesi*; questo è il caso considerato nel punto ii) in precedenza. Un caso diverso è quello in cui l'infimo (supremo) sia finito, ma non faccia parte dell'insieme, come nei due esempi. Nella pratica tale situazione è indesiderabile e viene evitata avendo cura di scegliere in modo opportuno $c(\cdot)$ ed F; ciò sarà sempre fatto per i problemi trattati in questo corso, e pertanto si utilizzerà sempre la notazione "min" (o "max") che corrisponde al fatto che l'infimo (supremo) dell'insieme di numeri reali considerato—l'immagine della regione ammissibile F attraverso la funzione obiettivo $c(\cdot)$—vi appartiene, considerando accettabile il caso in cui esso sia $-\infty$ ($+\infty$). È comunque anche vero che tale distinzione è spesso irrilevante nella pratica, poiché nella maggior parte dei casi gli algoritmi vengono implementati utilizzando *numeri a virgola mobile* come "approssimazione" dei numeri reali. Tali numeri hanno una precisione ε finita, il che implica che due soluzioni ammissibili x' ed x'' tali che $|c(x') - c(x'')| < \varepsilon$ siano indistinguibili in termini di valore della funzione obiettivo. Nei due esempi precedenti, quindi, valori di x rispettivamente opportunamente grandi o vicini a 0 possono essere considerati soluzioni ottime a tutti gli effetti. La gestione della precisione numerica negli algoritmi illustrati in queste dispense in effetti può essere non banale, e non verrà affrontata esplicitamente. Accenniamo solamente che qualsiasi condizione del tipo "$x = 0$", "$x > 0$" o "$x < 0$" deve essere implementata (rispettivamente) come $|x| \leq \varepsilon$, $x \geq \varepsilon$ e $x \leq -\varepsilon$ per un opportuno valore di $\varepsilon > 0$, tipicamente più grande della precisione di macchina dei numeri a virgola mobile considerati e spesso non banale da determinare.

In certi casi ciò che il problema richiede è semplicemente la determinazione di una qualsiasi soluzione ammissibile, ovvero di fornire un elemento $x \in F$, se ne esiste uno, oppure di dichiarare che F è vuoto; in questo caso si parla di *problema decisionale* oppure di *problema di esistenza*. Dato un problema decisionale definito su $F \subseteq F'$, ad esso è naturalmente associato il *problema di certificato*: dato $x \in F'$, verificare se $x \in F$. Il problema di certificato è un problema decisionale che richiede semplicemente una risposta "sì" oppure "no".

In teoria, qualsiasi problema decisionale può essere formulato come problema di ottimizzazione: basta scegliere un opportuno insieme $F' \supseteq F$ e definire $c(x) = 0$ per ogni $x \in F$, $c(x) = 1$ per ogni $x \in F' \setminus F$. Analogamente, un problema di ottimizzazione può essere espresso come problema decisionale: basta usare come insieme in cui si cerca una soluzione ammissibile l'insieme delle sue soluzioni ottime. Quest'ultima equivalenza è però solamente teorica; in pratica è difficile definire esplicitamente l'insieme delle soluzioni ottime di un problema non essendo noto il suo valore ottimo. In alternativa, dato il problema di ottimizzazione (1.1) possiamo definire il suo *problema decisionale associato*, o sua *versione decisionale*, come il problema di verificare l'esistenza di un soluzione ammissibile nell'insieme

$$F_k = \{x \in F : c(x) \leq k\} ,$$

dove k è un prefissato valore. Si cerca cioè se esiste una soluzione ammissibile del problema di ottimizzazione che fornisca un valore della funzione obiettivo non superiore a k. In un certo senso, il problema decisionale associato ad un problema di ottimizzazione ne è una versione parametrica: facendo variare il parametro k e risolvendo ogni volta un problema di esistenza, è possibile determinare il valore ottimo della funzione obiettivo, o almeno una sua approssimazione con precisione arbitraria (risolvere il problema di ottimizzazione equivale a risolverne la variante decisionale per ogni possibile valore di k).

Per il problema di equipartizione (1.2) il problema decisionale associato richiede di stabilire se esiste un sottoinsieme S tale che la differenza in valore assoluto tra la somma dei numeri in S e quella dei

numeri in $N \setminus S$ sia minore od uguale di un dato numero (intero) k. Per $k = 0$ si ottiene il problema di decidere se esiste un sottoinsieme S tale che la somma dei numeri in S e quella dei numeri in $N \setminus S$ sia uguale. Per il problema del Circle Packing (1.4)–(1.7) il problema decisionale associato richiede, dato un numero reale r, di stabilire se esiste un modo di "impaccare" n cerchi di raggio r nel quadrato unitario.

Il senso di definire un problema di ottimizzazione è, almeno per le applicazioni pratiche, strettamente collegato alla possibilità di sviluppare procedure di calcolo, o *algoritmi*, in grado di risolverne efficientemente le istanze. Da questo punto di vista, i problemi di ottimizzazione possono essere divisi in (almeno) due classi: quella dei problemi polinomiali ($\mathcal{P}$), e quella dei problemi $\mathcal{NP}$-ardui ($\mathcal{NP}$-completi quando si parli di problemi decisionali); per un rapido richiamo di questi concetti si veda l'Appendice A. Data l'esistenza di problemi per i quali non si conoscono algoritmi risolutivi di complessità polinomiale, è opportuno discutere più in dettaglio cosa significhi *risolvere un problema di ottimizzazione*.

In generale, un algoritmo che risolva il problema (P) è una procedura che prende in input una (descrizione di una) qualsiasi istanza p di (P) e fornisce in output una soluzione ottima x^* per quell'istanza. Nel caso in cui esistano più soluzioni ottime diverse, l'algoritmo deve semplicemente riportare *una qualsiasi* di esse. In alcuni casi si può essere interessati a determinare *tutte* le soluzioni ottime alternative del problema, ma molto spesso tale insieme può risultare "troppo grande", ad esempio di cardinalità esponenziale nelle dimensioni dell'istanza, e quindi normalmente si accetta che l'algoritmo ne riporti una. Ciò ha senso, in quanto soluzioni con lo stesso valore della funzione obiettivo sono per definizione equivalenti "agli occhi del decisore", e pertanto non c'è alcun motivo di distinguere tra di esse. Per molti problemi ciò è sostanzialmente—a meno di un fattore moltiplicativo polinomiale—equivalente a fornire in output il solo *valore ottimo* dell'istanza (per un approfondimento di questi concetti si veda il §4.3); possiamo quindi definire formalmente un algoritmo come una funzione $\mathcal{A} : P \to \mathbb{R}$. Un algoritmo per (P) che determini una soluzione ottima per *qualsiasi* istanza del problema viene detto *algoritmo esatto*. Poiché gli algoritmi esatti possono avere complessità troppo elevata, ad esempio esponenziale nelle dimensioni dell'istanza, ci si trova spesso nella necessità di ricorrere ad *algoritmi euristici*, detti anche semplicemente *euristiche* (si veda il Capitolo 5), ossia algoritmi che determinano solamente *una qualsiasi* soluzione ammissibile. In generale si è interessati ad ottenere "buone" valutazioni superiori (per problemi di minimo); per questo è opportuno introdurre misure che indichino "quanto buona" è una data soluzione. Data un'istanza p del problema di ottimizzazione (P), con valore ottimo $z(\,p\,)$, ed una sua soluzione ammissibile $\bar{x} \in F$, l'*errore assoluto* di $\bar{x}$

$$E_{\bar{x}} = c(\,\bar{x}\,) - z(\,p\,)$$

è una misura della "bontà" di $\bar{x}$ come soluzione di p; si noti che $c(\,\bar{x}\,) \geq z(\,p\,)$, ossia l'euristica produce un'*approssimazione superiore* (inferiore per un problema di massimo) del valore ottimo dell'istanza, e pertanto $E_{\bar{x}} \geq 0$ (nel caso di problemi di massimo bisogna quindi invertire il segno nella definizione). Poiché tale misura non è invariante rispetto ai cambiamenti di scala, si preferisce utilizzare l'*errore relativo*

$$R_{\bar{x}} = (\,c(\,\bar{x}\,) - z(\,p\,)\,) \,/\, z(\,p\,)$$

(assumiamo $z(p) > 0$ per semplificare la discussione, altrimenti occorre adattare marginalmente la trattazione). Dato $\varepsilon > 0$, la soluzione $\bar{x}$ si dice *ε-ottima* se $R_{\bar{x}} \leq \varepsilon$: un algoritmo euristico può essere considerato "buono" se produce soluzioni con "piccolo" errore relativo per *tutte* le istanze di (P). Un algoritmo $\mathcal{A}$ si dice quindi *ε-approssimato* se produce una soluzione ε-ottima per ogni istanza. Algoritmi ε-approssimati con ε "piccolo" possono essere valide alternative agli algoritmi esatti; per ulteriori informazioni si consulti l'Appendice A. In effetti, in generale determinare il valore reale esatto del problema può non essere possibile, per via del fatto precedentemente accennato che le computazioni sono svolte con numeri in virgola mobile e quindi a precisione finita, ma questo aspetto non verrà approfondito in queste dispense. Accenniamo qui solamente al fatto che spesso in pratica anche gli algoritmi esatti sono in effetti approssimati, per via di tale problema, con una precisione ε "abbastanza vicina" a quella di macchina per cui la soluzione ottenuta sia sostanzialmente indistinguibile da quella ottima; per molti problemi $\mathcal{NP}$-ardui ottenere una soluzione di questo tipo

è comunque molto complesso, in teoria come in pratica. Esistono comunque casi in cui gli algoritmi possono fornire il valore esatto, o perché il problema ha proprietà particolari (ad esempio, è noto che il valore ottimo è un intero) oppure eseguendo le computazioni attraverso sistemi di algebra simbolica che hanno precisione infinita (ma che sono tipicamente molto inefficienti). Ancora, di questi aspetti non è possibile trattare in dettaglio in queste dispense.

Si noti comunque che, per molti problemi di ottimizzazione, il problema di determinare *una qualsiasi* soluzione ammissibile ha la stessa complessità del problema originario; quindi, in generale gli algoritmi euristici possono "fallire", ossia non riportare nessuna soluzione ammissibile anche per istanze in cui $F \neq \emptyset$. In questo caso si assume che la valutazione superiore (inferiore, per un problema di massimo) ottenuta dall'algoritmo sia $+\infty$ $(-\infty)$, ossia "arbitrariamente cattiva", e quindi $R_{\bar{x}} = +\infty$.

Un problema fondamentale delle misure appena introdotte è che, in generale, il valore $z(\,p\,)$ non è noto, ed anzi calcolarlo è "altrettanto difficile" che determinare una soluzione ottime (si veda il §4.3), per cui calcolare l'errore (assoluto o relativo) di una soluzione $\bar{x}$ è non banale. Un metodo per stimare $z(\,p\,)$ è quello di costruire una qualche approssimazione del problema dato, ad esempio considerando solamente alcune delle condizioni (vincoli) che le soluzioni ammissibili devono soddisfare. In particolare, si definisce *rilassamento* di (1.1) qualsiasi problema

$$(\bar{P}) \qquad \min\{\,\bar{c}(\,x\,) \,:\, x \in \bar{F}\,\} \tag{1.8}$$

tale che $F \subseteq \bar{F}$ e $\bar{c}(\,x\,) \leq c(\,x\,)$ per ogni $x \in F$. In altre parole, $(\bar{P})$ è un rilassamento di (P) se ha "più soluzioni" di (P) e/o se la sua funzione obiettivo è un'approssimazione inferiore della funzione obiettivo $c(\cdot)$ di (P) sull'insieme F. È immediato verificare che il valore ottimo di $(\bar{P})$ fornisce una *valutazione inferiore* del valore ottimo di (P), ossia $z(\,\bar{P}\,) \leq z(\,P\,)$; nel caso di problemi di massimo, la seconda condizione diventa $\bar{c}(\,x\,) \geq c(\,x\,)$ per ogni $x \in F$, ed il rilassamento fornisce una *valutazione superiore* del valore ottimo. È spesso possibile definire i rilassamenti in modo che siano risolubili mediante algoritmi di complessità inferiore rispetto a quelli necessari per il problema originale; si veda il Capitolo 6.

L'utilità fondamentale di un rilassamento è quella di permettere la *stima della qualità* di una soluzione ammissibile $\bar{x}$ dell'istanza p, ad esempio prodotta da un'euristica; infatti, se $\bar{p}$ è un rilassamento di p (e assumendo ancora $z(\,\bar{p}\,) > 0$ per semplicità), si ha che

$$R_{\bar{x}} = \frac{c(\,\bar{x}\,) - z(\,p\,)}{z(\,p\,)} \leq \bar{R}_{\bar{x}} = \frac{c(\,\bar{x}\,) - z(\,\bar{p}\,)}{z(\,\bar{p}\,)} \quad . \tag{1.9}$$

A differenza di $R_{\bar{x}}$, $\bar{R}_{\bar{x}}$ può essere effettivamente calcolato una volta ottenuta $\bar{x}$ e risolto $\bar{p}$; se risulta $\bar{R}_{\bar{x}}\varepsilon$, allora si può senz'altro concludere che $\bar{x}$ è ε-ottima. In altri termini, un rilassamento "facile" può fornire un *certificato di ottimalità* (approssimata) per un problema di ottimizzazione "difficile".

In effetti, può persino accadere che risolvere il rilassamento permetta di risolvere direttamente il problema originale. Questo capita in particolare se la soluzione ottima x^* di $(\bar{p})$ soddisfa le condizioni $x^* \in F$ e $\bar{c}(\,x^*\,) = c(\,x^*\,)$, ossia è ammissibile per il problema originale e la funzione obiettivo $\bar{c}$ ha in x^* lo stesso valore della funzione obiettivo reale c. In questo caso, infatti, è immediato verificare che x^* è anche *soluzione ottima per* (p), in quanto

$$\bar{c}(\,x^*\,) = z(\,\bar{p}\,) \leq z(\,p\,) \leq c(\,x^*\,) = \bar{c}(\,x^*\,)$$

ossia x^* fornisce contemporaneamente sia una valutazione inferiore che superiore del valore ottimo, e le due coincidono. Quando ciò non accade, la valutazione inferiore $z(\,\bar{p}\,)$ e la soluzione x^* del rilassamento possono comunque essere sfruttate per risolvere il problema (P). Infatti, combinando opportunamente euristiche e rilassamenti è possibile realizzare algoritmi esatti, anche se di complessità esponenziale al caso pessimo, per molti problemi di ottimizzazione; si veda il Capitolo 7.

1.2 Tecniche di Modellazione

La costruzione di un modello analitico di un sistema reale è una delle attività più creative e certamente più utili nello studio di sistemi organizzativi e gestionali, nella progettazione industriale, nella descrizione di sistemi altamente complessi quali quelli informatici ed in molte altre discipline. In quanto

attività creativa, la costruzione di modelli non può affidarsi solamente all'uso di tecniche standard; non esistono cioè metodologie formali in grado di costruire automaticamente un modello analitico, anche se esistono tecniche e strumenti software in grado di facilitare ed automatizzare alcune fasi del processo di modellazione. La costruzione di un modello è comunque lasciata fondamentalmente alla creatività ed all'esperienza del singolo, il quale può certamente fare riferimento ai modelli che ha incontrato precedentemente cercando di adattarli ove possibile, ma può anche trovarsi nella necessità di crearne di interamente nuovi.

Come abbiamo discusso, nella costruzione di modelli (in generale, ed in particolare di ottimizzazione) è sempre presente la necessità di bilanciare accuratamente due esigenze contrastanti: da una parte il modello deve essere *significativo*, ossia rappresentare in maniera sufficientemente accurata la realtà, ma dall'altra deve essere *effettivo*, ossia deve permettere di ottenere risultati in tempi compatibili con quelli del processo decisionale.

In generale, una data situazione reale può essere modellata in molti modi diversi, a seconda di quali aspetti della realtà vengono inseriti nel modello e di quali assunzioni (spesso semplificate) si fanno sulle relazioni che li legano. Tali scelte possono influenzare in modo spesso drastico la difficoltà del problema di ottimizzazione che si ottiene; come vedremo in molteplici occasioni, accade sovente che modifiche apparentemente minori ad un problema di ottimizzazione lo rendano significativamente più facile o difficile da risolvere. Anche una volta specificate con precisione le proprietà che il modello deve avere, la decisione sulla *forma* del problema può essere non banale: esistono molti modi diversi per specificare lo stesso insieme di enti matematici, ma alcuni sono più utili di altri al fine di sviluppare approcci algoritmici. I motivi specifici per cui alcuni modelli sono "migliori" di altri sono non banali, e verranno discussi al momento opportuno.

In queste note viene effettuata una scelta precisa sulla forma dei modelli utilizzati: essi sono tutti problemi di *Programmazione Lineare Intera* (PLI). La giustificazione di tale scelta risiede sostanzialmente nella combinazione di due fatti:

1. come vedremo, tale classe di problemi possiede una notevole "espressività" che consente di modellare moltissimi problemi di interesse pratico;
2. d'altra parte, le assunzioni restrittive sulla forma del modello permettono di utilizzare alcune delle tecniche algoritmiche attualmente ritenute più efficienti per la soluzione di problemi $\mathcal{NP}$-ardui.

È opportuno sottolineare da subito che questa scelta è in qualche modo arbitraria, e fondamentalmente dettata dal fatto che queste note si intendono per un corso di base di Ricerca Operativa. Esistono molti casi in cui si utilizzano modelli di tipo diverso. Ad esempio, esistono molte situazioni pratiche in cui non è appropriato (o possibile) evitare di utilizzare relazioni nonlineari nel modello, ma d'altra parte esistono forme di problemi di ottimizzazione che consentono l'uso di relazioni nonlineari e che sono comunque "facili". Infine, esistono altri formalismi che hanno proprietà espressive analoghe a quella della PLI e per i quali esistono approcci algoritmici di analoga sofisticazione e potenza, ad esempio la *programmazione logica con vincoli*. Si può però affermare che la PLI fornisce un valido compromesso tra semplicità, potenza espressiva e disponibilità di solutori (per quanto possibile) efficienti per la sua risoluzione. Inoltre, molti dei concetti algoritmici utilizzati per risolvere problemi più generali sono presenti, almeno in forma semplificata, all'interno degli algoritmi per la PLI, quali quelli discussi nei Capitoli 5, 6 e 7; pertanto, questa classe di problemi fornisce quindi un adeguato "banco di prova" per lo studio degli approcci risolutivi per problemi $\mathcal{NP}$-ardui.

Una volta fissate le regole di base per la costruzione dei modelli, in questo caso l'uso della Programmazione Lineare Intera (PLI), la modellazione è fondamentalmente lasciata alla capacità umana di determinare relazioni tra enti molto diversi quali un frammento di realtà e specifici elementi di un ente matematico. Per quanto questo processo non sia quindi riconducibile ad un insieme di regole fisse meccanicamente applicabili, è certamente possibile fornire una lista di alcune "tecniche base" che consentono di affrontare con successo un gran numero di situazioni diverse. In questo paragrafo forniamo quindi una breve introduzione alla modellazione di problemi di ottimizzazione sotto forma

di *PLI* presentando un insieme di semplici modelli di alcuni noti problemi di ottimizzazione che si incontrano frequentemente nelle applicazioni, o direttamente o—più spesso—come sottoproblemi di problemi più complessi. Questi modelli contengono diversi "blocchi di base" che "codificano" parti di problema che si incontrano in moltissime applicazioni; componendo in modo opportuno questi blocchi si riescono a scrivere modelli di sistemi molto complessi.

1.2.1 Programmazione Lineare

Tra le funzioni multivariate $f : \mathbb{R}^n \to \mathbb{R}$, quelle *lineari* sono certamente le più semplici. Un funzione $f(x)$ si dice lineare se rispetta le proprietà i) $f(x+y) = f(x) + f(y)$ comunque scelti x e y in $\mathbb{R}^n$, e ii) $f(\alpha x) = \alpha f(x)$ comunque scelti x in $\mathbb{R}^n$ ed $\alpha \in \mathbb{R}$. Si può mostrare $f(x)$ è lineare se e solo se ha la forma $f(x) = \sum_{i=1}^{n} a_i x_i$ per n costanti date $a_i \in \mathbb{R}$; quindi, l'insieme delle funzioni lineari su $\mathbb{R}^n$ è isomorfo ad $\mathbb{R}^n$ stesso. In altri termini, $f(x) = ax$, denotando con ciò il *prodotto scalare* tra il vettore (costante) $a \in \mathbb{R}^n$ ed il vettore (variabile) $x \in \mathbb{R}^n$. Se $f(x) = ax$ è lineare, allora $g(x) = ax + c$, per una qualsiasi fissata $c \in \mathbb{R}$, è detta *lineare affine*. Spesso, con un piccolo abuso di notazione, ci si riferisce anche alle funzioni lineari affini semplicemente come lineari.

Un problema di *Programmazione Lineare* (*PL*) è un problema di ottimizzazione caratterizzato da:

1. un numero *finito* n di variabili, che possono assumere valori reali; in altri termini, il vettore $[x_i]_{i=1}^n = x \in \mathbb{R}^n$ delle variabili;
2. una funzione obiettivo lineare $f(x) = cx$, dove $c \in \mathbb{R}^n$ è il *vettore dei costi* (fissato);
3. un insieme ammissibile definito da un insieme finito di m *vincoli lineari* del tipo $ax = b$, e/o $ax \leq b$, e/o $ax \geq b$, dove $a \in \mathbb{R}^n$ e $b \in \mathbb{R}$.

I problemi di *PL* formano una delle più importanti classi di modelli di ottimizzazione, poiché permettono di modellare con sufficiente accuratezza molte situazioni reali e possono essere risolti efficientemente in generale, come mostrato nel Capitolo 3. Introduciamo ora alcuni esempi di problemi di *PL*.

Esempio 1.3. Pianificazione della produzione

La società *Pintel* deve pianificare la produzione della sua fabbrica di microprocessori. La Pintel possiede due diverse linee di prodotti: i processori *Pintium*, più potenti e destinati al mercato "server", ed i *Coloron*, meno potenti e destinati al mercato "consumer". L'impianto è in grado di realizzare 3000 "wafer" alla settimana: su ogni wafer trovano posto o 500 Coloron oppure 300 Pintium. La resa di un wafer dipende anch'essa dal tipo di processore: i Coloron, di minori dimensioni, hanno una resa media del 60%, mentre i Pintium, più grandi e quindi maggiormente sottoposti a difetti, solamente del 50%. I processori Pintium si vendono a 500\$ al pezzo, mentre i Coloron si vendono a 200\$ al pezzo. La divisione commerciale della Pintel ha anche stabilito che la massima quantità di processori che possono essere messi sul mercato ogni settimana senza causare un calo dei prezzi è di 400000 unità per i Pintium e di 700000 unità per i Coloron. Si vuole determinare le quantità di ciascun tipo di processore da produrre settimanalmente in modo da massimizzare il ricavo totale.

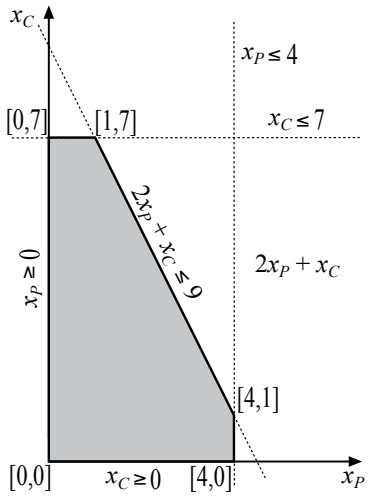


Figura 1.1: Insieme ammissibile per il problema della Pintel

Formuliamo, matematicamente il problema. A questo scopo introduciamo le variabili x_P ed x_C, che indicano il numero di processori rispettivamente di Pintium e Coloron da produrre. Dobbiamo innanzitutto porre le seguenti limitazioni superiori ed inferiori sul valore delle variabili

$$0 \leq x_P \leq 400000 \quad , \quad 0 \leq x_C \leq 700000$$

che corrispondono al fatto che non è possibile produrre quantità negative di processori, ed alle limitazioni imposte dalla divisione commerciale per impedire un calo dei prezzi.

Restano da considerare le limitazioni legate al processo produttivo. Per esprimerle, indichiamo con w_P e w_C il numero di wafer utilizzati per produrre rispettivamente i Pintium e i Coloron; il seguente vincolo lega queste due variabili alla produzione settimanale

$$w_P + w_C \leq 3000 \ .$$

Conoscendo il numero di pezzi per wafer e la resa produttiva si ottengono queste relazioni che legano il numero di

pezzi processori funzionanti con il numero di wafer prodotti:

$$x_P = w_P \cdot 300 \cdot 0.5 = w_P \cdot 150 \quad , \quad x_C = w_C \cdot 500 \cdot 0.6 = w_C \cdot 300 \ . \tag{1.10}$$

Eliminando le due variabili w_P e w_C dal vincolo sulla produzione settimanale si ottiene il vincolo

$$2x_P + x_C \leq 900000 \ .$$

Quindi, l'insieme ammissibile per il problema è

$$F = \{ [\, x_P \,,\, x_C \,] \ : \ 0 \leq x_P \leq 400000 \ , \ 0 \leq x_C \leq 700000 \ , \ 2x_P + x_C \leq 900000 \ \} \ .$$

Ad esempio, $[\, x_P \,,\, x_C \,] = [\, 0 \,,\, 0 \,]$ è ovviamente una soluzione ammissibile per il problema. Un ovvio insieme di "supporto" per il modello è $F' = \mathbb{R}^2$, e la soluzione $[\, x_P \,,\, x_C \,] = [\, 400000 \,,\, 700000 \,]$ non è ammissibile per il problema in quanto viola il vincolo sulla capacità di produzione dei wafers.

Il ricavo ottenuto dalla vendita della produzione settimanale è dato dalla seguente funzione (lineare) nelle variabili decisionali del problema:

$$c(\, x_P \,,\, x_C \,) = 500x_P + 200x_C \ .$$

Di conseguenza, un modello analitico per il problema della Pintel è il seguente:

$$\begin{array}{lrcrcl} \max & 500x_P & + & 200x_C & & \\ & x_P & & & \leq & 400000 \\ & & & x_C & \leq & 700000 \\ & 2x_P & + & x_C & \leq & 900000 \\ & x_P & , & x_C & \geq & 0 \end{array} \tag{1.11}$$

Una volta determinata la soluzione $[\, x_P \,,\, x_C \,]$, ossia in termini di processori, è possibile ricavare la soluzione $[\, w_P \,,\, w_C \,]$, ossia in termini di wafers, semplicemente utilizzando le relazioni (1.10).

Esercizio 1.1. Fornire tre diverse soluzioni ammissibili e valutare i ricavi che si ottengono.

L'insieme ammissibile del problema della Pintel è mostrato in Figura 1.1 (entrambi gli assi sono stati scalati di un fattore 100000 per comodità di rappresentazione). L'insieme ammissibile è il *poliedro convesso* tratteggiato, delimitato dalle rette corrispondenti ai vincoli del problema. È possibile verificare che, per la linearità della funzione obiettivo, una soluzione ottima del problema (se esiste) si trova sempre in un *vertice* del poliedro, in questo caso quello corrispondente al punto $[\, 4 \,,\, 1 \,]$.

Un aspetto che resta ancora da discutere è il dominio ammissibile per la variabili w_P e w_C. Tali variabili rappresentano la produzione di unità discrete di bene (i singoli wafers), e quindi sarebbe ragionevole imporre l'ulteriore vincolo che, nella soluzione attesa del problema, esse possano assumere soltanto valori interi. Si noti che imporre l'interezza di w_P e w_C *non* è equivalente ad imporre l'interezza di x_P e x_C; la soluzione ottima del problema è $[\, x_P \,,\, x_C \,] = [\, 400000 \,,\, 100000 \,]$, quindi intera, ma tale soluzione corrisponde, in termini di wafers, a $[\, w_P \,,\, w_C \,] = [\, 2666.\overline{6} \,,\, 333.\overline{3} \,]$. D'altro canto, i dati numerici del problema sono previsioni/misurazioni di fenomeni reali e quindi necessariamente soggetti ad incertezza; date le quantità in gioco si può ragionevolmente ipotizzare che una differenza di un wafer non cambi sostanzialmente la qualità della soluzione in pratica, e pertanto che la soluzione ammissibile intera $[\, w'_P \,,\, w'_C \,] = [\, 2666 \,,\, 334 \,]$, che corrisponde a $[\, x'_P \,,\, x'_C \,] = [\, 399900 \,,\, 100200 \,]$ sia comunque una "buona" decisione in pratica.

In effetti è possibile quantificare il concetto di "buona soluzione". Il valore della funzione obiettivo corrispondente a $[\, x_P \,,\, x_C \,] = [\, 400000 \,,\, 100000 \,]$ è pari a 220000000\$. È facile capire che tale valore è una *valutazione superiore* corretta sul massimo ricavo che sarà mai possibile ottenere dalla vendita dei microprocessori, qualsiasi sia la configurazione scelta. Infatti, a qualsiasi soluzione $[\, w_P \,,\, w_C \,]$ intera ammissibile deve corrispondere una soluzione $[\, x_P \,,\, x_C \,]$ che soddisfa (1.11), mentre come abbiamo appena visto il viceversa non è vero. Di conseguenza, solo un sottoinsieme delle soluzioni di (1.11) corrisponde effettivamente a soluzioni $[\, w_P \,,\, w_C \,]$ intere ammissibili. Poiché quella determinata è la migliore tra tutte le soluzioni di (1.11), certamente il suo valore non può essere inferiore a quello della migliore soluzione che rispetta anche i vincoli di integralità sui wafers; in altri termini, (1.11) è un *rilassamento* (continuo) del problema reale (cf. (1.8)), e pertanto il suo valore ottimo può essere usato come in (1.9) per stimare l'errore di qualsiasi soluzione ammissibile. Il valore di funzione obiettivo di $[\, x'_P \,,\, x'_C \,] = [\, 399900 \,,\, 100200 \,]$, che corrisponde a $[\, w'_P \,,\, w'_C \,] = [\, 2666 \,,\, 334 \,]$, è 219990000\$; in altri termini la soluzione ammissibile può al caso pessimo dare un ricavo inferiore di 10000\$ rispetto a quella ottima, il che porta ad una stima (superiore) sull'errore relativo di $10000/219990000 \approx 0.000045$, ossia dello 0.0045%.

Questo errore deve essere posto nel contesto dell'*incertezza* relativa ai coefficienti numerici del problema: facilmente in una settimana si produrranno un pò più o un pò meno di 3000 wafers, le rese medie saranno leggermente superiori o inferiori a quelle previste, si riusciranno a vendere a prezzo pieno un pò più o un pò meno del numero stimato di processori, e così via. Tutte queste incertezze possono spesso essere stimate, tipicamente sulla base di statistiche relative al funzionamento del sistema nel passato, e tali stime possono essere trasformate in una valutazione di quanto al massimo possa variare il valore ottimo del problema; si veda il §3.3.4. Se, come è probabile, il massimo

errore stimato per la soluzione $[\, w'_P \,,\, w'_C \,] = [\, 2666 \,,\, 334 \,]$ è almeno comparabile a quello inerente nell'incertezza dei dati (ed è probabile che sia ad esso molto inferiore), allora la soluzione può essere ritenuta ottima a tutti gli effetti pratici.

In effetti l'incertezza nei dati in input del problema è un fenomeno molto diffuso nelle applicazioni pratiche, che in molti casi deve essere gestito in modo appropriato affinché le decisioni prese, ottime solo per il *caso nominale* in cui i dati siano proprio quelli stimati, non si rivelino disastrose qualora i dati siano diversi. Questo porta allo studio delle classi dei problemi di *ottimizzazione stocastica* ed *ottimizzazione robusta*, la cui trattazione però non può trovare spazio in queste dispense.

Quello mostrato è un (semplice) esempio della classe dei problemi di *product mix*, nei quali si vuole decidere la quantità da produrre/offrire di un certo numero di prodotti/servizi diversi, il che comporta il consumo (in quantità variabili) di un certo insieme di risorse. Tali problemi mirano a massimizzare il ricavo proveniente dai prodotti/servizi scelti, eventualmente considerando il costo delle risorse utilizzate. Solo uno dei due aspetti (ricavo e costo) può essere considerato un dato modello, come mostra il confronto tra l'esempio precedente ed il successivo.

Esempio 1.4. Il problema della Fonderia

Una fonderia deve produrre 1000 pezzi del peso ciascuno di un chilogrammo. Il ferro con cui tali pezzi sono fatti dovrà contenere manganese e silicio nelle seguenti quantità:

$$3.25\% \leq \text{silicio} \leq 5.5\% \quad , \quad 0.45\% \leq \text{manganese} \quad .$$

Sono disponibili tre tipi di materiale ferroso con le seguenti caratteristiche:

Materiale ferroso	A	B	C
Silicio (%)	4.00	1.00	0.60
Manganese (%)	0.45	0.50	0.40
Costo (€/ kg.)	0.025	0.030	0.018

Inoltre si può aggiungere direttamente manganese al costo di 10 €al kg. Il problema che si vuole modellare è quello di determinare il piano di produzione che minimizza il costo del materiale utilizzato. Si vuole cioè individuare le quantità di materiale per ciascuno dei tre tipi A, B, o C e di manganese puro da acquistare per produrre i 1000 pezzi richiesti, spendendo il meno possibile.

Per costruire un modello analitico per il problema introduciamo le variabili x_1, x_2, x_3, x_4 con il seguente significato:

$x_1 (\geq 0)$: la quantità in kg di materiale ferroso A da utilizzare;
$x_2 (\geq 0)$: la quantità in kg di materiale ferroso B da utilizzare;
$x_3 (\geq 0)$: la quantità in kg di materiale ferroso C da utilizzare;
$x_4 (\geq 0)$: la quantità in kg di manganese da utilizzare.

Abbiamo imposto che le quantità di prodotto acquistate siano dei valori non negativi (vincoli di non negatività). Esistono poi altri vincoli che dovranno essere rispettati e che descriviamo di seguito. Il numero totale di kg prodotti deve essere 1000:

$$x_1 + x_2 + x_3 + x_4 = 1000 \, .$$

La quantità di silicio, in kg, presente nel prodotto risultante è data da

$$0.04x_1 + 0.01x_2 + 0.006x_3 \, .$$

La percentuale di silicio nel prodotto finito sarà quindi data da

$$100 \frac{0.04x_1 + 0.01x_2 + 0.006x_3}{1000} \, .$$

Tale numero deve essere compreso tra 3.25 e 5.5. Possiamo quindi esprimere la condizione sulla percentuale di silicio mediante i due vincoli lineari

$$4x_1 + x_2 + 0.6x_3 \geq 3250 \quad , \quad 4x_1 + x_2 + 0.6x_3 \leq 5500 \, .$$

Analogamente, per la condizione sulla percentuale di manganese si ottiene

$$0.45x_1 + 0.5x_2 + 0.4x_3 + 100x_4 \geq 450 \, .$$

Infine il costo del prodotto risultante è

$$0.025x_1 + 0.030x_2 + 0.018x_3 + 10x_4 \, .$$

Il problema della determinazione di un piano di produzione che minimizza il costo può quindi essere formulato come segue:

$$\begin{array}{rrcrcrcrcl} \min & 0.025x_1 & + & 0.030x_2 & + & 0.018x_3 & + & 10x_4 & & \\ & 4x_1 & + & x_2 & + & 0.6x_3 & & & \geq & 3250 \\ & 4x_1 & + & x_2 & + & 0.6x_3 & & & \leq & 5500 \\ & 0.45x_1 & + & 0.5x_2 & + & 0.4x_3 & + & 100x_4 & \geq & 450 \\ & x_1 & + & x_2 & + & x_3 & + & x_4 & = & 1000 \\ & x_1 & , & x_2 & , & x_3 & , & x_4 & \geq & 0 \end{array}$$

Le variabili x_1, x_2, x_3 e x_4 corrispondono alle scelte operative che il problema reale richiede di compiere, e ciascun vincolo del modello corrisponde ad una condizione imposta dal problema reale. Determinare i valori delle variabili in modo che i vincoli siano soddisfatti e la funzione obiettivo assuma il minimo valore fornisce il miglior piano di produzione.

I problemi di questo tipo possono differire in modo significativo per il dettaglio dei vincoli considerati, che dipendono dalla situazione modellata, ma presentano una struttura in gran parte analoga. Tali problemi si possono spesso modellare come *PL* in quanto le relazioni tra input ed output (risorse e prodotti) sono lineari (o possono essere ragionevolmente approssimate come tali) in molti ambiti applicativi. Ciò però non è vero in tutti gli ambiti, e porta a dover considerare modelli *nonlineari*. In queste dispense non tratteremo tali classi di problemi, ma nel seguito mostreremo come si possa spesso convenientemente approssimare relazioni non lineari mediante quelle lineari.

1.2.2 Variabili logiche

Negli esempi precedenti abbiamo incontrato variabili che rappresentano il livello di attività produttive, come il numero di processori da produrre nel'Esempio 1.3 oppure il numero di pezzi da produrre nella fonderia nell'Esempio 1.4. Variabili di questo tipo vengono anche dette *variabili quantitative*; ciò che le caratterizza è il fatto di rappresentare quantità di beni, di oggetti da produrre, comprare o utilizzare, oppure valori assunti da grandezze fisiche o economiche. In altri casi le variabili rappresentano delle scelte di tipo logico, ossia "vero" o "falso"; si parla allora di *variabili logiche*. Per esprimere questo tipo si scelta si utilizzano usualmente *variabili booleane*, o *binarie*, che possono assumere uno solo dei due valori numerici 0 o 1, attribuendo al valore 0 il significato di *falso* ed a 1 quello di *vero*.

È importante sottolineare come l'uso di variabili binarie, o più in generale di variabili vincolate ad assumere solamente un insieme discreto di valori, se da un lato aumenta l'espressività dei modelli di ottimizzazione, dall'altro ne può rendere molto difficile la soluzione. Si definiscono problemi di *Programmazione Lineare Intera* (*PLI*), o *Mista*, i problemi in cui i vincoli e la funzione obiettivo sono lineari, come nel caso della *PL*, ma in cui tutte le variabili, o un loro sottoinsieme, possono assumere solamente valori interi. Mentre la *PL* è un problema "facile", la *PLI* è in generale un problema "difficile" (si veda l'Appendice A).

Vediamo adesso alcuni esempi di modelli che usano variabili logiche.

Esempio 1.5. Il problema dello zaino

Il famoso ladro internazionale Arsenico Lupon è nella fase finale del colpo del secolo, che lo consacrerà a gloria imperitura come miglior ladro di tutti i tempi: è riuscito ad entrare indisturbato nel Louvre, dal quale può rubare tutto quello che vuole. O meglio, tutto quello che può: ha nascosto un furgone nel seminterrato, e potrà portare via solamente quello che riesce a farci entrare. Ha quindi catalogato tutti i principali n reperti del Louvre, assegnando a ciascuno di essi un valore $c_i > 0$ (che tiene conto sia del valore monetario quando lo rivenderà al mercato nero che del contributo che darà alla sua leggenda), ed una misura $a_i > 0$ del volume che occuperà nel furgone (si suppone che la forma esatta non sia rilevante), che ha un volume disponibile pari a $b > 0$. Poichè $b < \sum_i a_i$, ossia non può rubare tutto quello che vorrebbe, deve decidere quale sottoinsieme dei reperti prendere, e di conseguenza quale lasciare. Ovviamente stà considerando solamente reperti che può effettivamente trasportare, ossia per cui vale $a_i \leq b$; gli altri li ha scartati a priori. Il problema è quindi determinare il sottoinsieme ammissibile di reperti che massimizza il valore (monetario e di reputazione) totale, dato semplicemente dalla somma dei valori dei reperti trafugati.

Questo problema è noto come problema dello zaino (KP, da *Knapsack Problem*). Per modellarlo introduciamo l'insieme $I = \{1, 2, \ldots, n\}$ degli indici degli elementi tra cui scegliere (reperti), il che porta immediatamente alla definizione combinatoria

$$c(S) = \textstyle\sum_{i \in S} c_i \quad , \quad F = \left\{ S \subseteq I : \sum_{i \in S} a_i \leq b \right\} .$$

In questa descrizione del problema, il sottoinsieme S è l'unica variabile, che prende valori nell'insieme delle parti di

I (2^I) ed è soggetta al vincolo $\sum_{i \in S} a_i \leq b$.

Possiamo riformulare il problema come *PLI* introducendo, per ogni oggetto $i \in I$, una variabile binaria $x_i \in \{ 0 , 1 \}$, con il significato che la variabile assume valore 1 se l'elemento i-esimo appartiene al sottoinsieme selezionato, e 0 altrimenti (si decide cioè se trafugare o meno il reperto). La condizione relativa alla capacità dello zaino diviene

$$\sum_{i \in I} a_i x_i \leq b \ ;$$

infatti, dato che ciascuna x_i può assumere solo i valori 0 o 1, nella somma vengono considerati i volumi dei soli oggetti selezionati. Analogamente, la funzione obiettivo, da massimizzare, è

$$c(x) = \sum_{i \in I} c_i x_i$$

in quanto nella funzione obiettivo si sommano i costi dei soli oggetti selezionati. La formulazione che si ottiene è quindi

$$\text{(KP)} \qquad \max \left\{ \sum_{i \in I} c_i x_i \ : \ \sum_{i \in I} a_i x_i \leq b \ , \ x_i \in \{ 0 , 1 \} \quad i \in I \right\} \ .$$

Il problema dello zaino si trova in letteratura anche descritto come un problema di minimo con vincolo di $\geq$.

Esercizio 1.2. Costruire un'istanza del problema dello zaino con 6 oggetti, definendone costo e peso; formulare quindi l'istanza, fornire due soluzioni ammissibili e valutarne il costo.

Le variabili logiche permettono quindi di selezionare sottoinsiemi di un insieme dato. Con i vincoli opportuni è possibile descrivere insiemi con strutture arbitrariamente complesse.

Esempio 1.6. Albero di copertura di costo minimo

La banca Gatto & Volpe ha molte filiali sparse per l'Italia ed un Centro Elettronico Unificato (CEU) in cui vengono svolte tutte le transazioni. La banca ha bisogno di collegare tutte le filiali al CEU: per ragioni di sicurezza i dati non possono transitare sulla rete pubblica, e quindi occorre affittare linee dedicate. È possibile affittare una linea dedicata dal CEU ad ogni filiale, ma, se la capacità delle linee è sufficientemente grande, ciò non è necessario: può essere più conveniente collegare gruppi di filiali "vicine" tra loro, e solo una di esse al CEU. Il problema è quindi determinare quali linee affittare (nell'ipotesi che la capacità delle linee sia sufficiente) per collegare tutte le filiali al CEU minimizzando il costo di affitto.

Per modellare il problema, si può considerare un grafo non orientato (V , E), dove V è l'insieme dei vertici con $|V| = n$, E è l'insieme dei lati con $|E| = m$ e ad ogni lato $\{ i , j \} \in E$ è associato un costo $c_{ij} > 0$ (per maggiori dettagli sui grafi, si rinvia all'Appendice B). Nel grafo, ogni nodo rappresenta una filiale, tranne un nodo "radice" che rappresenta il CEU, e gli archi rappresentano i potenziali collegamenti tra coppie di filiali o tra filiali e CEU, con il relativo costo. Il problema del progetto della rete dati della banca può quindi essere formulato matematicamente come il problema di determinare un grafo parziale $G' = (V , E')$, connesso e di costo minimo, dove il costo di G' è dato dalla somma dei costi dei lati in E'. È facile verificare che la struttura di collegamento ottimale (qualora la capacità delle linee sia sufficiente) è un albero di copertura di G, cioè un grafo parziale di G che sia connesso e privo di cicli. Il problema del progetto della rete dati della banca corrisponde quindi al problema di ottimizzazione con

$$c(T) = \sum_{\{ i , j \} \in T} c_{ij} \quad , \quad F = \left\{ T \subseteq E \ : \ (N , T) \text{ è un albero di copertura per } G \right\} \ .$$

Possiamo formulare come *PLI* il problema di determinare un *albero di copertura di costo minimo* (MST, da *Minimal Spanning Tree*) introducendo, per ogni lato $\{ i , j \} \in E$, una variabile binaria x_{ij} che assume il valore 1 se il lato $\{ i , j \}$ viene selezionato per formare l'albero di copertura e 0 se il lato non viene selezionato. Affinché l'insieme dei lati selezionati formi un grafo parziale connesso, è necessario e sufficiente che, per ogni sottoinsieme S dei vertici (non vuoto e non coincidente con V) vi sia almeno un lato selezionato che ha un estremo in S e l'altro in $V \setminus S$ (ossia sia selezionato almeno uno degli archi del *taglio* $(S , N \setminus S)$; per maggiori dettagli si veda l'Appendice B). Pertanto, imponendo i seguenti *vincoli di connessione*

$$\sum_{i \in S , j \notin S} x_{ij} \geq 1 \qquad \emptyset \subset S \subset V \tag{1.12}$$

si garantisce che i valori assunti dalle variabili decisonali definiscano un grafo parziale connesso. La funzione obiettivo, da minimizzare, è

$$\sum_{\{ i , j \} \in E} c_{ij} x_{ij} \ , \tag{1.13}$$

pertanto una formulazione *PLI* per il problema è la seguente:

$$\text{(MST)} \qquad \min \left\{ \sum_{\{ i , j \} \in E} c_{ij} x_{ij} \ : \ \sum_{i \in S , j \notin S} x_{ij} \geq 1 \quad \emptyset \subset S \subset V \ , \ x_{ij} \in \{ 0 , 1 \} \quad \{ i , j \} \in E \right\} \ .$$

Nell'ipotesi che i pesi siano strettamente positivi, è facile verificare che qualunque soluzione ottima x^* del problema definisce un albero di copertura di costo minimo. Infatti, qualunque soluzione ammissibile rappresenta un sottografo connesso di G, che quindi contiene sicuramente un albero di copertura. Se per assurdo la soluzione rappresentasse un sottografo con più di $n - 1$ lati, ossia non fosse un albero, allora tale sottografo conterrebbe almeno un ciclo: eliminando un qualsiasi lato di questo ciclo si otterrebbe ancora un grafo connesso, il cui vettore di incidenza $\bar{x}$

sarebbe una soluzione ammissibile per il problema con costo strettamente minore di x^*, il che contraddice l'ottimalità di quest'ultima.

Questa formulazione "basata sui tagli" ha un numero esponenziale di vincoli, uno per ogni possibile sottoinsieme proprio di V. Esistono altre formulazioni con un numero polinomiale di vincoli. In generale, come vedremo nel seguito, formulazioni diverse dello stesso problema possono essere utili in quanto possono suggerire approcci algoritmici diversi.

Esercizio 1.3. Costruire un'istanza del problema dell'albero di copertura di costo minimo per un grafo con 4 vertici e 6 lati, definendo i pesi dei lati; formulare quindi l'istanza utilizzando i vincoli sui tagli, fornire due soluzioni ammissibili e valutarne il costo.

Una volta che si sia costruito un modello che rappresenta un certo insieme (complesso), è tipicamente possibile utilizzarlo come base per costruire modelli che rappresentano insiemi (ancora più complessi) che sono sottoinsiemi dell'insieme iniziale.

Esempio 1.7. Il problema del commesso viaggiatore

Il Presidente (fantoccio) della Galassia, Zaphod Beelebrox, sta pianificando il suo giro della Galassia per cercare di essere ri-eletto. Nonostante la prodigiosa velocità della Propulsione a Probabilità Infinita della sua astronave Cuore D'Oro, le distanze siderali delle n capitali dei Distretti Galattici rendono il viaggio lento, e quindi problematico (dato che nei momento di ozio le sue due teste regolarmente iniziano a litigare tra di loro). Ha quindi chiesto a Marvin, l'intelligentissimo (e paranoico) androide che l'accompagna, di organizzare il giro in modo da minimizzare il tempo totale di viaggio. Marvin concettualizza il problema attraverso un grafo non orientato e completo (cioè contenente tutti i possibili lati) $G = (V, E)$, dove V è l'insieme dei vertici (capitali dei Distretti Galattici), con $|V| = n$, ed E è l'insieme dei lati $\{i, j\}$ ($|E| = m = n(n-1)/2$), a ciascuno dei quali è associato il costo $c_{ij} > 0$ che rappresenta il tempo di volo tra le località associate ai vertici i e j. Il piano di viaggio della Cuore D'Oro, ossia visitare in sequenza le n località rientrando alla località di partenza, corrisponde ad un *ciclo Hamiltoniano* sul grafo G, cioè ad una sequenza di lati che inizia e termina nello stesso vertice e che include ogni altro vertice una ed una sola volta. Il costo (lunghezza) del ciclo Hamiltoniano è la somma dei costi dei suoi lati; quello che Martin deve determinare è quindi il meno costoso (più corto) tra tutti i cicli Hamiltoniani.

Il *problema del ciclo Hamiltoniano di lunghezza minima* è noto come il *problema del commesso viaggiatore* (TSP, da *Travelling Salesman Problem*) ed ha molte applicazioni pratiche, tipicamente collegate a problemi di trasporto, ma non solo: ad esempio, problemi analoghi si ritrovano nel servire in modo ottimale n richieste di lettura/scrittura su uno stesso disco magnetico (in ambiente parallelo o concorrente) minimizzando il ritardo dovuto ai movimenti della testina. Si osservi che l'avere supposto il grafo completo non comporta alcuna limitazione; infatti, è sempre possibile rendere completo un grafo con l'aggiunta di lati a costo opportunamente elevato ($+\infty$).

(TSP) è un problema di ottimizzazione in cui l'insieme ammissibile F è l'insieme di tutti i cicli Hamiltoniani del grafo G, e la funzione obiettivo $c(P)$ è la lunghezza del ciclo Hamiltoniano P. La sua versione decisionale richiede di determinare se il grafo G ha un ciclo Hamiltoniano di lunghezza non superiore ad un prefissato valore k.

Per definire una formulazione *PLI* di (TSP) utilizziamo, analogamente a quanto fatto per (MST), una variabile logica x_{ij} per ogni lato $\{i, j\} \in E$, che vale 1 se $\{i, j\}$ appartiene al ciclo scelto e 0 altrimenti; la funzione obiettivo, da minimizzare, è allora la stessa (1.13) di (MST). Il fatto che si voglia ottenere un ciclo comporta che in ciascun nodo incidano esattamente due archi, proprietà che può essere imposta per mezzo dei vincoli:

$$\sum_{\{i,j\} \in E} x_{ij} = 2 \quad i \in V \qquad , \qquad x_{ij} \in \{0, 1\} \quad \{i, j\} \in E \ . \tag{1.14}$$

Si noti come il vincolo di integralità sulle variabili sia critico per garantire che i vincoli (1.14) assicurino che nel sottografo rappresentato dalle variabili $x_{ij} > 0$ ciascun vertice abbia esattamente due lati incidenti; ove tale vincolo non fosse imposto il sottografo potrebbe contenere molti lati incidenti nello stesso vertice, "distribuendo" le due unità disponibili frazionalmente su di essi. In generale, si noti che in un modello matematico qualsiasi vincolo logicamente necessario debba essere indicato *esplicitamente*: anche se una condizione può apparire ovvia a chi modella, il modello non ha "alcuna percezione della realtà" e la corrispondenza tra le due entità è completamente a carico del modellatore.

I vincoli (1.14) non garantiscono che i lati $\{i, j\}$ le cui variabili associate x_{ij} assumono il valore 1 formino un ciclo Hamiltoniano; infatti esse garantiscono solo una *copertura per cicli* del grafo come illustrato nell'esempio in Figura 1.2, in cui i lati selezionati formano una copertura di tutti i vertici del grafo mediante due cicli disgiunti, uno di 3 archi e l'altro di 4. Per imporre che i lati selezionati formino un unico ciclo (Hamiltoniano) possiamo utilizzare i vincoli di connessione basati sui tagli (1.12), introdotti per (MST). L'aggiunta dei vincoli di connessione ai vincoli di copertura per cicli (1.14) impone che la copertura per cicli formi un grafo connesso: ciò è possibile solo se si ha un

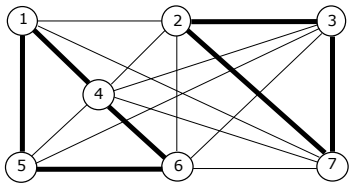


Figura 1.2: una copertura per cicli (archi evidenziati)

unico ciclo nella copertura, che è quindi un ciclo Hamiltoniano. La formulazione completa di (TSP) diventa quindi

$$\text{(TSP)} \qquad \begin{array}{lll} \min & \sum_{\{\,i\,,\,j\,\}\in E} c_{ij}x_{ij} & \\ & \sum_{i\in S\,,\,j\notin S} x_{ij} \geq 1 & \emptyset \subset S \subset V \\ & \sum_{\{\,i\,,\,j\,\}\in V} x_{ij} = 2 & i \in V \\ & x_{ij} \in \{\,0\,,\,1\,\} & \{\,i\,,\,j\,\} \in E \end{array} \quad .$$

Come per (MST), esistono formulazioni per (TSP) aventi un numero polinomiale di vincoli; per esse si rinvia alla bibliografia suggerita.

Esercizio 1.4. Assegnare dei pesi agli archi del grafo in Figura 1.2 supponendo che gli archi mancanti abbiano peso infinito. Costruire quattro cicli Hamiltoniani e valutarne la lunghezza.

Esercizio 1.5. Costruire un'istanza relativa ad un grafo completo con 4 nodi, e formulare (TSP) per tale istanza.

La formulazione di (TSP) coincide con quella di (MST) a cui sono stati aggiunti i vincoli (1.14). Pertanto, l'insieme ammissibile di (MST) contiene quello di (TSP); siccome la funzione obiettivo è identica, (MST) è un *rilassamento* di (TSP). Questa osservazione ha una sua utilità pratica, in quanto mentre (TSP) è un problema $\mathcal{NP}$-arduo, esistono algoritmi polinomiali (e molto efficienti computazionalmente) per risolvere (MST). Inoltre, rimuovendo dalla formulazione i vincoli (1.12), e lasciando quindi solamente i vincoli (1.14), si ottiene un altro rilassamento polinomiale, che può essere risolto attraverso algoritmi per problemi di assegnamento (si veda il Paragrafo 1.10).

1.2.3 Relazioni binarie

Le variabili booleane sono assimilabili alle *variabili proposizionali*, ossia variabili che possono assumere i valori *vero* o *falso*. Spesso le richieste del modello possono essere analizzate in termini delle ben note relazioni logiche tra le variabili proposizionali; è quindi molto utile costruire vincoli lineari tra variabili binarie equivalenti alle classiche relazioni logiche del calcolo proposizionale. Nel seguito, dato un letterale (proposizione elementare) a del calcolo proposizionale indicheremo con $x(a)$ la corrispondente variabile booleana, associando il valore 1 di $x(a)$ al valore *vero* di a ed il valore 0 di $x(a)$ al valore *falso* di a. Analizziamo adesso le più comuni relazioni tra variabili proposizionali.

Negazione. Data la variabile proposizionale a, la variabile complementare $b = \neg a$ viene rappresentata facilmente dalla *variabile complementare* $x(b) = 1 - x(a)$, con $x(b) \in \{\,0\,,\,1\,\}$. Se si hanno due variabili proposizionali a e b e si vuole imporre che una sia il complemento dell'altra, è sufficiente imporre alle corrispondenti variabili booleane di rispettare il vincolo $x(a) + x(b) = 1$.

Implicazione. La relazione logica $a \Longrightarrow b$ (a implica b) è esprimibile mediante la disuguaglianza $x(b) \geq x(a)$; infatti, $x(b)$ è forzata ad assumere il valore 1 se $x(a) = 1$.

Unione (Or). Date due variabili proposizionali a e b, la variabile $c = a \vee b$, che assume il valore *vero* quando almeno una delle due variabili è vera, può essere espressa mediante le seguenti relazioni:

$$x(c) \geq x(a) \quad , \quad x(c) \geq x(b) \quad , \quad x(c) \leq x(a) + x(b) \quad , \quad x(c) \in \{\,0\,,\,1\,\} \ .$$

Infatti, le due prime diseguaglianze impongono alla variabile booleana $x(c)$ di assumere il valore 1 se una delle due altre variabili ha il valore 1. La terza impone il valore $x(c) = 0$ se $x(a) = x(b) = 0$.

Unione esclusiva (Or esclusivo). Date due variabili proposizionali a e b, la variabile $c = a \oplus b$, che assume il valore *vero* quando una sola delle due variabili è vera, può essere espressa mediante le seguenti relazioni:

$$\begin{array}{c} x(c) \geq x(a) - x(b) \quad , \quad x(c) \geq x(b) - x(a) \\ x(c) \leq x(a) + x(b) \quad , \quad x(c) \leq 2 - x(a) - x(b) \quad , \quad x(c) \in \{\,0\,,\,1\,\} \end{array} \ .$$

Infatti, le due prime diseguaglianze impongono alla variabile booleana $x(c)$ di assumere il valore 1 quando una sola delle due altre variabili vale 1. La terza impone il valore $x(c) = 0$ se $x(a) = x(b) = 0$ e la quarta impone $x(c) = 0$ se $x(a) = x(b) = 1$.

Intersezione (And). Date due variabili binarie a e b, la variabile $c = a \wedge b$, che assume il valore *vero* solo quando ambedue le variabili sono vere, può essere espressa mediante le seguenti relazioni:

$$x(c) \leq x(a) \quad , \quad x(c) \leq x(b) \quad , \quad x(c) \geq x(a) + x(b) - 1 \quad , \quad x(c) \in \{\, 0 \,,\, 1 \,\} \ .$$

Infatti, le prime due diseguaglianze impongono alla variabile booleana $x(c)$ di assumere il valore 0 quando almeno una delle due altre variabili vale 0. La terza impone $x(c) = 1$ se $x(a) = x(b) = 1$.

In generale, è possibile formulare molti problemi del calcolo proposizionale sotto forma di problemi di ottimizzazione. Questo tipo di formulazione permette di utilizzare tecniche di ottimizzazione in alternativa o in appoggio alle normali tecniche inferenziali usate nel calcolo logico.

Esempio 1.8. Avventure nel calcolo logico

Casciana Jones, famoso avventuriero/archeologo/robivecchi, si trova davanti alla porta della camera del tesoro in una piramide a gradoni Tolteca. Sulla porta ci sono quattro leve, contrassegnate dal sole, dalla luna, da una montagna e da un fiume, insieme alle seguenti misteriose iscrizioni:

- il sole e la luna non possono risplendere insieme;
- il fiume può scorrere solamente se almeno uno tra il sole e la luna risplendono;
- quando la montagna trema ed il fiume scorre, il sole non può risplendere;
- l'occhio di Quezalcotl solo potrai rimirare quando la montagna trema ed il fiume scorre.

Casciana intuisce che spostare la leva del sole e della luna verso l'alto li fa "risplendere", spostare la leva del fiume verso l'alto lo fa "scorrere" e spostare la leva della montagna verso l'alto la fa "tremare". Deve quindi determinare la giusta posizione delle quattro leve per poter entrare nella porta e vedere "l'occhio di Quezalcotl" (uno smeraldo di circa 4 Kg), sapendo che la scelta sbagliata gli costerebbe sicuramente la vita. Casciana non è mai stato bravo con la logica ma ha studiato Ricerca Operativa ed ha sul suo smartphone un solutore di problemi di *PLI*. Si aiuti l'avventuriero/archeologo/straccivendolo a continuare la sua epica avventura descrivendo il problema di *PLI* che deve risolvere per determinare la posizione giusta delle quattro leve.

Introduciamo le variabili binarie x_s per il sole, x_l per la luna, x_f per il fiume ed x_m per la montagna, intendendo che tali variabili prendono valore 1 se la leva corrispondente deve essere spostata verso l'alto (ossia, rispettivamente, se il sole e la luna risplendono, il fiume scorre e la montagna trema). La prima iscrizione può essere tradotta nel vincolo

$$x_s + x_l \leq 1 \ .$$

La seconda iscrizione può essere tradotta nel vincolo

$$x_f \leq x_s + x_l$$

in quanto x_f può assumere valore 1 solo se ha valore 1 almeno uno tra x_s ed x_l. La terza iscrizione può essere tradotta nel vincolo

$$x_s \leq 2 - x_f - x_m$$

in quanto se entrambe x_f ed x_m assumono valore 1, allora x_s deve assumere valore 0. Infine, la quarta iscrizione richiede che

$$x_f + x_m \geq 2$$

o, alternativamente

$$x_f = 1 \qquad\qquad x_m = 1 \ .$$

Inoltre, bisogna porre esplicitamente i vincoli di integralità sulle variabili, ossia

$$x_s \in \{\, 0 \,,\, 1 \,\} \qquad x_l \in \{\, 0 \,,\, 1 \,\} \qquad x_f \in \{\, 0 \,,\, 1 \,\} \qquad x_m \in \{\, 0 \,,\, 1 \,\} \ .$$

I coefficienti della funzione obiettivo possono essere ad esempio posti tutti a zero, in quanto ciò che interessa è una qualsiasi soluzione ammissibile, se esiste. In alternativa, è possibile eliminare l'ultimo vincolo ed usare

$$\max x_f + x_m$$

come funzione obiettivo; chiaramente, esiste una configurazione delle leve che permette di vedere l'occhio di Quezalcotl se e solo se il valore ottimo di tale funzione obiettivo è 2.

Le formule del calcolo logico possono essere trasformate in formule diverse ma equivalenti mediante l'applicazione di opportune regole di equivalenza, tra le quali

$$\neg(a \vee b) \equiv \neg a \wedge \neg b \ \ , \ \ \neg(a \wedge b) \equiv \neg a \vee \neg b \ \ , \ \ a \Longrightarrow b \equiv \neg a \vee b \ \ .$$

Ciò implica la possibilità di portare le formule in alcune *forme normali* particolarmente semplici, tra

le quali una delle più utilizzate è quella *Congiuntiva* (FNC), ossia la congiunzione di m *clausole*

$$F = C_1 \wedge C_2 \wedge \ldots \wedge C_m$$

ciascuna delle quali è la disgiunzione di un certo sottoinsieme di letterali diretti o negati

$$C_i = \pm P_{i_1} \vee \pm P_{i_2} \vee \ldots \vee \pm P_{i_{k(i)}}$$

dove con $\pm P_j$ si indica o il letterale P_j o la sua negazione $\neg P_j$. Qualsiasi formula F del calcolo proposizionale può essere trasformata in una formula F' in FNC (anche se in principio F' potrebbe essere di lunghezza esponenzialmente più grande rispetto ad F, problema che però può essere risolto introducendo nuovi letterali), che risulta particolarmente utile per lo sviluppo di modelli *PLI* equivalenti a problemi logici di particolare rilevanza.

Esempio 1.9. Il problema della Soddisfattibilità Proposizionale

Il problema della *Soddisfattibilità Proposizionale* (SAT, da *SATisfiability*) richiede di determinare se una data formula del calcolo proposizionale F in FNC sia oppure no soddisfattibile. Si vuole cioè determinare se esiste un assegnamento di valore di verità *vero* o $falso$ ai letterali (proposizioni elementari) P_1, P_2, ..., P_n che renda vera la formula F. Tale problema può essere rappresentato come *PLI* in modo particolarmente semplice. Introduciamo gli insiemi $J = \{ 1, 2, \ldots, m \}$ degli indici delle clausole e $I = \{ 1, 2, \ldots, n \}$ degli indici dei letterali, le variabili binarie x_j "naturali" associate ai letterali P_j, $j \in J$, e definiamo

$$a_{ij} = \begin{cases} 1 & \text{se il letterale } P_j \text{ appare diretto nella clausola } C_i \\ -1 & \text{se il letterale } P_j \text{ appare negato nella clausola } C_i \\ 0 & \text{se il letterale } P_j \text{ non appare nella clausola } C_i \end{cases} .$$

Dato un qualunque vettore $x \in \{ 0, 1 \}^n$, che possiamo interpretare come un assegnamento di valori di verità agli n letterali P_1, ..., P_n, è facile verificare che la generica clausola C_i, $i \in I$ è soddisfatta dall'assegnamento di valori di verità corrispondente ad x se e solo se risulta

$$\textstyle\sum_{j \in J} a_{ij} x_j \geq 1 - n(i) \, , \tag{1.15}$$

dove $n(i)$ è il numero di letterali che appaiono negati in C_i. Una formulazione *PLI* di (SAT) è quindi

$$\min \left\{ 0 \, : \, \textstyle\sum_{j \in J} a_{ij} x_j \geq 1 - n(i) \quad i \in I \ , \ x_j \in \{ 0, 1 \} \quad j \in J \right\}$$

Questo è un problema di ottimizzazione con una funzione obiettivo costante: il suo valore ottimo è quindi 0 se esiste almeno una soluzione ammissibile x, ossia un assegnamento di valori di verità alle proposizioni elementari che rende vere tutte le clausole C_1, ..., C_m, e $+\infty$ altrimenti. In effetti (SAT) è un *problema di decisione* piuttosto che non un problema di ottimizzazione: si è interessati solamente a determinare l'esistenza di una qualsiasi soluzione ammissibile, senza distinguere in alcun modo tra di esse. Una variante di (SAT) che è un problema di ottimizzazione è quella nella quale si cerca l'assegnamento di valori di verità ai letterali che massimizza il numero delle clausole C_i soddisfatte (Max-SAT).

Esercizio 1.6. Si dia una formulazione analitica di (Max-SAT).

Esiste pertanto una rilevante interfaccia tra problemi legati al calcolo logico e problemi di ottimizzazione: in effetti, (SAT) è stato il primo problema che è stato dimostrato essere $\mathcal{NP}$-completo (si veda l'Appendice A). Ciò permette di utilizzare tecniche di ottimizzazione per la soluzione di problemi relativi al calcolo logico e, viceversa, tecniche di inferenza per risolvere problemi di ottimizzazione. Esistono persino alcuni interessanti risultati teorici che mostrano come le deduzioni logiche nel calcolo proposizionale possono essere viste come combinazioni lineari dei vincoli nella corrispondente formulazione di *PLI*, e quindi come le tecniche inferenziali siano un caso particolare di alcune tecniche per la risoluzione di problemi di *PLI*, dimostrando come la relazione tra ottimizzazione e calcolo logico sia profonda.

1.2.4 Vincoli di assegnamento e semiassegnamento

Negli esempi precedenti, per ciascun elemento del problema (oggetto, arco, variabile proposizionale, ...) era solamente disponibile una scelta "semplice tra due stati" (selezionato/non selezionato, vero/falso, ...). In molti casi, invece, per un dato elemento sono possibili un certo numero (finito e fissato) di scelte diverse, tra le quali occorre sceglierne una ed una sola. Assumendo che ci siano m scelte possibili, sarebbe naturale pensare di codificare questo in un modello *PLI* attraverso una singola variabile x intera che possa prendere valori tra 0 ed $m-1$, codificando così le m scelte possibili diver-

se. Spesso questa scelta modellistica si rivela non adeguata, in particolare quando sia poi necessario esprimere condizioni (logiche) relativamente alle scelte compiute per più di un elemento diverso. In questo caso, si preferisce esprimere la condizione attraverso l'introduzione di m variabili binarie x_1, x_2, ..., x_m, dove $x_i = 1$ indica che si è presa la scelta i-esima, con il vincolo

$$\sum_{i=1}^{m} x_i = 1$$

che assicura che esattamente una delle scelte possibili sia effettuata. Vediamo adesso alcune applicazioni di questo semplice concetto.

Esempio 1.10. Assegnamento di costo minimo

L'agenzia matrimoniale Cuori Solitari, deve organizzare il gran ballo di fine anno. L'agenzia ha individuato due sottoinsiemi disgiunti di clienti, convenzionalmente indicati come M ed F, entrambi di cardinalità n, in cui ciascun cliente $i \in M$ può potenzialmente essere interessato ad intrecciare una relazione con un cliente $j \in F$. Dai profili psicologici raccolti dai clienti, l'agenzia è in grado di calcolare per ciascun $i \in M$ il sottoinsieme $F(i) \subseteq F$ dei clienti dell'altro insieme con i quali potrebbe essere interessato ad intrecciare una relazione, e che contraccambiano l'interesse. Questo immediatamente definisce l'analogo insieme $M(j) \subseteq M$ per ogni $j \in F$ (tale che $j \in F(i) \iff i \in M(j)$), e l'insieme C delle coppie (i, j) "compatibili". Per ciascuna di esse l'agenzia è anche in grado di calcolare, dai profili dei clienti, il costo c_{ij} della cena da offrire, che deve consistere di piatti graditi ad entrambi i commensali. L'agenzia vuole quindi decidere come formare le coppie per il gran ballo in modo da evitare coppie incompatibili e minimizzare il costo complessivo delle cene.

Una formulazione del problema dell'agenzia Cuori Solitari fa uso di variabili binarie x_{ij} per ciascuna coppia compatibile $(i, j) \in C$. I vincoli devono assicurare che sia scelto esattamente un $j \in F$ per ciascun $i \in M$, e viceversa. Ciò porta immediatamente al modello *PLI*

$$\begin{array}{lll} \min & \sum_{(i,j)\in C} c_{ij}x_{ij} & \\ & \sum_{j\in F(i)} x_{ij} = 1 & i \in M \\ & \sum_{i\in M(j)} x_{ij} = 1 & j \in F \\ & x_{ij} \in \{0, 1\} & (i, j) \in C \end{array}$$

Questo problema, noto come il problema dell'*assegnamento di costo minimo*, è polinomiale ed ha molte applicazioni in pratica; algoritmi per la sua soluzione sono descritti nella Sezione 2.6.

Dato un grafo non orientato G, il problema di determinare una copertura per cicli di costo minimo del grafo (si veda l'Esempio 1.7) può essere formulato come un problema di assegnamento di costo minimo: sia gli oggetti i che gli elementi j corrispondono ai vertici del grafo, e le coppie $\{i, j\}$ "compatibili" corrispondono ai lati del grafo, con il relativo costo. Di conseguenza, il problema dell'assegnamento di costo minimo è un rilassamento di (TSP). Si noti quindi come uno stesso problema possa avere rilassamenti diversi: ad esempio, sia il problema dell'assegnamento di costo minimo che (MST) sono rilassamenti di (TSP). Diversi rilassamenti possono "catturare" parti diverse della struttura di un problema: ad esempio, (MST) incorpora i vincoli di connessione ma non quelli di formazione di cicli, mentre il problema dell'assegnamento incorpora i vincoli di formazione di cicli ma non quelli di connessione. Infine, si noti che sia il problema dell'assegnamento che (MST) sono problemi "facili", mentre (TSP), che può essere considerato "l'intersezione" di questi due problemi, è "difficile". In generale, solo in casi molto particolari l'intersezione di due strutture combinatorie corrispondenti a problemi "facili" genera un problema a sua volta "facile".

Il caso precedente era particolare nel senso che i due insiemi di elementi da assegnare sono della stessa cardinalità, e ci sia simmetria nel senso che ciascun elemento di un insieme debba essere assegnato ad uno ed un solo elemento dell'altro, il che assicura che non ci siano più coppie diverse che condividono uno dei due elementi. La combinazione delle due condizioni da luogo a quelli che sono definiti come *vincoli di assegnamento*. Tali vincoli possono essere utilizzati per creare un *ordinamento* tra oggetti. Si supponga che, all'interno di un problema, si debba decidere con quale ordine effettuare n lavori $I = \{1, 2, \ldots, n\}$. In tal caso, i vincoli di assegnamento tra I ed I impongono un ordinamento dei lavori: se $x_{ij} = 1$ indica che il lavoro i viene effettuato come j-esimo; in questo modo, una soluzione ammissibile per i vincoli di assegnamento assegna ad ogni lavoro i una posizione all'interno dell'ordinamento.

Spesso capita invece che la relazione non sia simmetrica, ad esempio perché i due insiemi di elementi non sono della stessa cardinalità e quindi avere più elementi che condividono la stessa scelta è inevita-

bile. Si parla in questo caso di *vincoli di semiassegnamento*, che sono presenti in molti modelli diversi. Quello che segue in particolare combina vincoli di semiassegnamento con relazioni logiche come quelle viste in precedenza.

Esempio 1.11. Assegnamento di frequenze

Nei primi, ruggenti anni della telefonia cellulare il gestore italiano UJN, sempre un po' in ritardo rispetto al più blasonato "incumbent", doveva ottimizzare la sua infrastruttura di trasmissione per provare a strappare (e trattenere) nuovi clienti. UJN disponeva di un insieme $S = \{ 1 , 2 , \dots , n \}$ di stazioni radio base (antenne) in grado di coprire tutta la città. Per poter attivare la rete, però, in quei tempi antichi prima dell'avvento della modulazione OFDM doveva assegnare a ciascuna antenna una frequenza di trasmissione in modo tale che le antenne adiacenti—che servono celle sovrapposte—non facessero eccessiva interferenza. Era disponibile un insieme $F = \{ f_1 , \dots , f_m \}$, di frequenze, in cui ogni f_i era un valore numerico (espresso in Mhz). Inoltre, per ogni coppia (non orientata) $\{ i , j \}$ di antenne era possibile stimare se la distanza era tale da fare eccessiva interferenza se si fosse assegnata la stessa frequenza a i ed a j. Ciò può essere rappresentato attraverso un grafo (non orientato) $G = (S , E)$ i cui sono le antenne ed esiste il lato $\{ i , j \} \in E$ se e solo se le due antenne i e j sono adiacenti. Dato che acquisire il diritto di uso di ogni frequenza, per l'intera città, aveva un (sostanziale) costo, UJN doveva determinare un'assegnamento di frequenze alle antenne che non producesse interferenza utilizzando il minimo numero di frequenze, considerando che a due vertici non adiacenti nel grafo può essere assegnata la stessa frequenza. In altre parole, si vuole "colorare" il grafo G con i "colori" f_i in modo tale che i vertici adiacenti abbiano colori diversi e che il numero di colori utilizzati sia minimo.

Matematicamente, una soluzione ammissibile corrisponde ad una partizione dell'insieme S in m sottoinsiemi $S(1)$, $S(2)$, $\dots$, $S(m)$, che possono anche essere vuoti, in cui il generico sottoinsieme $S(f)$ rappresenta le antenne a cui è assegnatala frequenza $f \in F$, con la condizione che siano tutte abbastanza lontane tra di loro da non fare interferenza tra di loro. In altri termini, per ciascun'antenna $i \in S$ dobbiamo selezionare una ed una sola frequenza f_j, $j \in F$, senza però il vincolo che la stessa frequenza non sia assegnata a più antenne diverse (anzi, questo è desiderabile nel problema). Per modellare come *PLI* il problema possiamo quindi introdurre nm variabili binarie x_{if} per $i \in S$ ed $f \in F$, intendendo che $x_{if} = 1$ se la frequenza f viene assegnata all'antenna i. La variabile x_{if} rappresenta l'appartenenza o meno di i a $S(f)$.

Siccome ad ogni antenna deve essere assegnata una ed una sola frequenza, dobbiamo imporre i vincoli di semiassegnamento

$$\sum_{f \in F} x_{if} = 1 \qquad i \in S \ . \tag{1.16}$$

Dobbiamo inoltre garantire che l'assegnamento di frequenze non causi interferenza, ossia che a vertici adiacenti siano assegnate frequenze diverse. Questo può essere fatto attraverso i vincoli logici

$$x_{if} + x_{jf} \leq 1 \qquad \{ i , j \} \in E \quad , \quad f \in F \ . \tag{1.17}$$

La correttezza di tali vincoli è immediata, ma può essere facilmente verificata considerando la condizione logica "$\neg x_{if} \vee \neg x_{jf}$" (almeno una delle due condizioni tra x_{if} e x_{jf} deve essere falsa), che è una clausola, ed utilizzando la formula (1.15) che fornisce immediatamente $-x_{if} - x_{jf} \geq 1 - 2$. Per formulare la funzione obiettivo si deve esprimere il numero di frequenze utilizzate; in questo caso è conveniente introdurre, per ogni frequenza $f \in F$, un'ulteriore variabile logica $y_f \in \{ 0 , 1 \}$ che assume il valore 1 se essa viene utilizzata e 0 altrimenti. In tal modo la funzione obiettivo, da minimizzare, è

$$\sum_{f \in F} y_f \quad .$$

Si deve quindi esprimere il legame tra le frequenze utilizzate (cioè le variabili y_f) e le antenne assegnate ad esse (cioè le variabili x_{if}); in altri termini si deve imporre che se alla frequenza f è stata assegnata almeno un'antenna allora "deve" essere $y_f = 1$. Ciò è esprimibile nei due modi diversi

$$y_f \geq x_{if} \qquad i \in S \quad , \quad f \in F \tag{1.18}$$

$$n y_f \geq \sum_{i \in F} x_{if} \qquad f \in F \tag{1.19}$$

Il primo insieme di mn vincoli (1.18) forza $y_f = 1$ se almeno una delle variabili x_{if} ha valore 1; tale insieme di vincoli è corrispondente alle implicazioni logiche "$x_{if} \Longrightarrow y_f$", ossia "se all'antenna i è assegnata la frequenza f, allora questa è utilizzata" (cf. 1.2.3). Il secondo insieme (1.19), di soli m vincoli, mette in relazione y_f con il numero $\sum_{i \in S} x_{if}$ di frequenze assegnate ad f. Se esso è positivo (ma sicuramente non superiore ad n), allora il vincolo impone $y_f = 1$. Si noti quindi come le variabili booleane abbiano una "doppia natura", logica e numerica.

Complessivamente, una possibile formulazione *PLI* del problema (GC, da *Graph Coloring*) è quindi

$$\text{(GC)} \qquad \begin{array}{ll} \min & \sum_{f\in F} y_f \\ & \sum_{f\in F} x_{if} = 1 \quad i \in S \\ & x_{if} + x_{jf} \le 1 \quad (i\,,\,j) \in E \quad , \quad f \in F \\ & y_f \ge x_{if} \qquad i \in S \quad , \quad f \in F \\ & x_{if} \in \{\,0\,,\,1\,\} \quad i \in S \quad , \quad f \in F \\ & y_f \in \{\,0\,,\,1\,\} \quad f \in F \end{array}$$

Si noti che se alla frequenza f non sono stati assegnati antenne, y_f può assumere entrambi i valori 0 e 1; siccome si minimizza la somma delle y_f, e nessun vincolo impone che y_f sia 1, all'ottimo tale variabile assumerà il corretto valore 0. Questo vale anche per il modello che usa i vincoli (1.19). In effetti è facile verificare che nel modello come è scritto il vincolo di integralità $y_f \in \{\,0\,,\,1\,\}$ non è necessario purché sia presente quello sulle x_{if}: infatti, se queste ultime sono intere allora per qualsiasi frequenza f utilizzata si avrà sicuramente $y_f \ge x_{if} = 1$ per almeno un $i \in S$, e pertanto il valore ottimo di y_f sarà necessariamente 1 anche in assenza dei vincoli di integralità sulle y_f. È peró facile verificare che lo stesso risultato non vale qualora si utilizzi l'altra variante del modello.

Esercizio 1.7. Dimostrare che la rimozione dei vincoli $y_f \in \{\,0\,,\,1\,\}$, $f \in F$ non può essere applicata se si usano i vincoli (1.19) al posto dei vincoli (1.18).

Invece, in nessun caso è possibile eliminare il vincolo di integralità sulle x_f senza perdere una parte sostanziale della struttura del problema originario. Si consideri infatti il *rilassamento continuo* di (GC) in cui sono stati rimossi tutti i vincoli di integralità, e quindi tutte le variabili possono assumere qualsiasi valore nell'intervallo $[\,0\,,\,1\,]$. È immediato verificare che la soluzione che assegna valore $1\,/\,m$ a tutte le variabili, sia x_{if} che y_f, è ammissibile per il problema. Infatti sono ovviamente rispettati sia i vincoli (1.16) che quelli (1.17) (purché $m \ge 2$) che infine quelli (1.18); ovviamente, tale soluzione di valore ottimo 1 non ha assolutamente alcun significato utile in termini della soluzione del problema originario.

Esercizio 1.8. Si formuli la generalizzazione del problema (GC) in cui ad ogni antenna i devono essere assegnate esattamente n_i frequenze diverse, e anche "frequenze vicine" fanno interferenza, nel senso che se a due nodi adiacenti i e j vengono assegnate due frequenze h e k, allora occorre che h e k siano "distanti", ossia che $|\,f_h - f_k\,| > \delta$ per una soglia δ fissata.

Presentiamo adesso il modello di una situazione che parrebbe non avere alcuna relazione com il problema precedentemente descritto, ma che si presta invece da una descrizione come *PLI* del tutto analoga.

Esempio 1.12. Ordinamento di lavori su macchine: minimizzazione del numero delle macchine

Nel malfamato porto spaziale di Mos Eisley, sul pianeta Tatooine dell'Orlo Esterno della Galassia, la Han Solo Corporation (HSCo) ha, con la connivenza della mafia degli Hutt, ottenuto il totale monopolio del servizio ai trasporti Imperiali. La HSCo dispone di un insieme $M = \{\,1\,,\,2\,,\,\ldots\,,\,p\,\}$ di moli spaziali che può usare per le operazioni di carico e scarico dei trasporti. Per ogni giorno è noto l'insieme $T = \{\,1\,,\,2\,,\,\ldots\,,\,n\,\}$ dei trasporti in arrivo, ciascuno dei quali può essere fatto atterrare in uno qualsiasi dei moli. Il trasporto i-esimo arriva all'ora t_i, e le sue operazioni di carico e/o scarico richiedono tempo d_i, entrambi noti. Una volta atterrato le operazioni iniziano immediatamente ed il trasporto non può essere spostato ad un altro molo finché non siano terminate, al che il trasporto decolla immediatamente; i tempi t_i e d_i sono già "generosamente dimensionati" per tenere conto della ben nota inaffidabilità dei piloti imperiali e dei droidi di servizio ai moli, in modo da essere sicuri che il trasporto non arrivi prima di t_i e sia già partito a $t_i + d_i$. La vuole utilizzare il minor numero possibile di moli diversi, in quanto in ciascuno di essi deve essere presente un funzionario "amico" della Dogana Imperiale che dovrà poi essere adeguatamente ricompensato per chiudere un occhio (o anche tutti e due) sulle numerose attività illegali che vengono compiute durante le operazioni di carico e scarico. Si aiuti quindi la HSCo a continuare a fare una montagna di soldi dalle sue attività illegali (buona parte dei quali vengono poi segretamente utilizzati per finanziare la Resistenza) modellando come *PLI* il problema di assegnare ciascun trasporto ad un molo in modo ammissibile (i periodi in cui trasporti diversi usano lo stesso molo devono essere disgiunti) minimizzando il numero di moli diversi utilizzati.

Analogamente al caso precedente, per modellare il problema come *PLI* introduciamo pn variabili binarie x_{im} per $i \in T$ ed $m \in M$, intendendo che $x_{im} = 1$ se il trasporto i viene fatto atterrare al molo m, e $x_{im} = 0$ altrimenti. Per rappresentare mediante le variabili x_{im} l'insieme delle soluzioni ammissibili, dobbiamo imporre che ogni trasporto sia fatto atterrare in uno ed un solo molo, il che può essere espresso mediante i vincoli di semiassegnamento

$$\sum_{m\in M} x_{im} = 1 \quad , \quad i \in T\,.$$

Dobbiamo inoltre garantire che se due trasporti i e j sono assegnati allo stesso molo m allora i loro periodi di stazionamento, cioè gli intervalli $[\,t_i\,,\,t_i + d_i\,]$ e $[\,t_j\,,\,t_j + d_j\,]$, siano *disgiunti*; in altri termini si deve avere che o

$t_i + d_i < t_j$ (il trasporto i parte prima dell'arrivo del trasporto j), oppure il viceversa, $t_j + d_j < t_i$. Per ogni trasporto $i \in T$ definiamo quindi l'insieme $T(i) \subseteq T \setminus \{ i \}$ dei trasporti j che sono *incompatibili* con esso:

$$T(i) = \{ j \in T \setminus \{ i \} \ : \ [t_i , t_i + d_i] \cap [t_j , t_j + d_j] \neq \emptyset \} \quad , \quad i \in T \ .$$

Si noti che gli *insiemi di incompatibilità* $T(i)$, $i \in T$, sono facilmente costruibili a partire dai dati di "input" del problema. Alternativamente possiamo considerare il grafo non orientato $G = (T, E)$ i cui vertici sono i trasporti ed i cui lati $\{ i , j \}$ rappresentano la relazione di incompatibilità; tale grafo è derivabile in tempo polinomiale dagli input t_i e d_i dell'insieme, e pertanto potrebbe anche essere considerato come parte dell'input stesso (anche eliminando dall'input t_i e d_i, che non sarebbero a quel punto più necessari). Mediante tali insiemi possiamo scrivere i seguenti *vincoli di compatibilità*:

$$x_{im} + x_{jm} \leq 1 \qquad \{ i , j \} \in E \quad , \quad m \in M \ .$$

Per formulare la funzione obiettivo si deve esprimere il numero di moli utilizzati; come nel caso precedente introduciamo per questo le variabili logiche $y_m \in \{ 0 , 1 \}$ per $m \in M$, che usiamo per definire la funzione obiettivo $\min \sum_{m \in M} y_m$, ed (ad esempio) i vincoli "logici" $y_m \geq x_{im}$. La corrispondente formulazione *PLI* del problema è

$$\text{(MCMS)} \qquad \begin{array}{lll} \min & \sum_{m \in M} y_m & \\ & \sum_{m \in M} x_{im} = 1 & i \in T \\ & x_{im} + x_{jm} \leq 1 & \{ i , j \} \in E \quad , \quad m \in M \\ & y_m \geq x_{im} & i \in T \quad , \quad m \in M \\ & x_{im} \in \{ 0 , 1 \} & i \in T \quad , \quad m \in M \\ & y_m \in \{ 0 , 1 \} & m \in M \end{array}$$

Tale problema è noto come (MCMS), da *Minimal Cardinality Machine Scheduling*, in quanto si ritrova in molti ambiti che richiedono genericamente di assegnare (molti) "lavori" i a (poche) "macchine" j in modo tale che ciascuna macchina sia in grado di eseguire i lavori ad essa assegnati minimizzando il numero delle macchine utilizzate. Ciò ha applicazioni nella logistica produttiva, nei trasporti, nell'informatica (assegnazione di task non interrompibili a CPU) e molti altri. Ma dal punto di vista matematico, (MCMS) è fondamentalmente identico a (GC) quando utilizziamo i dati di input sui lavori per costruire il grafo G di (in)compatibilità tra di essi come precedentemente fatto per le stazioni radio base. Ciò mostra come una singola struttura, o anche un intero modello, possano essere utilizzati in molteplici ambiti applicativi diversi.

Esercizio 1.9. Costruire un'istanza del problema (MCMS) con 7 lavori (trasporti), definendo le durate e i tempi di inizio di essi; formulare quindi l'istanza, fornire due soluzioni ammissibili e valutare il numero di macchine (moli) occorrenti.

Esercizio 1.10. Costruire un modello della variante del problema in cui i moli sono di k tipi diversi, $K = \{ 1 , 2 , \ldots , k \}$, per ciascun molo $m \in M$ è noto il suo tipo $k(m) \in K$, e ciascun trasporto $i \in T$ può atterrare solamente in moli di un certo sottoinsieme di tipi $k(i) \subseteq K$.

1.2.5 Selezione di sottoinsiemi

Negli esempi precedenti abbiamo visto come la selezione di opportuni sottoinsiemi possa essere uno degli *output* fondamentali di un modello di *PLI*, e come particolari sottoinsiemi di coppie possano essere una parte fondamentale dell'*input* del modello in quanto rappresentano relazioni cruciali ai fini di determinare le condizioni logiche di ammissibilità di una soluzione. Vediamo adesso una particolare tecnica di modellazione che generalizza quest'ultimo concetto costruendo formulazioni in cui ci sono variabili che corrispondono a particolari famiglie di sottoinsiemi di uno o più insiemi dati. Sia E un insieme finito di elementi (ad esempio $E = \{ 1 , 2 , \ldots , n \}$), e

$$F = \{ F_1 , F_2 , \ldots , F_m \}$$

una famiglia di m suoi sottoinsiemi, ovvero $F_j \subseteq E$ per $j = 1, \ldots, m$. Con un piccolo abuso di notazione, frequentemente utilizzato, possiamo alternativamente considerare $F = \{ 1 , 2 , \ldots , m \}$ come l'insieme degli *indici* dei sottoinsiemi disponibili. Ad ogni sottoinsieme F_j, $j \in F$, è associato un costo c_j (fornito in input insieme alla descrizione di F); il problema che vogliamo considerare è quello di determinare una sottofamiglia $S \subseteq F$ di costo minimo che soddisfi particolari vincoli.

Introduciamo per questo m variabili binarie x_1, x_2, $\ldots$, x_m, col significato che $x_j = 1$ se $F_j \in S$, e $x_j = 0$ altrimenti, per $j = 1, \ldots, m$. Il costo di S è allora definito da:

$$\sum_{j \in F} c_j x_j \ .$$

Considereremo tre tipi di vincoli, detti rispettivamente di copertura, di partizione e di riempimento, che danno origine a tre diversi problemi di selezione di sottoinsiemi.

- *Problema di copertura*: il caso in cui si voglia che ogni elemento di E sia selezionato *almeno* una volta, cioè che appaia in almeno uno degli insiemi di S, può essere espresso attraverso il *vincolo di copertura*

$$\sum_{j\,:\,i\in F_j} x_j \geq 1 \qquad i \in E \ , \tag{1.20}$$

che garantisce che per ogni $i \in E$ esista almeno un indice j per cui $F_j \in S$ e $i \in F_j$. Il problema di copertura (SC, da *Set Covering*) è allora definito da

$$\text{(SC)} \qquad \min \left\{ \sum_{j\in F} c_j x_j \ : \sum_{j\,:\,i\in F_j} x_j \geq 1 \quad i \in E \quad , \quad x \in \{\,0\,,\,1\,\}^m \right\}$$

Si noti che è possibile rappresentare la famiglia F mediante una matrice $A \in \mathbb{R}^{n\times m}$ il cui generico elemento a_{ij} vale 1 se $i \in F_j$ e 0 altrimenti; così facendo, i vincoli di copertura divengono

$$\sum_{j\in F} a_{ij}\, x_j \geq 1 \qquad i \in E \ .$$

- *Problema di partizione*: in questo caso si richiede invece che ogni elemento di E sia selezionato *esattamente* una volta, cioè che appaia in uno ed uno solo degli insiemi di S. Formalizziamo questa richiesta attraverso i *vincoli di partizione*

$$\sum_{j\,:\,i\in F_j} x_j = 1 \qquad i \in E \ , \tag{1.21}$$

che garantiscono, che per ogni $i \in E$, esista un solo indice j per cui $i \in F_j$ e $F_j \in S$. Il *problema di partizione* (SPP, da *Set Partitioning*) si ottiene facilmente sostituendo i vincoli (1.21) a quelli di copertura (1.20):

$$\text{(SPP)} \qquad \min \left\{ \sum_{j\in F} c_j x_j \ : \sum_{j\,:\,i\in F_j} x_j = 1 \quad i \in E \quad , \quad x \in \{\,0\,,\,1\,\}^m \right\}$$

- *Problema di riempimento*: il caso rimanente è quello in cui si voglia che ogni elemento di E sia selezionato non più di una volta, cioè che appaia in al più uno degli insiemi di S. Formalizziamo questa richiesta attraverso i *vincoli di riempimento*

$$\sum_{j\,:\,i\in F_j} x_j \leq 1 \qquad i \in E \ ,$$

che garantiscono che per ogni $i \in E$ non possa esistere più di un indice j per cui $i \in F_j$ e $F_j \in S$. Il *problema di riempimento* ((SPA), da *Set Packing*) è allora definito come

$$\text{(SPA)} \qquad \min \left\{ \sum_{j\in F} c_j x_j \ : \sum_{j\,:\,i\in F_j} x_j \leq 1 \quad i \in E \quad , \quad x \in \{0,1\}^m \right\}$$

Si noti che, se tutti i costi sono non negativi, questo problema ha banalmente soluzione ottima nulla; per questo, lo si trova più spesso formulato come problema di massimizzazione.

Esercizio 1.11. Sia data la famiglia $F = \{\{\,1\,,\,3\,,\,5\,\},\{\,2\,,\,3\,\},\{\,1\,,\,4\,\},\{\,3\,,\,4\,,\,5\,\},\{\,2\,\},\{\,5\,\},\{\,1\,,\,5\,\}\}$ formata da 7 sottoinsiemi di $E = \{\,1\,,\,2\,,\,3\,,\,4\,,\,5\,\}$. Formulare i problemi di copertura, partizione e riempimento sapendo che il vettore dei costi è $c = [\,3\,,\,5\,,\,1\,,\,9\,,\,2\,,\,4\,,\,1\,]$. Determinare una soluzione ammissibile, se esiste, per ciascun problema.

Le tre condizioni (copertura, partizione, riempimento) sono solo tre possibili esempi del tipo di formulazioni che possono essere costruite con questa tecnica. Gli esercizi seguenti forniscono alcuni spunti per possibili generalizzazioni.

Esercizio 1.12. Si formulino analiticamente i problemi di copertura, partizione e riempimento generalizzati in cui ogni oggetto i deve far parte di, rispettivamente, almeno, esattamente ed al più b_i sottoinsiemi.

Esercizio 1.13. Si formulino analiticamente le ulteriori generalizzazioni dei problemi di copertura, partizione e riempimento, generalizzate nell'esercizio precedente, in cui di ogni sottoinsieme F_j si possano prendere più copie, eventualmente limitate ad un numero massimo di u_j.

Un punto critico dei problemi così definiti è che l'insieme di tutti i possibili sottoinsiemi di E ha cardinalità esponenziale 2^n; pertanto, queste formulazioni sono "banalmente" utilizzabili solamente

quando la particolare famiglia di sottoinsiemi F di interesse sia "abbastanza piccola". Questo è in qualche modo necessariamente garantito nel caso in cui essa sia esplicitamente fornita come input del problema, come nel seguente esempio.

Esempio 1.13. Un problema di selezione di configurazioni

Anno 1980 dell'Era Normale (AD). Tommaso, fan sfegatato della serie TV "Gundam", sta pianificando il suo investimento nei *Gunpla*, i dettagliatissimi (e costosissimi) modellini in plastica da montare che raffigurano i "mecha" della serie e che faranno la fortuna della Bandai/Sunrise negli anni a venire. Tommaso può comprare tre confezioni: il Mobil Suit "Gundam" (G), formato dalla parte superiore (G^u), quella inferiore (G^d) ed un caccia "Core Fighter" (C), l'aereo di supporto "G-Fighter" (F), formato dalla parte superiore (F^u) e quella inferiore (F^d), ed il C da solo, al costo di 65000 Lire, 40000 Lire e 15000 Lire rispettivamente. Questi possono essere montati in 8 diverse combinazioni, mostrate nella seguente tabella. Tommaso assegna a ciascuna di esse un valore "V", dipendente da quanto siano memorabili le scene in cui appaiono nella serie e quanto sembrino "ganze" quando montate. Inoltre, Tommaso valuta che la sua collezione guadagnerebbe altri 10 punti agli occhi dei suoi amici se fosse composta da almeno metà delle combinazioni possibili. Aiutate Tommaso ad allocare in modo ottimale il budget di 165000 Lire faticosamente raggranellato con le paghette scrivendo come *PLI* il problema di determinare quante confezioni di ciascun tipo comprare e quali combinazioni montare in modo da massimizzare il valore totale della collezione.

combinazione		G^u	G^d	CF	F^u	F^d	V
G-Fighter	(1)				1	1	6
G-Bull	(2)	1		1	1		3
G-Bull Easy	(3)				1		2
G-Sky	(4)		1	1		1	5
Gundam Sky	(5)	1	1	1		1	9
Gundam	(6)	1	1	1			10
Core Fighter	(7)			1			7
G-Armor	(8)	1	1	1	1	1	8

Definiamo le variabili binarie x_c per $c \in C = \{1, \ldots, 8\}$ che indicano se vengono montate le corrispondenti combinazioni, le variabili y_a (intere e non negative) per $a \in A = \{G, F, C\}$ che indicano quante confezioni di ogni tipo vengano comprate, e la singola variabile binaria z che indica se la collezione comprende almeno 4 combinazioni. Una formulazione del problema di Tommaso è:

$$
\begin{array}{lrrrrrrrrrrrrrrrrrrl}
\max & & & & 10z & +6x_1 & +3x_2 & +2x_3 & +5x_4 & +9x_5 & +10x_6 & +7x_7 & +8x_8 & & \\
& y_G & & & & & -x_2 & & & -x_5 & -x_6 & & -x_8 & \geq & 0 \\
& y_G & & & & & & & -x_4 & -x_5 & -x_6 & & -x_8 & \geq & 0 \\
& y_G & & +y_C & & & -x_2 & & -x_4 & -x_5 & -x_6 & -x_7 & -x_8 & \geq & 0 \\
& & y_F & & & -x_1 & -x_2 & -x_3 & & & & & -x_8 & \geq & 0 \\
& & y_F & & & -x_1 & & & -x_4 & -x_5 & & & -x_8 & \geq & 0 \\
& 65y_G & +40y_F & +15y_C & & & & & & & & & & \leq & 165 \\
& & & & 4z & -x_1 & -x_2 & -x_3 & -x_4 & -x_5 & -x_6 & -x_7 & -x_8 & \leq & 0 \\
& y_G, & y_F, & y_C & & & & & & & & & & \in & \mathbb{N} \\
& & & & z, & x_1, & x_2, & x_3, & x_4, & x_5, & x_6, & x_7, & x_8 & \in & \{0, 1\}
\end{array}
$$

I primi 5 vincoli riguardano la disponibilità dei componenti di base: nell'ordine i G^u, G^d, C, F^u ed F^d utilizzati per le combinazioni effettivamente montate devono essere minori od uguali di quelli disponibili nel complesso delle confezioni comperate. Il sesto vincolo riguarda il budget finanziario. Il settimo impone che z possa valere 1 solo se almeno quattro delle x_c valgono 1; se ciò accade, la funzione obiettivo è tale per cui $z = 1$ è sicuramente la soluzione ottima. I successivi sono i vincoli di integralità ed i bounds sulle variabili.

In molti casi, però, la famiglia F di interesse non è direttamente fornita in input, ma deve essere calcolata a partire da condizioni logiche sui sottoinsiemi ammissibili, come nell'Esempio seguente.

Esempio 1.14. Un problema di instradamento

La Trans-Port (TP), azienda logistica che lavora all'Interporto di Guasticce, deve pianificare i viaggi dei suoi camion. Ciascun camion parte dal deposito (d) e deve consegnare containers pieni provenienti dal porto agli importatori (I) e/o ritirare containers vuoti dagli esportatori (E) per riportarli a d, da dove vengono poi trasportati al porto per essere riutilizzati; le tratte tra il porto e d sono ovviamente fisse e comunque organizzate indipendentemente, quindi non devono essere considerate. Ciascun camion porta fino a due containers, e può essere usato sia per la consegna che per il ritiro; per motivazioni organizzative, però, deve prima consegnare (a meno che non parta vuoto) e poi ritirare (a meno che non rientri vuoto). È nota la distanza f_{hk} tra tutte le coppie $\{h, k\}$ di punti rilevanti (d,

tutti gli $i \in I$ e tutti gli $e \in E$), che può essere assunta simmetrica. È anche nota la domanda c_i, $i \in I$ e c_e, $e \in E$ di containers da consegnare/ritirare per ogni esportatore/importatore; si noti che un certo esportatore i potrebbe anche essere un importatore e, ma questo si modella semplicemente considerandoli diversi e ponendo $f_{ie} = 0$. Si può assumere che la TP disponga di un numero illimitato di camion (una volta rientrato al deposito un camion può iniziare un altro viaggio, e la durata dei viaggi è tale per cui i camion effettivamente disponibili sicuramente riusciranno a fare tutti i viaggi necessari entro la giornata lavorativa). Occorre formulare come *PLI* il problema che la TP deve risolvere per soddisfare le domande di tutti gli esportatori/importatori minimizzando la distanza totale percorsa dai suoi camion, e quindi il costo di trasporto.

Per modellare il problema definiamo, a partire dai dati in input, quattro insiemi di cammini:

- P_1 tutti i cammini che toccano un solo utente, ossia della forma $d \to i \to d$ per $i \in I$ oppure $d \to e \to d$ per $e \in E$;
- P_2 tutti i cammini che toccano esattamente due utenti, ossia della forma $d \to i_1 \to i_2 \to d$ per $i_1, i_2 \in I$ oppure $d \to i \to e \to d$ per $i \in I$, $e \in E$ oppure $d \to e_1 \to e_2 \to d$ per $e_1, e_2 \in E$ (si noti che per la "policy" aziendale non è possibile visitare prima un esportatore e poi un esportatore, quindi i cammini corrispondenti sono esclusi);
- P_3 tutti i cammini che toccano esattamente tre utenti, ossia della forma $d \to i_1 \to i_2 \to e \to d$ per $i_1, i_2 \in I$ ed $e \in E$, oppure $d \to i \to e_1 \to e_2 \to d$ per $i \in I$ ed $e_1, e_2 \in E$ (si noti che non è fisicamente possibile visitare tre importatori o esportatori per via della capacità del camion, e che per la "policy" aziendale non è possibile visitare prima un esportatore e poi un esportatore, quindi i cammini corrispondenti sono esclusi);
- P_4 tutti i cammini che toccano esattamente quattro utenti, ossia della forma $d \to i_1 \to i_2 \to e_1 \to e_2 \to d$ per $i_1, i_2 \in I$ ed $e_1, e_2 \in E$ (ancora, è necessario visitare prima i due importatori per caricare i container, e solo dopo i due esportatori per consegnarli).

Si noti che nella definizione dei cammini niente impone che $i_1 \neq i_2$ e/o $e_1 \neq e_2$: lo stesso camion può consegnare/ritirare due containers allo/dallo stesso importatore/esportatore. Per tener conto di questo fatto, per ciascun cammino p che contiene un importatore i si definisce a_p^i come il numero di volte che il cammino visita l'importatore (1 o 2), ed analogamente per gli esportatori. Ovviamente, se $i_1 = i_2$ si avrà $f_{i_1,i_2} = 0$, e similmente per gli esportatori. Si definisce $P = P_1 \cup P_2 \cup P_3 \cup P_4$, e per ogni $p \in P$ il costo f_p come la somma delle distanze dei vari tratti del cammino. Per il vincolo organizzativo, qualsiasi camion deve percorrere uno dei cammini in P. Introducendo una variabile intera c_p per $p \in P$ che indica il numero di camion che effettuano il cammino p, il problema può essere formulato semplicemente come:

$$
\begin{aligned}
\min\ & \textstyle\sum_{p \in P} f_p c_p & & \\
& \textstyle\sum_{p \in P\,:\, i \in p} a_p^i c_p = c_i & & i \in I \\
& \textstyle\sum_{p \in P\,:\, e \in p} a_p^e c_p = c_e & & e \in E \\
& c_p \in \mathbb{N} & & p \in P
\end{aligned}
$$

La funzione obiettivo rappresenta la distanza totale percorsa dai camion. I primi due blocchi di vincoli assicurano la soddisfazione delle domande rispettivamente per gli importatori e gli esportatori. Infatti, per ogni importatore si sommano tutte le variabili che corrispondono ai cammini che lo comprendono (come tale), ciascuna moltiplicata per il fattore 1 o 2 a seconda che il cammino lo visiti una o due volte e che di conseguenza gli porti uno o due container. Di conseguenza, $a_p^i c_p$ è il numero totale di container (pieni) portati all'importatore i da camion che fanno esattamente il cammino p, e sommando su tutti i p che contengono i si ottiene il numero totale di container portati ed i, che deve essere pari alla sua domanda. Argomentazioni completamente simmetriche valgono per gli esportatori. Infine sono presenti, come necessario, i normali vincoli (di segno e) di integralità delle variabili.

La formulazione ha $|I| + |E|$ vincoli, quindi un numero che aumenta linearmente col numero di utenti che in pratica non può essere troppo grande. Per contro ha una variabile per ciascun elemento dell'insieme P. La cardinalità di tale insieme cresce molto più rapidamente col numero di utenti, ma pur sempre in modo polinomiale: $|P| \in O(|I|^2 |E|^2)$, e tutti i suoi elementi possono essere enumerati (compreso il calcolo del loro costo) con tale complessità. Pertanto, a meno che il numero di utenti non sia molto grande le dimensioni del modello rimangono ragionevolmente contenute ed è possibile pensare di utilizzarlo in pratica. In effetti si trova che il modello così costruito risulta preferibile a modelli di dimensioni inferiori quando risolto con le tecniche standard, per motivi che non possono ancora essere discussi.

La definizione di F ottenuta imponendo condizioni logiche sui suoi elementi espone facilmente al rischio che la sua cardinalità divenga molto grande: si pensi al caso in cui, nell'esempio precedente, i camion possano fare cammini con un numero in principio arbitrario di nodi, o comunque con un limite k molto grande. In realtà utilizzare formulazioni di questo tipo può essere, sotto alcune condizioni, possibile anche in questo caso; anzi, tali formulazioni sono in effetti tra le migliori per alcune specifiche classi di problemi rilevanti per le applicazioni, tra cui quelli di instradamento (routing). Il loro uso richiede

però l'utilizzo di metodologie di ottimizzazione sofisticate (generazione di colonne) che non possono essere descritte in queste dispense.

1.2.6 Problemi di flusso e di cammino

Nei paragrafi precedenti abbiamo tipicamente introdotto o variabili quantitative, o variabili logiche. Nei fatti però anche le variabili quantitative possono essere considerate come "logiche" nel momento in cui indicano non solo se una scelta viene compiuta oppure no, ma anche "quante volte viene compiuta", come visto negli Esempi 1.13 e 1.14. In quei casi la variabile era a valori interi (ma non binaria), e quindi indicava una molteplicità di scelte. Introduciamo adesso una famiglia di problemi, noti come *problemi di flusso su grafo*, che mostrano (tra le altre cose) come questo possa accadere anche con variabili quantitative non vincolate ad assumere valori interi.

Esempio 1.15. Il problema delle Tartaruge Ninja

Le Tartarughe Ninja Mutanti Teenager (TMNT) vorrebbero, per combattere più efficacemente il male, potersi muovere più facilmente di nascosto tra i vari quartieri di New York utilizzando le fogne nelle quali vivono, cosa resa difficile dal fatto che i sistemi fognari di diversi quartieri non sono collegati tra di loro. Per questo, attraverso i loro contatti da supereroi sono riusciti a convincere il Consiglio Comunale a varare un piano di espansione e connessione dei sistemi fognari attuali, con la giustificazione tecnica di connetterli tutti al nuovo e più efficiente depuratore. Per analizzare la situazione hanno sviluppato il grafo $G = (N, A)$ di Figura 1.3(a), dove i nodi da 1 a 4 rappresentano i quartieri ed il nodo 5 rappresenta il depuratore, mentre gli archi rappresentano i nuovi tratti di fognatura che sarebbe possibile scavare per collegarli. Poiché i lavori di scavo sono molto costosi, ed il budget del Comune di New York è sempre sotto pressione per le mille altre necessità, è fondamentale per il successo dell'operazione minimizzare il costo di realizzazione pur garantendo che il nuovo impianto sarà in grado di funzionare senza problemi. Per questo si conosce la quantità di rifiuti (liquidi) che ciascun quartiere produce per unità di tempo (il numero vicino al nodo nella Figura), ed è necessario dimensionare opportunamente l'impianto di collegamento al depuratore in modo da consentirne il trasporto. Si possono scegliere condotte di diversa portata e il costo di messa in opera di una condotta sarà una funzione della sua portata, oltreché naturalmente della sua lunghezza: accanto ad ogni arco in Figura è mostrato il costo necessario per realizzare il corrispondente collegamento di capacità unitaria, e si suppone che il costo scali linearmente con la capacità (una conduttura di capacità doppia costerà il doppio). Occorre selezionare le condotte (archi) da costruire e per ciascuna di essa stabilire la capacità in modo che sia in grado di far fluire tutto il liquame necessario, a costo minimo. Si noti che i collegamenti hanno una direzione prefissata di percorrenza, corrispondente al fatto che il liquame può fluire in una sola direzione per via della pendenza della condotta. Una soluzione ammissibile è data da un insieme di condotte che garantiscano il fluire dei liquami da tutti i quartieri fino al depuratore, ciascuna con la corrispondente portata. Si vuole formulare il problema come *PL*.

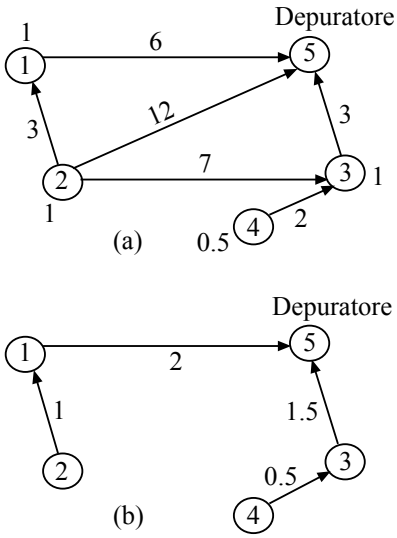


Figura 1.3: Progetto dell'espansione della rete fognaria di New York

Per costruire il modello introduciamo delle variabili quantitative x_{ij}, una per ogni arco (i, j) del grafo, che rappresentano la quantità di rifiuti liquidi inviati da i a j, nell'unità di tempo, lungo tale condotta. Esse sono chiaramente vincolate ad essere non negative; il valore $x_{ij} = 0$ indica che si è deciso di non utilizzare l'arco (i, j), ossia di costruire la corrispondente condotta, mentre un valore $x_{ij} > 0$ indica che la condotta dovrà essere costruita con tale capacità (in quanto deve avere capacità almeno sufficiente a far transitare i liquami previsti, ma per contro non deve avere capacià superiore a questa in quanto altrimenti la soluzione è chiaramente non ottima). Dobbiamo adesso introdurre dei vincoli che garantiscano che i liquami prodotti da ciascun quartiere giungano al depuratore. I liquami, per giungere al depuratore, dovranno passare per qualcuna delle condotte che vi arrivano direttamente; dovrà quindi essere

$$x_{15} + x_{25} + x_{35} = 3.5$$

poiché 3.5 è la quantità totale di rifiuti prodotta da tutti i quartieri, che quindi deve arrivare tutta al depuratore. Vincoli analoghi possono essere scritti per i nodi corrispondenti agli altri quartieri; in questo caso, però, è necessario tenere in conto anche del flusso in uscita dal nodo. In altri termini, per ogni quartiere occorre garantire che tutto il liquame che vi giunge, sia quello ivi prodotto che quello che lo attraversa proveniendo da altri quartieri, sia inviato, attraverso qualcuna delle condotte che escono dal nodo, o direttamente al depuratore o comunque ad un diverso quartiere, dal quale poi sarà ulteriormente instradato fino a raggiungere il depuratore. Per i nodi 3 ed 1 questo implica

$$x_{23} + x_{43} + 1 = x_{35} \quad , \quad x_{21} + 1 = x_{15} \; .$$

Per i nodi che hanno solamente archi uscenti, 2 e 4, i vincoli saranno

$$1 = x_{21} + x_{25} + x_{23} \quad , \quad 0.5 = x_{43} \ .$$

Il costo di posa in opera della condotta corrispondente all'arco (i, j) è quindi il prodotto della portata della condotta, x_{ij}, per il coefficiente c_{ij} che indica il costo della condotta unitaria sul tratto (il quale a sua volta avrà tipicamente una forma del tipo $c_{ij} = \delta l_{ij}$, l_{ij} è la lunghezza del tratto da i a j e δ è il costo per unità di lunghezza di una condotta di capacità unitaria). Una formulazione del problema della TMNT è quindi

$$\begin{array}{lrcrcrcrcrcrcr} \min & 6x_{15} & + & 3x_{21} & + & 7x_{23} & + & 12x_{25} & + & 3x_{35} & + & 2x_{43} & & \\ & -x_{15} & + & x_{21} & & & & & & & & & = & -1 \\ & & - & x_{21} & - & x_{23} & - & x_{25} & & & & & = & -1 \\ & & & & + & x_{23} & & & - & x_{35} & + & x_{43} & = & -1 \\ & & & & & & & & & & - & x_{43} & = & -0.5 \\ & x_{15} & & & & & + & x_{25} & + & x_{35} & & & = & 3.5 \\ & x_{15} & , & x_{21} & , & x_{23} & , & x_{25} & , & x_{35} & , & x_{43} & \geq & 0 \end{array} \ .$$

In Figura 1.3(b) è rappresentata una possibile soluzione, che come si può facilmente verificare rispetta tutti i vincoli. Il costo della soluzione è 20.5. La soluzione indica quindi non solo quali condotte costruire (informazione logica) ma anche, per ciascuna di quelle costruite, la sua postata (informazione quantitativa) utilizzando le stesse variabili di flusso x_{ij}.

Introduciamo adesso formalmente la famiglia dei problemi di flusso su grafo (MCF, da *Min-Cost Flow*) che generalizza l'esempio precedente. È dato un grafo *orientato* $G = (N, A)$, i cui nodi producono o consumano tutti uno stesso tipo di "flusso" (acqua, gas, elettricità, automobili, treni, pacchetti in una rete di comunicazione, ...) che transita dai nodi di produzione a quelli di consumo attraverso gli archi. I singoli elementi del flusso sono "fungibili", ossia interscambiabili tra loro: i nodi che consumano flusso sono indifferenti allo specifico nodo che lo ha prodotto. A ciascun $i \in N$ è associato un valore reale b_i, detto *deficit*: se è positivo indica la quantità di flusso che esce dalla rete al nodo i, che viene detto allora *destinazione*, mentre se è negativo indica la quantità di flusso che entra nella rete al nodo i, che viene detto allora *origine*. I nodi che non sono né origini né destinazioni ($b_i = 0$) sono detti *di trasferimento*. Si noti che nell'Esempio precedente c'erano origini multiple, una sola destinazione e nessun arco di trasferimento, ma in generale possono essercene molteplici di ciascun tipo, purché sia rispettata la condizione

$$\textstyle\sum_{i \in N} b_i = 0 \ ,$$

ossia "tutto ciò che è prodotto nelle origini viene consumato nelle destinazioni"; infatti, se la condizione non è valida il problema non può avere soluzione ammissibile (anche se ci sono modi per "aggirare" questa richiesta, si veda il §2.1.2). I principali vincoli del problema sono quelli di *conservazione del flusso*

$$\textstyle\sum_{(j,i) \in BS(i)} x_{ji} - \sum_{(i,j) \in FS(i)} x_{ij} = b_i \qquad i \in N \ ,$$

in cui $BS(i)$ e $FS(i)$ indicano rispettivamente la stella entrante e la stella uscente di $i \in N$ (si veda l'Appendice B), e garantiscono appunto che la differenza tra il flusso entrante in un nodo e quello uscente sia pari al deficit (quindi, in un nodo con deficit negativo il flusso uscente è superiore a quello entrante e viceversa). Si noti che nell'Esempio non era stato esplicitamente scritto il vincolo di conservazione del flusso relativo al deposito, cosa possibile in quanto è implicato dagli altri, come discusso nel Capitolo 2. A ciascun arco (i, j) è inoltre associata una capacità massima u_{ij}, che può essere infinita, ed un costo c_{ij} (che può essere nullo, o negativo) che viene pagato per ciascuna unità di flusso che transita lungo l'arco. La formulazione completa del problema è quindi

$$\text{(MCF)} \qquad \begin{array}{ll} \min & \textstyle\sum_{(i,j)) \in A} c_{ij} x_{ij} \\ & \textstyle\sum_{(j,i) \in BS(i)} x_{ji} - \sum_{(i,j) \in FS(i)} x_{ij} = b_i \quad i \in N \\ & 0 \leq x_{ij} \leq u_{ij} \qquad\qquad\qquad\qquad\quad (i, j) \in A \end{array}$$

Sono possibili varianti del problema, in cui ad esempio il flusso sugli archi non sia necessariamente non-negativo; si veda il §2.1.2.

Il problema (MCF) è estremamente rilevante in molte applicazioni, principalmente ma non esclusivamente nella logistica, nei trasporti e nelle telecomunicazioni, in quanto il movimento di entità lungo

una rete è una caratteristica di un grandissimo numero di sistemi tecnologici ed industriali (ma anche, ad esempio, biologici). Modelli di flusso appaiono quindi molto spesso come parti di problemi più complessi in cui occorre rappresentare, tra le altre cose, scelte quali la rotta che gli elementi del sistema devono fare per giungere dalla loro origine alla loro destinazione.

Esercizio 1.14. Si sviluppi una formulazione di *PLI* basata su variabili di flusso per il problema dell'Esempio 1.14.

In effetti è facile vedere che il problema (MCF) contiene come caso particolare quello che è considerato la "base" di tutti i problemi di instradamento: il problema di determinare un cammino di costo minimo tra due nodi dati $s \in N$ e $t \in N$, con $s \neq t$, in un grafo G. Tale problema si modella infatti scegliendo $b_s = -1$, $b_t = 1$, e $b_i = 0$ per ogni $i \in N \setminus \{ s , t \}$ e capacità u_{ij} infinite (o comunque ≥ 1). Nel grafo viene quindi prodotta un'unica unità di flusso, nel nodo (origine) s, che deve raggiungere il nodo (destinazione) t. Ogni volta che il flusso attraversa un arco viene pagato il suo costo c_{ij}, ed è quindi chiaro che la soluzione ottima consisterà nell'inviare tutto il flusso lungo il cammino di costo minimo. Questa idea permette di costruire molti modelli in cui occorra rappresentare il concetto di cammino ottimo, come illustrato dal seguente esempio.

Esempio 1.16. Un problema di cammino ottimo

Il Prof. Frignoni (PF) è stato inopinatamente (si vocifera di un errore, o di pressioni indebite) invitato al prestigioso workshop sui Metodi Unificati per il Controllo Combinatorio Algoritmico (MUCCA) di *Assois* (*Ass*), uno sperduto paesino delle Alpi francesi noto (al di fuori della Terra) per l'ottimo piazzamento che regolarmente ottiene nell'annuale classifica "I 10 posti in cui si mangia peggio dell'Universo conosciuto" della Guida Galattica per gli Autostoppisti. Il giorno prima del ritorno, un'e-mail di SNCF (la società ferroviaria francese) lo informa che il treno che aveva prenotato, uno dei pochissimi disponibili, è stato soppresso a causa dei perduranti scioperi. Il PF deve quindi trovare un altro modo per tornare a casa; ma lo sciopero coinvolge in un modo o nell'altro tutti i mezzi di trasporto, per cui è impossibile sapere con certezza se un determinato trasferimento (treno, bus, aereo, metro, ...) sarà effettivamente disponibile. Senza perdersi d'animo, il PF si appresta ad applicare gli strumenti che conosce per cavarsi dall'impiccio. Descrive tutte le alternative di viaggio attraverso un grafo spazio-tempo $G = (N , A)$ i cui nodi sono coppie (luogo , orario), a partire da quelli della forma (Ass , t) per tutti i momenti t in cui un mezzo dovrebbe partire da *Ass*. Gli archi "viaggio" $(i , j) = ((s_i , t_j) , (s_j , t_j)) \in A$ rappresentano le corse previste in partenza dal luogo s_i al tempo t_i ed in arrivo al luogo $s_j \neq s_i$ al tempo $t_j > t_i$. Sono anche presenti archi "attesa" in cui $s_i = s_j$ (ed ancora $t_j > t_i$, corrispondente alla partenza di un qualche mezzo da s_i) che rappresentano lo stare fermi in attesa di una coincidenza. A ciascun arco (i , j) è associato il costo monetario $c_{ij} > 0$ (tipicamente nullo sugli archi "attesa", tranne quelli che richiedono un soggiorno in albergo) e la probabilità p_{ij} che l'arco sia effettivamente percorribile (< 1 sugli archi "attesa" se la coincidenza è "stretta" o gli alberghi in s_i sono "pieni"). Il PF ha un budget monetario $C > 0$ (molto ridotto, le università italiane essendo notoriamente sottofinanziate) per pagarsi il viaggio, e vuole che la probabilità di successo del viaggio, ossia la probabilità cumulata che tutti gli spostamenti pianificati vadano a buon fine, sia almeno del 90%. Si aiuti il povero PF a tornare a casa (probabilmente) sano e salvo scrivendo come *PLI* il problema di determinare gli spostamenti da fare minimizzando il tempo totale di viaggio, ossia quello tra l'effettiva partenza da *Ass* e l'effettivo arrivo a casa.

Per formulare il problema conviene considerare già aggiunti al grafo spazio-tempo una super-sorgente s, collegata a tutti i nodi (Ass , t), ed una super-destinazione d a cui sono collegati tutti i nodi (casa , t); il costo c_{ij} di tali archi "artificiali" si prende pari a 0, e la probabilità p_{ij} pari ad 1. Introducendo quindi le tipiche variabili binarie (di flusso) $x_{ij} \in \{ 0 , 1 \}$ per ogni $(i , j) \in A$, che indicano se il corrispondente arco (mezzo nel caso di archi viaggio, attesa per la coincidenza nel caso di quelli attesa) viene preso, una formulazione del problema è la seguente:

$$
\begin{aligned}
&\min \textstyle\sum_{(i , j) \in A \,:\, j = (\text{casa} , t)} t x_{ij} - \sum_{(i , j) \in A \,:\, i = (Ass , t)} t x_{ij} \\
&\sum_{(j , i) \in BS(i)} x_{ji} - \sum_{(i , j) \in FS(i)} x_{ij} = \begin{cases} -1 & \text{se } i = s \\ 1 & \text{se } i = d \\ 0 & \text{altrimenti} \end{cases} && i \in N \\
&\textstyle\sum_{(i , j) \in A} c_{ij} x_{ij} \leq C \\
&\textstyle\sum_{(i , j) \in A} \log(p_{ij}) x_{ij} \geq \log(0.9) \\
&x_{ij} \in \{ 0 , 1 \} && (i , j) \in A
\end{aligned}
$$

Per via dei vincoli di conservazione di flusso (primo blocco), le variabili x_{ij} descriveranno un cammino tra s e d sul grafo G, che è chiaramente aciclico (ogni arco descrive solamente spostamenti "avanti" nel tempo). In particolare possiamo assumere che non ci siano altri archi che entrano nei nodi (Ass , t) a parte quello proveniente da s;

pertanto esattamente uno di tali archi verrà scelto, ed analogamente per i nodi (casa, t). Di conseguenza la funzione obiettivo computa correttamente la differenza tra il tempo di arrivo a casa e quello di partenza da *Ass*. Il secondo vincolo esprime la condizione che il costo del viaggio sia compatibile col budget. Il terzo vincolo rappresenta la condizione sulla probabilità di successo; questa concettualmente è

$$\prod_{(i,j)\in A : x_{ij}=1} p_{ij} \geq 0.9$$

e si riconduce alla forma data prendendo il logaritmo (che, essendo una funzione monotona crescente, mantiene il verso delle diseguaglianze) su entrambi i lati ed osservando che

$$\log\left(\prod_{(i,j)\in A : x_{ij}=1} p_{ij}\right) = \sum_{(i,j)\in A : x_{ij}=1} \log(p_{ij}) = \sum_{(i,j)\in A} \log(p_{ij}) x_{ij} \ ,$$

dove la prima uguaglianza discende dalle proprietà del logaritmo e la seconda dal fatto che le x_{ij} sono variabili binarie. Ovviamente non vengono presi in considerazione archi con probabilità 0 (in effetti, < 0.9), e pertanto tutti i logaritmi sono ben definiti.

Il modello dell'esempio usa esplicitamente variabili binarie, ossia logiche. Questa restrizione è in realtà non necessaria nelle varianti "facili" di problemi di cammino ottimo; in effetti, il problema (MCF) ed i suoi casi particolari rivestono un ruolo particolare nell'interfaccia tra i problemi "facili" e quelli "difficili", come discusso nel Capitolo 2 e nel Capitolo 4. Comunque, la maggior parte dei problemi che hanno componenti di cammino ottimo sono $\mathcal{NP}$-hard, il che implica che il vincolo di integralità sulle variabili sia necessario. L'esempio introduce anche la tecnica modellistica del grafo spazio-tempo, che si rivela estremamente potente in molte applicazioni reali, in particolare legate alla logistica ed ai trasporti, nelle quali è necessario tenere in conto allo stesso tempo dell'aspetto geografico e di quello temporale degli spostamenti. Al di lá di questo aspetto specifico, comunque, i problemi di flusso o cammino sono importanti in molte applicazioni in quanto si possono costruire grafi che rappresentano situazioni complesse anche completamente scollegate da evidenti reti di trasporto o telecomunicazioni; alcuni esempi a tale proposito saranno forniti nel Capitolo 2.

1.2.7 Variabili a valori discreti

Come abbiamo visto in molti esempi precedenti, le variabili booleane hanno, nei modelli, una "doppia natura": da una parte vengono interpretate come variabili binarie, e quindi il loro valore viene associato a valori di verità, dall'altra sono normali variabili numeriche il cui dominio è limitato ad un insieme di valori discreti, per cui si può operare su di esse con le usuali operazioni aritmetiche. In questa e nelle successive sezioni mostreremo ulteriori esempi in cui questa "doppia natura" viene sfruttata.

In diversi casi è utile richiedere che una variabile possa assumere solo alcuni, specifici, valori, ossia imporre il vincolo $x \in V = \{v_1, v_2, \ldots, v_k\}$. In questo caso x è detta *variabile a valori discreti*. Questo è stato fatto in precedenza per $V = \{0, 1\}$, o più in generale V un intervallo di valori interi contigui, ma si vuole considerare il caso in cui i valori v_i possono essere completamente arbitrari. Ad esempio, se si desidera acquistare un transceiver per una rete in fibra ottica l'industria attualmente offre l'insieme finito di velocità $V = \{10, 25, 40, 50, 100, 400, 800\}$ (in Gb/sec) che non possono essere facilmente ricondotte ad un intervallo di valori interi contigui. Per esprimere la condizione $x \in V$ possiamo introdurre k variabili binarie $y_1, y_2, \ldots, y_k$ con il significato che, per ogni i, $y_i = 1$ se $x = v_i$, e $y_i = 0$ altrimenti. In questo modo (con il noto abuso di notazione), i vincoli

$$x = \sum_{i\in V} v_i y_i \qquad , \qquad \sum_{i\in V} y_i = 1 \qquad , \qquad y_i \in \{0, 1\} \quad i \in V$$

implementano la condizione $x \in V$ in un modello *PLI*. Un caso particolare, che accade spesso nelle applicazioni, è quello in cui $0 \in V$, ossia $0 = v_i$ per un qualche i (spesso $i = 1$). In questo caso è possibile semplificare leggermente il modello evitando di definire la variabile v_i corrispondente (ad esempio v_1) e trasformare il vincolo in $\sum_{i\in V'} y_i \leq 1$, dove $V' = V \setminus \{0\}$. Ovviamente, il fatto che $v_i = 1$ per ogni $i \in V'$ implica che è stato selezionato il valore 0 per x.

In questi casi si usa distinguere x, che è una *variabile strutturale* del modello—ovvero tra quelle che riflettono le decisioni fondamentali da assumere—dalle y_i che sono *variabili ausiliarie*, ovvero tra quelle che sono introdotte nel modello col solo scopo di permettere la formulazione di qualche specifica condizione. Questa distinzione è solo qualitativa, in quanto tutte le variabili di un modello (siano

esse binarie o quantitative) sono *variabili decisionali*, ossia che rappresentano decisioni effettive del problema, sia pure di "tipo" diverso.

Esempio 1.17. Il problema delle Tartaruge Ninja (parte II)

Nella realtà, le condotte per le fognature sono solitamente disponibili soltanto in un numero finito di portate diverse; nel nostro esempio, supponiamo che le condotte sono disponibili in tre sole dimensioni, con portate rispettivamente di 0.7, 1.4 e 3. In questo caso non possiamo più supporre che la portata della condotta installata sia uguale alla quantità di liquami effettivamente inviati: potrebbe infatti essere necessario installare condotte di portata maggiore. Per questo introduciamo un altro insieme di variabili a valori discreti $y_{ij} \in V = \{ 0 , 0.7 , 1.4 , 3 \}$, una per ogni arco (i , j) del grafo, che rappresentano la portata della condotta installata tra il nodo i ed il nodo j (0 se non viene installata nessuna condotta), ed inserire i vincoli

$$x_{ij} \leq y_{ij} \quad , \quad y_{ij} \in V \qquad (i , j) \in A \tag{1.22}$$

per garantire che la quantità di flusso inviata lungo ogni condotta non sia superiore alla portata della condotta effettivamente installata. Possiamo adesso trasformare questo problema in modo da sostituire le variabili quantitative y_{ij}, vincolate ad assumere un insieme discreto di valori, con variabili binarie. Per esprimere il vincolo (1.22) introduciamo, per ogni arco (i , j), tre nuove variabili binarie y^1_{ij}, y^2_{ij} e y^3_{ij} e sostituiamo il vincolo $y_{ij} \in V$ con

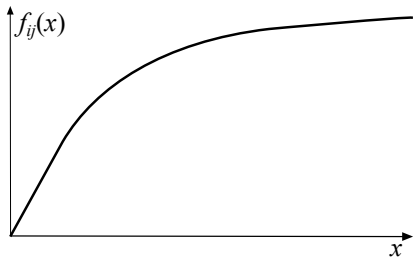


Figura 1.4: Funzione costo per il problema di progetto di rete

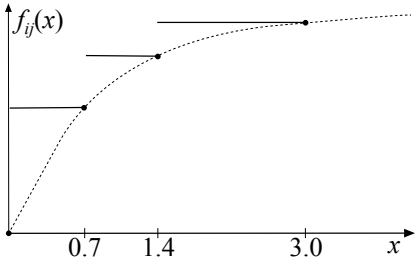


Figura 1.5: Una funzione costo a gradini

$$\begin{aligned} 0.7y^1_{ij} + 1.4y^2_{ij} + 3y^3_{ij} &= y_{ij} && (1.23)\\ y^1_{ij} + y^2_{ij} + y^3_{ij} &\leq 1 && (1.24)\\ y^1_{ij} \, , \, y^2_{ij} \, , \, y^3_{ij} &\in \{ 0 , 1 \} \end{aligned}$$

Utilizzando (1.23) possiamo poi sostituire y_{ij} ovunque con l'espressione (lineare) $0.7y^1_{ij} + 1.4y^2_{ij} + 3y^3_{ij}$, eliminando così tutte le variabili y_{ij} ed i vincoli (1.23) dal problema.

Questa operazione consente inoltre di risolvere "gratuitamente" un altro aspetto della modellazione che non rappresenta in modo accurato la realtà. Infatti, a causa delle economie di scala il costo di posa di una condotta non sarà lineare nella sua capacità (una condotta di capacità doppia costa il doppio), ma piuttosto una funzione *concava* del tipo di quella indicata in Figura 1.4, poiché installare una condotta di capacità doppia costa tipicamente (molto) meno del doppio. Pertanto, ad ogni $(i , j) \in A$ è associata una funzione $f_{ij}(x)$ che fornisce il costo globale dell'acquisto e posa in opera della condotta con portata x lungo tutto il tratto da i a j. Questo renderebbe il nostro problema di *ottimizzazione nonlineare*, con una funzione obiettivo della forma $\sum_{(i , j) \in A} f_{ij}(y_{ij})$, e quindi non più rappresentabile come *PLI*. Ma poiché y_{ij} è una variabile a valori discreti non siamo effettivamente interessati al valore delle funzioni f_{ij} in tutti i punti, ma solamente a quello nei punti $y \in V$. In altri termini è possibile formulare il problema eliminando completamente le variabili y_{ij} relative alle portate e considerando come costo per il flusso x_{ij} quello della più piccola sezione in commercio sufficiente a trasportarlo; nei fatti stiamo assumendo che il costo del flusso abbia la forma, detta *a gradini*, della funzione mostrata in figura 1.5. Infatti, oltre alle riformulazioni già viste, che portano ad eliminare le y_{ij} dappertutto tranne che in funzione obiettivo, possiamo considerare che

$$f_{ij}(0.7y^1_{ij} + 1.4y^2_{ij} + 3y^3_{ij}) = f_{ij}(0.7)y^1_{ij} + f_{ij}(1.4)y^2_{ij} + f_{ij}(3)y^3_{ij}$$

e quindi ottenere nuovamente una funzione obiettivo lineare, modellando questa variante più realistica del problema si progetto della rete fognaria come *PLI*. Tutto questo richiede però un significativo aumento del numero di variabili (in particolare binarie) del modello, il che, come avremo modo di discutere, può avere un impatto molto pesante sul costo di risoluzione dello stesso.

1.2.8 Variabili semicontinue e funzioni lineari a tratti

Un'ulteriore generalizzazione delle variabili a valori discreti è quella delle cosiddette *variabili semicontinue*. Si tratta di variabili che hanno "due regimi diversi": o sono fissate ad un valore discreto, tipicamente zero, oppure possono variare in un intervallo continuo che non contiene quel valore. Discutiamo per semplicità il solo caso in cui il vincolo sia formalmente $x \in V = \{ 0 \} \cup [l , u]$, dove $0 < l < u$. Questa situazione si incontra frequentemente nella pianificazione della produzione (o trasporto) di beni, in cui si ha un doppio livello di decisione: il primo livello sulla produzione o meno del bene, e il secondo livello sulla quantità di beni da produrre.

Per modellare i valori che x può assumere, introduciamo una variabile binaria y che assume il valore 0 se si decide di non produrre, e il valore 1 se si è deciso di produrre: possiamo allora modellare il vincolo $x \in V$ come

$$ly \le x \le uy \quad , \quad y \in \{\, 0\,,\, 1\,\} \quad , \quad x \in \mathbb{R} \ . \tag{1.25}$$

Infatti, se $y = 0$, la variabile x è "forzata" ad assumere il valore 0, mentre se $y = 1$, essa può assumere un qualsiasi valore reale nell'intervallo $[\, l\,,\, u\,]$.

Esercizio 1.15. Discutere la modellazione di una variabile semicontinua $x \in V = \{\, c\,\} \cup [\, l\,,\, u\,]$, con $c < l < u$ arbitrario. Discutere inoltre cosa cambierebbe nei modelli discussi se fosse invece $c > u > l$.

Abbiamo per adesso esaminato solo l'effetto di (1.25) sulla regione ammissibile, ma le variabili x ed y possono avere coefficienti (non nulli) in funzione obiettivo. Se questi sono rispettivamente c e b, ossia (il corrispondente frammento del)la funzione obiettivo è $cx + by$, il modello sta rappresentando una *funzione a carico fisso*

$$f(\,x\,) = \begin{cases} 0 & \text{se } x = 0 \\ b + cx & \text{se } 0 < x \le u \end{cases} \tag{1.26}$$

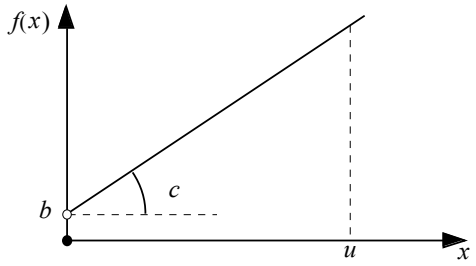


Figura 1.6: Una funzione con "carico fisso"

quale quella rappresentata in Figura 1.6. Si noti che in questo caso abbiamo assunto $l = 0$, scelta che sarebbe parsa insensata nella discussione precedente—in quanto $V = \{\, 0\,\} \cup [\, 0\,,\, u\,] = [\, 0\,,\, u\,]$—ma che invece è perfettamente ragionevole quando si consideri anche l'effetto sulla funzione obiettivo. In effetti la funzione a carico fisso $f(x)$ compare spesso ad esempio nei problemi di produzione: se non si produce niente ($x = 0$) allora non si paga alcun costo ($f(\,x\,) = 0$), mentre se si decide di produrre (con il limite della capacità produttiva u) allora si ha un *costo fisso*, o *costo di setup*, o *costo di investimento* b, fisso ed indipendente dalla produzione, e un *costo variabile*, o *costo di produzione* che dipende dalla quantità x effettivamente prodotta (in questo caso, linearmente). Si osservi che i vincoli introdotti non escludono la possibilità che sia contemporaneamente $y = 1$ e $x = 0$; in questo caso avremmo che il costo pagato per non produrre niente non è nullo come richiesto. Quindi il modello non riesce a rappresentare in modo univoco $f(\,x\,)$; tuttavia questo non è un problema se la funzione obiettivo viene minimizzata e $b > 0$, in quanto $b \cdot 1 + c \cdot 0 = b > b \cdot 0 + c \cdot 0 = 0$, e quindi la soluzione $[\, x\,,\, y\,] = [\, 0\,,\, 1\,]$ ha un valore peggiore di quella $[\, x\,,\, y\,] = [\, 0\,,\, 0\,]$ e non può essere quella ottima. Se il problema fosse di massimo, oppure se fosse $b < 0$, il modello non sarebbe corretto.

Il modello è chiaramente utilizzabile anche con $l > 0$; in questo caso si ha sia una funzione a carico fisso (si paga una quantità positiva per iniziare a produrre) che una minima quantità positiva prefissata (se si produce, si produce almeno $u > 0$). Si noti però che in questo caso il costo fisso è in effetti $b + cl$.

Esercizio 1.16. Disegnare e descrivere analiticamente la funzione $f(\,0\,) = 0$, e $f(\,x\,) = 180 + 2x$ per $0 < x \le 250$.

La trattazione precedente si estende facilmente al caso in cui la variabile x abbia "più di un tratto". Ad esempio, il vincolo $x \in V = \{\, 0\,\} \cup [\, l_1\,,\, u_1\,] \cup [\, l_2\,,\, u_2\,]$ per $0 \le l_1 < u_1 \le l_2 < u_2$ si traduce in

$$x = x_1 + x_2 \ , \ l_1 y_1 \le x_1 \le u_1 y_1 \ , \ l_2 y_2 \le x_2 \le u_2 y_2 \ , \ y_1 + y_2 \le 1 \ , \ [\, y_1\,,\, y_2\,] \in \{\, 0\,,\, 1\,\}^2 \tag{1.27}$$

Analogamente, consideriamo ad esempio la funzione a carico fisso a due tratti

$$f(\,x\,) = \begin{cases} 0 & \text{se } x = 0 \\ b_1 + c_1 x & \text{se } x \in [a_1, a_2] \\ b_2 + c_2 x & \text{se } x \in (a_2, a_3] \end{cases}$$

illustrata in Figura 1.7, dove assumiamo

$$b_2 + c_2 a_2 \ge b_1 + c_1 a_2. \tag{1.28}$$

La funzione $f(x)$ è definita in $V = \{0\} \cup [a_1, a_3]$ ed è la composizione di due funzioni lineari definite nei due sotto-intervalli $[a_1, a_2]$ ed $(a_2, a_3]$. Tale funzione è modellata dai vincoli (1.27) in cui si prenda $l_1 = a_1$, $u_1 = l_2 = a_2$ ed $u_2 = a_3$, unitamente alla funzione obiettivo

$$b_1 y_1 + c_1 x_1 + b_2 y_2 + c_2 x_2 \quad ,$$

purché al solito il problema sia di minimo. Infatti, come nel caso precedente il modello ha ambiguità nei punti in cui la funzione "salta" (non è continua), ossia in questo caso $x = a_2$. Tale punto è infatti rappresentabile mediante le due soluzioni diverse $[x, x_1, x_2, y_1, y_2] = [a_2, a_2, 0, 1, 0]$ oppure $[a_2, 0, a_2, 0, 1]$. La definizione richiede che, al fine di rappresentare correttamente il valore di $f(x)$, venga scelta la prima e non la seconda; questo accade sicuramente nella soluzione ottima, se il problema è di minimo, sotto la condizione (1.28). Il modello non sarebbe corretto se la condizione non valesse oppure il problema fosse di massimo, a meno che $f(x)$ non fosse definita diversamente (avendo valore $b_2 + c_2 a_2$ in $x = a_2$).

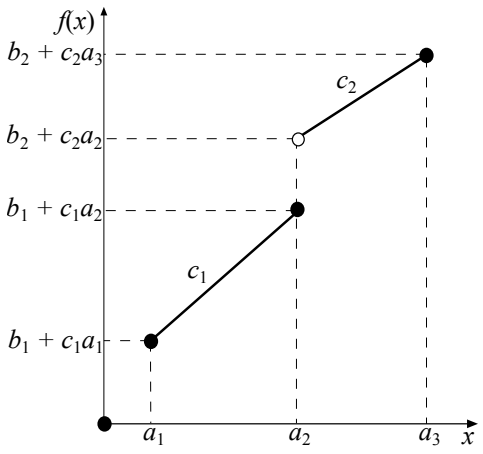


Figura 1.7: Una funzione lineare a due tratti

È ovvio come la trattazione si estenda immediatamente al caso in cui i due tratti della funzione non siano contigui, ossia risulti $u_1 < l_2$: semplicemente, x ha un "tratto proibito" (u_1, l_2) del quale non può assumere valori e nel quale quindi $f(x)$ non è definita. Può anche essere utile notare che il modello proposto non è l'unico possibile: ad esempio è si possono introdurre, in luogo delle x_1 ed x_2, variabili "di displacement" z_1 e z_2 che indicano il punto rispettivamente all'interno del primo e secondo intervallo in cui si sceglie x, se si è selezionato tale intervallo. Ciò porta al modello

$$\begin{array}{ll} \min & (b_1 + c_1 a_1) y_1 + c_1 z_1 + (b_2 + c_2 a_2) y_2 + c_2 z_2 \\ & x = a_1 y_1 + z_1 + a_2 y_2 + z_2 \\ & 0 \le z_1 \le (a_2 - a_1) y_1 \quad , \quad 0 \le z_2 \le (a_3 - a_2) y_2 \\ & y_1 + y_2 \le 1 \quad , \quad [y_1, y_2] \in \{0, 1\}^2 \end{array} \tag{1.29}$$

che si mostra facilmente essere completamente equivalente a quello già proposto, ma che può in certi casi essere preferibile come discusso nel seguito.

La tecnica modellistica presentata si generalizza in modo ovvio al caso di funzioni lineari a tratti definite su più di due intervalli, quali

$$f(x) = \begin{cases} b_1 + c_1 x & \text{se } x \in [a_1, a_2] \\ b_2 + c_2 x & \text{se } x \in (a_2, a_3] \\ \quad \vdots & \quad \vdots \\ b_n + c_n x & \text{se } x \in (a_n, a_{n+1}] \end{cases} \tag{1.30}$$

con condizioni analoghe a (1.28) nei punti di discontinuità (o le opposte nel caso di minimizzazione). Utilizzando ad esempio la variante (1.29) (l'altra è analoga), il modello diviene

$$\begin{array}{lll} \min & \sum_{i=1}^{n} (b_i + c_i a_i) y_i + \sum_{i=1}^{n} c_i z_i & \\ & x = \sum_{i=1}^{n} a_i y_i + \sum_{i=1}^{n} z_i & \\ & 0 \le z_i \le (a_{i+1} - a_i) y_i & i = 1, \ldots, n \quad . \\ & \sum_{i=1}^{n} y_i = 1 & \\ & y_i \in \{0, 1\} & i = 1, \ldots, n \end{array} \tag{1.31}$$

Esercizio 1.17. Formulare come *PLI* il problema di minimizzazione la funzione

$$f(x) = \begin{cases} 1+2x & \text{se } x \in [0,2] \\ 8-x & \text{se } x \in (2,5] \\ 3 & \text{se } x \in (5,6] \\ 2+x/2 & \text{se } x \in (6,10] \end{cases} .$$

1.2.9 Funzioni lineari a tratti "facili"

La necessità di utilizzare variabili binarie per rappresentare le funzioni lineari a tratti degli esempi precedenti deriva dalla loro *non convessità* (si veda il Capitolo 4); la formulazione "trasferisce" la non convessità del problema dalla funzione obiettivo all'insieme ammissibile per mezzo delle variabili a valori interi. Questo non è necessario qualora la funzione sia convessa, il che porta a formulazioni con un numero minore (o prive) di variabili binarie e quindi computazionalmente (molto) più efficienti. Illustriamo adesso diversi casi in cui questo è possibile.

L'intero approccio modellistico si basa su un'osservazione molto semplice: dati k numeri reali v_1, v_2, $\ldots$, v_k ed altri due numeri reali l ed u, allora

$$\begin{array}{rcl} l \leq \min\{\, v_i \,:\, i = 1, \ldots, k \,\} & \equiv & l \leq v_i \quad i = 1, \ldots, k \\ u \geq \max\{\, v_i \,:\, i = 1, \ldots, k \,\} & \equiv & u \geq v_i \quad i = 1, \ldots, k \end{array} . \tag{1.32}$$

È importante notare come il segno delle diseguaglianze e le condizioni max e min *non* siano interscambiabili in (1.32); ossia, le condizioni

$$l \leq \min\{\, v_i \,:\, i = 1, \ldots, k \,\} \;,\; u \geq \max\{\, v_i \,:\, i = 1, \ldots, k \,\}$$

non sono facilmente implementabili in *PLI*, ancorché questo sia in effetti possibile con le tecniche che vedremo in seguito (cf. §1.2.10). Tali tecniche però richiedono l'uso di variabili binarie, che vogliamo evitare in questa sezione.

Nella *PLI* è sempre consentito utilizzare funzioni lineari (affini) al posto di variabili singole; ossia, si può sempre considerare una variabile v come il valore di una funzione lineare affine $f(x) = \sum_{j=1}^{n} a_j x_j + b$ in n altre variabili x_j, $j \in J = \{1, \ldots, n\}$, semplicemente inserendo il vincolo

$$v = \textstyle\sum_{j \in J} a_j x_j + b \;.$$

Come già notato in passato, in questi casi si può anche sostituire la variabile v con l'espressione lineare nell'intero modello, eliminandola. Con questo ragionamento si ottiene l'immediata generalizzazione del concetto precedente: dato un insieme di funzioni lineari (affini)

$$F = \left\{\, f_i(x) = \textstyle\sum_{j \in J} a^i_j x_j + b^i \,,\, i = 1, \ldots, k \,\right\}$$

allora (con l'ormai ben noto abuso di notazione)

$$\begin{array}{rcl} l \leq \min\{\, f_i(x) \,:\, i \in F \,\} & \equiv & l \leq \sum_{j \in J} a^i_j x_j + b^i \quad i \in F \\ u \geq \max\{\, f_i(x) \,:\, i \in F \,\} & \equiv & u \geq \sum_{j \in J} a^i_j x_j + b^i \quad i \in F \end{array} . \tag{1.33}$$

Le condizioni (1.32) permettono di costruire facilmente vincoli (di $\leq$) sul minimo o (di $\geq$) sul massimo di funzioni lineari quando l ed u siano costanti, oppure altre funzioni lineari delle variabili del problema. Mediante un semplice trucco modellistico questo può anche essere utilizzato per esprimere (nelle giuste condizioni) funzioni obiettivo che siano il minimo od il massimo di funzioni lineari. Il trucco in questione è semplicemente

$$\min\{\, f(x) \,\} \;\equiv\; \min\{\, v \,:\, v \geq f(x) \,\} \quad , \quad \max\{\, f(x) \,\} \;\equiv\; \max\{\, v \,:\, v \leq f(x) \,\} \;.$$

Ossia, si può sempre trasformare un (frammento di) funzione obiettivo $f(x)$ abitrariamente complessa (in particolare, non necessariamente lineare) in una funzione lineare introducendo una nuova variabile (ausiliaria) v, sostituendola ad $f(x)$ in funzione obiettivo, ed introducendo un vincolo che la collega in modo opportuno al valore della funzione. Ancora, è cruciale che il verso del vincolo ($\geq$ o $\leq$) sia scelto in modo corretto rispetto al verso della funzione obiettivo (min o max) altrimenti la tecnica

fallisce completamente. In questo modo, (1.33) permette di ottenere

$$\begin{array}{rcl} \min\{\, f(\,x\,) = \max_{i\in F}\{\, \sum_{j\in J} a^i_j x_j + b^i \,\}\,\} & \equiv & \min\{\, v \,:\, v \geq \sum_{j\in J} a^i_j x_j + b^i \quad i \in F \,\} \\ \max\{\, f(\,x\,) = \min_{i\in F}\{\, \sum_{j\in J} a^i_j x_j + b^i \,\}\,\} & \equiv & \max\{\, v \,:\, v \leq \sum_{j\in J} a^i_j x_j + b^i \quad i \in F \,\} \end{array} \quad . \tag{1.34}$$

In altri termini, i problemi cosiddetti "min-max" e 'max-min", in cui le funzioni obiettivo sono massimi o minimi di funzioni lineari, sono facilmente esprimibili (nelle giuste condizioni) in *PLI*; in effetti, in *PL*. Vediamo adesso alcune applicazioni di questa tecnica modellistica.

Esempio 1.18. Ordinamento di lavori su macchine: minimizzazione del tempo di completamento

Attraverso i ben oliati (e pagati) contatti con il sottobosco dell'Amministrazione Imperiale, la HSC ha avuto una soffiata decisiva: Darth Vader in persona si è insospettito delle catastrofiche perdite economiche che l'Impero sta avendo su Tatooine e sta arrivando alla guida di una "commissione di inchiesta" (sostanzialmente un plotone di esecuzione). È quindi tempo di chiudere le operazioni sul pianeta, ma non prima di averne spremuto tutto il denaro possibile. Per l'ultimo giorno quindi la HSC cambia strategia: non è più importante utilizzare il minor numero di moli spaziali (si daranno alla fuga prima di pagare i funzionari corrotti, che in ogni caso entro la fine della giornata saranno morti) ma terminare le operazioni con i trasporti disponibili il più in fretta possibile per non rischiare di incappare nel temibile Sith. Per questo i trasporti in arrivo a breve, $T = \{\, 1\,,\, 2\,,\, \ldots\,,\, n\,\}$ devono essere assegnati ai moli disponibili (tutti quelli di Mos Eisley) in modo da finire le operazioni di scarico (il momento in cui si alza in volo l'ultimo trasporto servito) il più rapidamente possibile. Si assume che tutti i trasporti siano immediatamente disponibili e che possano essere fatti attendere in orbita fino al momento in cui sono serviti, ma resta il fatto che una volta atterrati essi devono rimanere nel molo per tutto il periodo d_i necessario alle operazioni, e che nessun molo può ospitare più di un trasporto allo stesso tempo. Aiutate la HSC a mettere a segno il suo ultimo rocambolesco colpo su Tatooine, sfuggendo per un pelo a Darth Vader, scrivendo come *PLI* il problema di assegnare i trasporti ai moli disponibili in modo da minimizzare il tempo di completamento.

Questo problema è una variante di quello dell'Esempio 1.12, di cui riutilizziamo la notazione. La differenza è che, indicando con $T(m)$ è l'insieme dei trasporti assegnati al molo $m \in M$, il tempo in cui il molo rimane occupato è $D(m) = \sum_{i\in T(m)} d_i$ (si suppone che il trsporto successivo venga fatto atterrare non appena parte il precedente). Il *tempo di completamento* (*makespan*) T, da minimizzare, è quindi il massimo dei tempi tra tutti i moli

$$T = \max\{\, D(m) \;:\; m \in M \,\}\,,$$

ossia il tempo necessario al molo più carico per terminare. Utilizziamo le stesse variabili x_{im} utilizzate in (MCMS) e gli stessi vincoli di semi-assegnamento (1.16), assieme ai vincoli di integralità, per imporre che ciascun trasporto sia assegnato da uno ed un solo molo. Per formulare la funzione obiettivo, da minimizzare, utilizziamo la tecnica generale (1.34); questo porta alla formulazione

$$\text{(MMMS)} \qquad \begin{array}{lll} \min & t & \\ & \sum_{m\in M} x_{im} = 1 & i \in T \\ & \sum_{i\in T} d_i x_{im} \leq t & m \in M \\ & x_{im} \in \{0,1\} & i \in T \;,\; m \in M \end{array} \quad .$$

Ovviamente, in qualsiasi soluzione ottima del problema la variabile ausiliaria t fornisce il tempo di completamento e non una sua approssimazione superiore, come i vincoli in teoria permetterebbero: infatti, se per assurdo in una soluzione ottima si avesse $t > D(m)$ per ogni $m \in M$, si otterrebbe un valore inferiore della funzione obiettivo ponendo $t = \max\{\, D(m) \;:\; m \in M \,\}$, contraddicendo la presunta ottimalità della soluzione originaria.

Questo problema di ordinamento di lavori su macchine con minimizzazione del tempo di completamento (MMMS, da *Minimal Makespan Machine Scheduling*), è, insieme a quello dell'Esempio 1.12, una delle tante varianti di problemi si schedulazione di lavori su macchine che trovano ampio uso nelle applicazioni. I problemi reali hanno tipicamente vincoli più complessi: le macchine non sono identiche ed il tempo di completmento dei lavori differisce a seconda della macchina a cui è assegnato, non tutte le macchine possono compiere tutti i lavori, ci sono vincoli di precedenza sui lavori, e molti altri. Tutti questi vincoli sono tipicamente esprimibili, in modo più o meno semplice, attraverso modelli *PLI*.

Esercizio 1.18. Costruire un'istanza del problema (MMMS) con 3 macchine e 7 lavori, definendo le durate di essi; formulare quindi l'istanza, fornire tre soluzioni ammissibili e valutarne il tempo di completamento.

Esempio 1.19. Valore assoluto e norme "lineari"

La funzione $|\,x\,|$—valore assoluto di x—si presta all'applicazione della tecnica (1.33) / (1.34) in quanto $|\,x\,| = \max\{\, x\,,\, -x\,\}$; di conseguenza

$$|\,x\,| \leq b \;\equiv\; -b \leq x \leq b \quad \text{e} \quad \min\{\,|\,x\,|\,\} \;\equiv\; \min\{\, v \,:\, v \geq x \;,\; v \geq -x \,\}\;.$$

Ovviamente la tecnica si adatta immediatamente quando x è rimpiazzato da una funzione lineare (affine).

Esercizio 1.19. Formulare il problema $\min\{\,|\,3-4x\,|\,:\,|\,x\,|\le 2\,\}$.

Esercizio 1.20. Siano $a_1,\ldots,a_n$ numeri reali positivi. Partizionare tali numeri in due insiemi I e J in modo tale che le somme dei valori assegnati a ciascun sottoinsieme abbiano la minima differenza in valore assoluto.

La generalizzazione del concetto di valore assoluto a $\mathbb{R}^n$ è quella di *norma* del vettore $x=[\,x_i\,]_{i=1}^n\in\mathbb{R}^n$. Le norme più comunemente usate sono

$$L_1(\,x\,)=\|\,x\,\|_1=\textstyle\sum_{i=1}^n|\,x_i\,|\quad,\quad L_2(\,x\,)=\|\,x\,\|_2=\sqrt{\textstyle\sum_{i=1}^n x_i^2}\quad,\quad L_\infty(\,x\,)=\|\,x\,\|_\infty=\max\{\,|\,x_i\,|\,:\,i=1\,,\,\ldots\,,\,n\,\}\ .$$

Tra di esse, la norma L_1 e la L_∞ sono particolarmente utili perché sono facilmente esprimibili come PL utilizzando le tecniche appena introdotte. Infatti, per qualsiasi $X\subset\mathbb{R}^n$, il problema di determinare l'elemento di norma rispettivamente L_1 ed L_∞ minima in X si scrive come

$$\begin{array}{ll}(PMD_1) & \min\left\{\ \textstyle\sum_{i=1}^n v_i\ :\ -v_i\le x_i\le v_i\ \ i=1,\ldots,n\ ,\ x\in X\ \right\}\\ (PMD_\infty) & \min\left\{\ v\ :\ -v\le x_i\le v\ \ i=1,\ldots,n\ ,\ x\in X\ \right\}\end{array}$$

Se X è rappresentabile attraverso vincoli lineari, ed eventualmente di integralità, i due problemi possono quindi essere formulati come PL o PLI. Data una qualsiasi norma $|||\,x\,|||$, la *distanza* tra due punti x ed y secondo quella norma è definita come $|||\,x-y\,|||$ (nel caso della norma L_1 questa viene anche indicata come *distanza Manhattan*); pertanto, i due problemi appena visti si trasformano facilmente nel problema di determinare il punto di X più vicino (secondo le norme specificate) ad un punto $\bar x\in\mathbb{R}^n$ dato, che ha molte applicazioni. Si lascia come facile esercizio di esprimere come PL i vincoli $\|\,x\,\|_1\le b$ e $\|\,x\,\|_\infty\le b$ e la loro generalizzazione $\|\,x-\bar x\,\|_1\le b$ e $\|\,x-\bar x\,\|_\infty\le b$.

Il fatto che le funzioni discusse in questa sezione si possano rappresentare senza l'ausilio di variabili intere è intimamente collegato alla loro *convessità*. In particolare,

$$f(\,x\,)=\max\{\,\textstyle\sum_{j\in J}a_j^ix_j+b^i\ :\ i\in F\,\}$$

è massimo di (un numero finito di) funzioni lineari e quindi convessa; infatti, (1.34) mostra che il suo *epigrafo* $epi(f)=\{\,(\,v\,,\,x\,)\ :\ v\ge f(\,x\,)\,\}$ è un *poliedro convesso* (cf. Capitolo 3). Fino ad ora abbiamo sempre visto le funzioni di questo tipo definite esplicitamente in termini del massimo di funzioni lineari. Ma le funzioni convesse di una sola variabile reale, come quella mostrata in Figura 1.8, ammettono anche una diversa descrizione come PL, analoga a (1.29), che alle volte risulta più comoda da usare. La funzione è definita esplicitamente per casi, come in (1.30), con le seguenti condizioni:

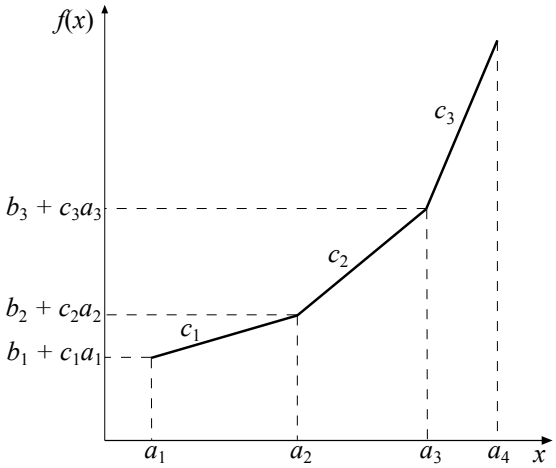


Figura 1.8: Una funzione lineare a tratti e convessa

- f è continua, ossia $b_{i+1}+c_{i+1}a_{i+1}=b_i+c_ia_{i+1}$ per $i=1,\ldots,n-1$;
- la derivata di f (nei tratti lineari) deve essere nondecrescente, ossia $c_{i+1}\ge c_i$ per $i=1,\ldots,n-1$.

In questo caso, $\min\{\,f(\,x\,)\,\}$ può essere equivalentemente espresso come

$$\begin{array}{ll}\min & b_1+\sum_{i=1}^n c_iz_i\\ & x=a_1+\sum_{i=1}^n z_i\\ & 0\le z_i\le a_{i+1}-a_i\quad i=1,\ldots,n\end{array}\quad . \tag{1.35}$$

Infatti, se all'ottimo la variabile x assume il valore $\bar x$, tale valore dovrà essere "ottenuto" aumentando il valore di alcune delle variabili z_i finché la loro somma non dia $\bar x-a_1$. Ma siccome la derivata di f è non decrescente, è chiaramente conveniente "far crescere prima il valore della variabili z_i di indice più basso"; in altri termini, in una soluzione ottima del problema si avrà certamente che

$$z_i=\begin{cases}a_{i+1}-a_i & \text{se } i<h\\ \bar x-a_h & \text{se } i=h\\ 0 & \text{se } i>h\end{cases}$$

dove h è il più piccolo indice tale che $\bar{x} \geq a_h$. È facile verificare che da questo discende la correttezza della formulazione.

Esercizio 1.21. Dimostrare le affermazioni precedenti.

Questa proprietà, e quindi la formulazione alternativa, non vale nel caso in cui $f(\cdot)$ non sia convessa, o nel caso in cui venga massimizzata. Un'analoga formulazione senza variabili binarie è possibile per la massimizzazione—ma non per la minimizzazione—di funzioni lineari a tratti *concave*, ossia continue e la cui derivata sia non crescente.

Proponiamo adesso un esercizio riassuntivo relativo alla modellizzazione di funzioni lineari a tratti di una sola variabile reale.

Esempio 1.20. Il problema della Fonderia (parte II)

In questa variante dell'Esempio 1.4, la fonderia deve produrre 1000 lingotti del peso ciascuno di un chilogrammo. Per questo può comprare il ferro da tre diversi fornitori, ciascuno dei quali propone condizioni di vendita differenti:

A fino a 350kg il ferro costa 0.03 €/kg, oltre 350kg e fino a 750kg costa 0.05 €/kg, oltre 750kg costa 0.08 €/kg;

B fino a 600kg il ferro costa 0.04 €/kg, oltre 600kg costa 0.02 €/kg, ma ordini maggiori di 450kg e minori di 600kg non possono essere accettati;

C pagando 10€ si può ordinare qualsiasi quantità di ferro compresa tra 100kg e 400kg; oltre i 400kg, il ferro costa 0.06 €/kg, mentre non si possono ordinare quantità inferiori ai 100kg.

Si vuole scrivere in forma di *PLI* il problema di determinare il piano di acquisto dai tre diversi fornitori che minimizza il costo totale del ferro. Ovviamente il problema potrebbe essere reso arbitrariamente più complesso, ad esempio considerando le percentuali di Manganese presenti nel ferro come nell'Esempio 1.4, ma questo renderebbe la trattazione poco didattica ed è lasciato per esercizio.

Introduciamo le variabili x_A, x_B e x_C che rappresentano la quantità in kg rispettivamente del ferro comprato da A, B e C. Il costo di acquisto di ciascun fornitore è una funzione lineare a tratti, per rappresentare la quale è necessario introdurre variabili ausiliarie. In particolare, la funzione lineare a tratti del fornitore A è convessa, ossia è continua e la derivata è crescente ($0.03 < 0.05 < 0.08$) Quindi, per rappresentarla è sufficiente suddividere x_A nella somma delle tre variabili ausiliarie continue x'_A, x''_A ed x'''_A, ciascuna col suo costo, utilizzando la formulazione (1.35) basata sul fatto che x'_A, corrispondente all' intervallo più a sinistra, è la meno costosa e quindi verrà automaticamente "riempita prima" di x''_A, ed analogamente per quest'ultima nei confronti di x'''_A. La funzione lineare a tratti del fornitore B è invece concava ($0.04 > 0.02$, ossia la derivata del tratto più a destra è minore di quella del tratto più a sinistra), e presenta una regione proibita; pertanto per rappresentarla sono necessarie le variabili continue x'_B e x''_B e le variabili binarie y'_B e y''_B, come indicato in (1.31). Similmente, la funzione lineare a tratti del fornitore C ha un costo fisso pari a 10€, ed un primo tratto "orizzontale", in cui materiale ha costo unitario nullo, fino a 400kg; ha inoltre la minima quantità positiva prefissata di 100kg. Quindi, utilizzando (1.31) richiede, per essere modellata, le variabili continue x'_C e x''_C e le variabili binarie y'_C e y''_C. Complessivamente modello per il problema è quindi

$$
\begin{aligned}
\min\ & 0.03x'_A + 0.05x''_A + 0.08x'''_A + 0.04x'_B + 24y''_B + 0.02x''_B + 10y'_C + 10y''_C + 0.06x''_C \\
& x_A + x_B + x_C = 1000 \\
& x'_A + x''_A + x'''_A = x_A \\
& 0 \leq x'_A \leq 350 \quad , \quad 0 \leq x''_A \leq 400 \quad , \quad 0 \leq x'''_A \leq 250 \\
& x'_B + 600y''_B + x''_B = x_B \\
& 0 \leq x'_B \leq 450y'_B \quad , \quad 0 \leq x''_B \leq 400y''_B \quad , \quad y'_B + y''_B \leq 1 \\
& 100y'_C + x'_C + 400y''_C + x''_C = x_C \\
& 0 \leq x'_C \leq 300y'_C \quad , \quad 0 \leq x''_C \leq 600y''_B \quad , \quad y'_C + y''_C \leq 1 \\
& y'_B \in \{0, 1\} \ , \ y''_B \in \{0, 1\} \ , \ y'_C \in \{0, 1\} \ , \ y''_C \in \{0, 1\}
\end{aligned}
$$

La funzione obiettivo rappresenta il costo totale del ferro, ossia la somma delle tre funzioni lineari a tratti. Il secondo vincolo assicura che il numero totale di kg prodotti sia 1000. Tutti gli altri vincoli descrivono le tre funzioni lineari a tratti come precedentemente discusso. Come precedentemente osservato i vincoli che definiscono x_A, x_B ed x_C in termini delle loro variabili ausiliarie potrebbero essere utilizzati per eliminarle dalla formulazione, ottenendone una in qualche modo più compatta, ma questo non è stato fatto per maggior chiarezza.

1.2.10 Vincoli disgiuntivi

Discutiamo adesso un'ultima tecnica modellistica che in qualche modo riassume in sé molte di quelle viste finora. Come vedremo più in dettaglio nel §3.1.1, nei modelli di *PL* (o *PLI*) si hanno un numero

finito di vincoli lineari del tipo

$$A_i x \leq b_i \quad , \quad i = 1, \ldots, m \tag{1.36}$$

dove $A_i \in \mathbb{R}^n$, ossia i vincoli sono definiti attraverso n funzioni lineari. Gli insiemi definiti in questo modo sono detti *poliedri convessi*. Ad esempio, il sistema

$$\begin{array}{rcrcl} x_1 & + & x_2 & \leq & 2 \\ & & x_2 & \leq & 1 \\ x_1 & & & \leq & 2 \\ x_1 & & & \geq & 0 \\ & & x_2 & \geq & 0 \end{array}$$

definisce il poliedro di vertici $[\,0\,,\,0\,]$, $[\,0\,,\,1\,]$, $[\,1\,,\,1\,]$ e $[\,2\,,\,0\,]$ in Figura 1.9.

Supponiamo ora che l'insieme ammissibile che si vuole rappresentare sia tutta l'area grigia: si tratta di una regione *non convessa*, che può essere rappresentata come l'*unione* (invece dell'intersezione) di due poliedri convessi, il triangolo di vertici $[\,0\,,\,0\,]$, $[\,2\,,\,0\,]$ e $[\,0\,,\,2\,]$ ed il rettangolo $[\,0\,,\,0\,]$, $[\,0\,,\,1\,]$, $[\,2\,,\,1\,]$ e $[\,2\,,\,0\,]$. Anche ora, come in casi precedenti, si presenta una scelta fra due alternative: le soluzioni ammissibili appartengono *o* al primo poliedro, *oppure* al secondo. In particolare, sono il primo ed il secondo vincolo ad essere critici, poiché tutti gli altri sono soddisfatti da tutti i punti della regione che vogliamo rappresentare; invece i punti ammissibili possono soddisfare *anche uno solo* di quei due vincoli. Si parla quindi in questo caso di *vincoli disgiuntivi*. Per rappresentare vincoli di questo tipo è naturale introdurre due variabili binarie y_1 ed y_2, con la convenzione che $y_1 = 0$ significa che x appartiene al primo insieme, e quindi il secondo vincolo può essere violato, mentre $y_2 = 0$ significa che x appartiene al secondo insieme, e quindi è il primo vincolo a poter essere violato. Possiamo quindi rappresentare l'insieme ammissibile per mezzo dei vincoli

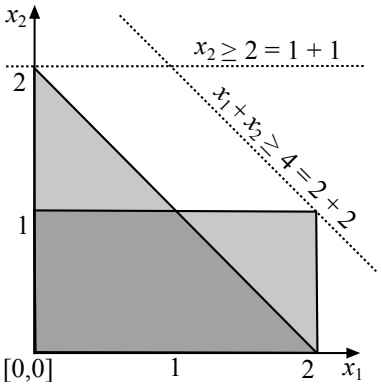


Figura 1.9: Rappresentazione di poliedri non convessi

$$\begin{array}{rcrcrcrcl} x_1 & + & x_2 & - & M_1 y_1 & & & \leq & 2 \\ & & x_2 & & & - & M_2 y_2 & \leq & 1 \\ x_1 & & & & & & & \leq & 2 \\ x_1 & & & & & & & \geq & 0 \\ & & x_2 & & & & & \geq & 0 \\ & & & & y_1 & + & y_2 & \leq & 1 \\ & & & & y_1 & , & y_2 & \in & \{0,1\} \end{array}$$

purché M_1 ed M_2 siano numeri "sufficientemente grandi" da rendere *ridondanti* i vincoli quando la variabile binaria corrispondente assume il valore 1. Si può facilmente verificare dalla figura che in questo caso specifico è sufficiente porre $M_1 = 2$ e $M_2 = 1$. Si noti che il vincolo $y_1 + y_2 \leq 1$ assicura che al più una delle variabili abbia valore 1, ossia che almeno uno dei due insiemi di vincoli sia soddisfatto; alternativamente si può usare una sola variabile binaria $y = y_1$ ponendo $y_2 = 1 - y$.

Più in generale, consideriamo il caso in cui si abbiano gli m vincoli (1.36), e che S_1, S_2, ..., S_p siano p sottoinsiemi, non necessariamente disgiunti, dell'insieme $\{1, 2, \ldots, m\}$. Definiamo per ogni $h = 1, \ldots, p$ l'insieme $X_h = \{\, x \,:\, A_i x \leq b_i \;\; i \in S_h \,\}$ di tutti i punti che soddisfano i vincoli i cui indici sono in S_h, e consideriamo l'insieme $X = X_1 \cup X_2 \cup \ldots \cup X_p$. Se tutti gli insiemi X_h sono limitati possiamo rappresentare X introducendo p variabili binarie e generalizzando l'idea vista in precedenza

per il caso $p = 2$: X è l'insieme di tutti i vettori x che soddisfano i vincoli

$$\begin{array}{lllllllll} A_i x & - & M_1 y_1 & & & & & \leq b_i & i \in S_1 \\ A_i x & & & - & M_2 y_2 & & & \leq b_i & i \in S_2 \\ \vdots & & & & & \ddots & & \vdots & \vdots \\ A_i x & & & & & & - \; M_p y_p & \leq b_i & i \in S_p \\ & y_1 & + & y_2 & \cdots & + & y_p & \leq p-1 & \\ & y_1 & , & y_2 & \cdots & , & y_p & \in \{0\,,\,1\} & \end{array} .$$

Per ogni $h = 1, \ldots, p$, M_h è una costante tale che tutti i vincoli $A_i x \leq b_i + M_h$ per $i \in S_h$ sono ridondanti per tutti gli insiemi S_k, $k \neq h$; una tale costante esiste certamente perché tutti gli insiemi X_k sono limitati per ipotesi.

Esercizio 1.22. Si proponga una procedura che permetta di determinare un valore opportuno per ciascuna costante M_h (suggerimento: si risolva un numero opportuno di problemi di PL).

Una variante interessante è il caso in cui si vuole che almeno k degli insiemi di vincoli siano soddisfatti (cioè x appartenga all'intersezione di almeno k degli insiemi X_i, $i = 1, \ldots, p$). In questo caso è sufficiente sostituire al vincolo $y_1 + y_2 + \cdots + y_p \leq p - 1$ il nuovo vincolo $y_1 + y_2 + \cdots + y_p \leq p - k$.

Proponiamo adesso alcuni esempi di formulazioni che usano vincoli disgiuntivi.

Esempio 1.21. Il problema della Fonderia (parte III)

Una fonderia deve produrre 1000 lingotti del peso ciascuno di un chilogrammo. Per questo può utilizzare tre tipi di materiale ferroso, con le seguenti caratteristiche:

Materiale ferroso	A	B	C
Silicio (%)	4.00	1.00	6.00
Manganese (%)	0.45	0.50	0.40
Costo (€/ kg.)	0.03	0.04	0.02

Inoltre si può aggiungere direttamente manganese al costo di 10 Euro al kg. Per essere vendibile sul mercato, il ferro con cui i lingotti sono fatti dovrà avere *almeno una* delle tre seguenti proprietà: i) contenere almeno lo 0.45% di manganese; ii) contenere non più del 3.25% di silicio; iii) contenere almeno il 5.5% di silicio. Se però il ferro soddisfa *almeno due* delle proprietà, allora la fonderia otterrà un bonus pari a 127 Euro sul prezzo di vendita totale rispetto al caso in cui sia soddisfatta una sola proprietà. Si scriva in forma di *PLI* il problema di determinare il piano di produzione che minimizza il costo del materiale utilizzato al netto dell'eventuale bonus ottenuto.

Il problema è ovviamente una variante di quello dell'Esempio 1.4 in cui ci sono condizioni disgiuntive sulle proprietà richieste al materiale (almeno una, almeno due, ...). Possiamo utilizzare quindi le stesse variabili continue dell'Esempio originale, ma dobbiamo introdurre anche tre variabili binarie y_i, y_{ii} e y_{iii} che indicano rispettivamente se il ferro rispetta le proprietà i), ii) e iii). Notiamo a questo proposito che la quantità di silicio non potrà comunque essere maggiore del 6% (se si utilizzasse il solo materiale C), né minore dello 0% (se si utilizzasse il solo manganese, che ovviamente non contiene silicio). Analogamente, il manganese non potrà comunque essere minore dello 0.4% (ancora, se si utilizzasse il solo materiale C). Infine introduciamo la variabile binaria z che indica se si ottiene il bonus. Un modello per il problema è quindi:

$$\begin{aligned} \min\ & 0.03x_A + 0.04x_B + 0.02x_C + 10x_M - 127z \\ & x_A + x_B + x_C + x_M = 1000 \\ & 0.0045x_A + 0.005x_B + 0.004x_C + x_M \geq 4.0 + 0.5y_i \\ & 0.04x_A + 0.01x_B + 0.06x_C \leq 32.5 + 27.5(1 - y_{ii}) \\ & 0.04x_A + 0.01x_B + 0.06x_C \geq 55y_{iii} \\ & y_i + y_{ii} + y_{iii} \geq 1 \\ & 2z \leq y_i + y_{ii} + y_{iii} \\ & x_A \geq 0 \ , \ x_B \geq 0 \ , \ x_C \geq 0 \ , \ x_M \geq 0 \\ & y_i \in \{0\,,\,1\} \ , \ y_{ii} \in \{0\,,\,1\} \ , \ y_{iii} \in \{0\,,\,1\} \ , \ z \in \{0\,,\,1\} \end{aligned}$$

La funzione obiettivo rappresenta il costo totale del materiale, a cui viene sottratto il bonus nel caso in cui venga ottenuto. Come nell'esempio originale il primo vincolo assicura che il numero totale di kg prodotti sia 1000. Il vincolo successivo assicura che la quantità di manganese presente nel prodotto risultante non sia inferiore a 4.5

kg (lo 0.45% di 1000 kg) se $y_i = 1$, mentre è ridondante se $y_i = 0$ perché, come notato in precedenza, 1000 kg di qualsiasi mix di materiale contengono sicuramente almeno 4 kg di manganese. Analogamente, il vincolo ancora successivo assicura che la quantità di silicio sia minore di 32.5 kg (il 3.25% di 1000 kg) se $y_{ii} = 1$, mentre è ridondante se $y_{ii} = 0$ (1000 kg di qualsiasi mix di materiale contengono comunque al massimo 60 kg di silicio). Allo stesso modo, il quarto vincolo assicura che la quantità di silicio sia maggiore di 55 kg (...) se $y_{iii} = 1$, mentre è ridondante se $y_{iii} = 0$. Tutto ciò implica che il quinto vincolo assicuri che almeno una delle tre condizioni debba valere; analogamente, il sesto garantisce che z possa valere 1 (e quindi, dato il segno del coefficiente in funzione obiettivo, che sicuramente lo farà in ogni soluzione ottima) solamente se almeno due delle condizioni i), ii) e iii) valgono. È anche ovvio che le condizioni ii) e iii) sono mutuamente esclusive, e quindi non è possibile che più di due tra le tre variabili valgano contemporaneamente 1, ma questo non è rilevante per le richieste fatte dal testo.

Esempio 1.22. Un problema di impaccamento (parte II)

Nel regno Spigoloso del mondo bidimensionale di Flatlandia, il famoso geometra Eucli2D—già praticamente una star per via dei suoi risultati sugli angoli, l'argomento favorito del regno—intende cementare definitivamente la sua fama risolvendo il problema dell'impaccamento ottimo di sfere: dato il numero naturale n, determinare il massimo valore r per cui sia possibile impaccare n sfere di raggio r nel quadrato unitario senza che si sovrappongano, tranne al più sulla frontiera. Trattandosi di Spigoloso le sfere possono essere soltanto in norma L_1, e trattandosi di Flatlandia il problema può essere pensato solamente in 2D ($\| x \|_1 = | x_1 | + | x_2 |$ per $x \in \mathbb{R}^2$). Si aiuti Eucli2D ad ottenere fama imperitura in tutto il regno di Spigoloso formulando come *PLI* il problema corrispondente.

Si tratta evidentemente della versione dell'Esempio 1.2 che usa la norma "spigolosa" L_1 invece di quella "rotonda" (Euclidea) L_2. Per costruire il modello introduciamo, come nell'Esempio originale, $2n$ variabili continue x_i ed y_i per $i = 1, \ldots, n$ che indicano le coordinate del centro dell'i-esima sfera (in norma L_1). Sia C l'insieme di tutte le coppie $\{ i , j \}$ non ordinate di sfere (la relazione è simmetrica): imporre che i e j non si intersechino equivale ad imporre che nessuno dei punti della sfera i appartenga alla sfera j. Di conseguenza, il punto in cui il modello si differenzia da quello originale è chiaramente l'unico blocco di vincoli in cui si usa esplicitamente la norma, ossia quelli della forma

$$\left\| \begin{bmatrix} x_i \\ y_i \end{bmatrix} - \begin{bmatrix} x_j \\ y_j \end{bmatrix} \right\|_1 \geq 2r$$

che impongono che la distanza (in norma L_1) tra i due centri sia almeno pari a $2r$. Questi vincoli definiscono un insieme "fortemente non convesso"; infatti l'insieme sarebbe convesso (esprimibile utilizzando solamente vincoli lineari) se il verso del vincolo fosse opposto ($\leq 2r$), e pertanto la regione ammissibile che definisce è "tutto lo spazio tranne un insieme convesso". Utilizzando la definizione, questo vuol dire

$$| x_i - x_j | + | y_i - y_j | \geq 2r \ ,$$

e siccome $| z | = \max\{ z , -z \}$ il vincolo si riscrive

$$\max \left\{ x_i - x_j + y_i - y_j \, , \, x_i - x_j - y_i + y_j \, , \, -x_i + x_j + y_i - y_j \, , \, -x_i + x_j - y_i + y_j \right\} \geq 2r \ .$$

Affinché il *massimo* tra quattro funzioni lineari sia maggiore od uguale a $2r$, *almeno una* di esse deve essere tale. Si tratta quindi di applicare la tecnica dei vincoli disgiuntivi. Ciò richiede di introdurre, per ciascuna coppia $\{ i , j \} \in C$, quattro variabili binarie y_{ij}^{++}, y_{ij}^{+-}, y_{ij}^{-+} e y_{ij}^{--}, il che conferma la non-convessità dell'insieme da definire. Con queste, una formulazione *PLI* del problema è

$$\begin{aligned}
&\max r \\
&r \leq x_i \leq 1 - r && i = 1, \ldots n \\
&r \leq y_i \leq 1 - r && i = 1, \ldots n \\
&x_i - x_j + y_i - y_j \geq 2r - 3y_{ij}^{++} && \{ i , j \} \in C \\
&x_i - x_j - y_i + y_j \geq 2r - 3y_{ij}^{+-} && \{ i , j \} \in C \\
&- x_i + x_j + y_i - y_j \geq 2r - 3y_{ij}^{-+} && \{ i , j \} \in C \\
&- x_i + x_j - y_i + y_j \geq 2r - 3y_{ij}^{--} && \{ i , j \} \in C \\
&y_{ij}^{++} + y_{ij}^{+-} + y_{ij}^{++} + y_{ij}^{--} \leq 3 && \{ i , j \} \in C \\
&y_{ij}^{++} \in \{ 0 , 1 \} \, , \, y_{ij}^{+-} \in \{ 0 , 1 \} \, , \, y_{ij}^{++} \in \{ 0 , 1 \} \, , \, y_{ij}^{--} \in \{ 0 , 1 \} && \{ i , j \} \in C
\end{aligned}$$

La funzione obiettivo massimizza r. I successivi due vincoli assicurano che il centro di ogni sfera sia a distanza (in norma L_1, ma in questo caso la cosa è ininfluente: tutte le norme sono equivalenti al valore assoluto quando ristrette lungo una sola dimensione) almeno r dalla frontiera del quadrato unitario. Ciascuno dei quattro vincoli successivi impone che la corrispondente versione "linearizzata" del vincolo di distanza sia rispettata se la corrispondente

variabile binaria vale 0, mentre è ridondante (dati i primi due vincoli) se la corrispondente variabile binaria vale 1. Per la stima della costante M dei vincoli si noti che, poiché $x_i \in [\,0\,,\,1\,]$ ed $y_i \in [\,0\,,\,1\,]$ per ogni i, la norma L_1 della differenza tra due centri non può essere superiore a 2; in altri termini, il lato sinistro dei vincoli è ≥ -2. Siccome ovviamente $r \leq 1/2$ per ogni $n \geq 1$, "lato sinistro $-2r$" ≥ -3, da cui la scelta della costante. Si noti che la costante sarebbe migliorabile utilizzando qualche argomento teorico che permetta di ricavare un upper bound valido sul valore di r. Ad esempio, poiché una palla in norma L_1 di raggio r ha area $2r^2$, deve risultare $2nr^2 \leq 1 \Longrightarrow r \leq \sqrt{1/(2n)}$. L'ultimo vincolo impone che almeno una delle quattro variabili sia 0, e quindi che almeno uno dei quattro vincoli sia rispettato, e quindi che il massimo tra le quattro corrispondenti forme lineari sia $\geq 2r$, concludendo la formulazione.

1.2.11 Esempi ed esercizi di modellazione

Concludiamo questa parte dedicata alle tecniche di modellazione con alcuni esempi nel quale si utilizzano alcune delle tecniche precedentemente illustrate; l'uso delle tecniche potrà poi essere autonomamente sperimentato svolgendo gli esercizi proposti alla fine del paragrafo.

Esempio 1.23. Dislocazione ottima di impianti

La società informatica MilanNet ha deciso di aprire nel territorio pisano sino a n possibili uffici di assistenza ai suoi m clienti. Per questo ha selezionato n possibili siti in cui aprire gli uffici; per ogni sito $i \in I = \{1, \ldots, n\}$ si conosce il costo d_i di installazione, il numero massimo u_i di clienti che l'ufficio potrebbe assistere qualora fisse attivato, e per ogni cliente $j \in J = \{1, \ldots, m\}$ il costo c_{ij} derivante dalla gestione del cliente j presso il centro i (se aperto). Si vuole decidere in quali delle n località aprire gli uffici di assistenza e, per ciascuno di essi, l'insieme dei clienti assegnati, in modo tale che ogni cliente sia assegnato ad uno ed un solo ufficio di assistenza e che il costo complessivo (di installazione e gestione) sia minimo.

Per formulare tale problema occorre introdurre due insiemi di variabili binarie: le variabili y_i, $i \in I$, per rappresentare la scelta relativa agli uffici da aprire, e le variabili x_{ij}, $i \in I$, $j \in J$, per assegnare i clienti agli uffici. La funzione obiettivo, da minimizzare, che include sia i costi di gestione che quelli di installazione è

$$\textstyle\sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} + \sum_{i \in I} d_i y_i \ .$$

I vincoli di semiassegnamento garantiscono che ogni cliente sia assegnato ad uno ed un solo ufficio:

$$\textstyle\sum_{i \in I} x_{ij} = 1 \qquad j \in J \ .$$

Dobbiamo poi aggiungere sia i vincoli sul numero massimo di clienti per ufficio

$$\textstyle\sum_{j \in J} x_{ij} \leq u_i \qquad i \in I \ , \tag{1.37}$$

sia quelli che garantiscono che i clienti siano assegnati ad uffici di cui sia stata decisa la costruzione:

$$x_{ij} \leq y_i \qquad j \in J \quad , \quad i \in I \ . \tag{1.38}$$

Questi ultimi esprimono l'implicazione $x_{ij} > 0 \Longrightarrow y_j = 1$. Per evitare di usare mn vincoli, si può imporre che la somma delle x_{ij} per i fissato, cioè il numero di clienti assegnati al sito i, sia nulla quando $y_i = 0$ e possa assumere un valore non superiore a u_i quando $y_i = 1$, mediante i vincoli

$$\textstyle\sum_{j \in J} x_{ij} \leq u_i y_i \qquad i \in I \ ;$$

questi implicano sia (1.37) che (1.38). Il problema può quindi essere formulato come

$$\begin{array}{lll} \min & \textstyle\sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} + \sum_{i \in I} d_i y_i & \\ & \textstyle\sum_{i \in I} x_{ij} = 1 & j \in J \\ & \textstyle\sum_{j \in J} x_{ij} \leq u_i y_i & i \in I \\ & y_i \in \{\,0\,,\,1\,\} & i \in I \\ & x_{ij} \in \{\,0\,,\,1\,\} & i \in I \ , \ j \in J \end{array}$$

Esercizio 1.23. Formulare un problema di installazione ottima di al più 4 impianti con 11 clienti, dando anche i costi di installazione e di gestione e le capacità degli impianti.

Esempio 1.24. La battaglia di Ilio

Teucri ed Achei si battono furiosamente sotto le mura di Ilio. Atena, dea della saggezza ed alleata dei Greci, spinge il re Agamennone ad organizzare un contrattacco con i cocchi per fermare l'avanzata nemica. Il figlio di Atreo ha a disposizione n Eroi (E), m Cocchi (C) trainati da focose pariglie di cavalli e k Aurighi (A) per condurli. Ogni Eroe può uscire in battaglia a piedi, oppure su un Cocchio; in questo caso ha bisogno di un'Auriga che lo conduca, ma questo compito può anche essere svolto da un altro Eroe (che in questo caso non combatte). Per ogni tripla

(e, c, a) con $e \in E$, $c \in C$ e $a \in A \cup E$ ($e \neq a$), i vaticini della dea con lo sguardo scintillante forniscono le misure f_e ed $f_{e,c,a}$ della forza che avrebbe in battaglia l'Eroe "e" da solo, oppure montato sul Cocchio "c" ed accompagnato dall'Auriga "a". Alcune delle triple `eac` sono invise a Zeus, che interverrebbe immediatamente dando la vittoria ai discendenti di Teucro se anche una sola di esse si mostrasse in battaglia; la dea conosce bene le debolezze di suo padre, e segnala questo ponendo $f_{e,c,a} = -\infty$. Il re di Micene chiede aiuto all'astuto Ulisse per determinare come dispiegare gli eroi in battaglia in nodo da massimizzare la loro forza totale. Aiutate Odisseo a compiere il volere del Fato scrivendo il modello di *PLI* del corrispondente problema.

Definiamo T come l'insieme di tutte le triple (e, c, a) con $e \in E$, $c \in C$, $a \in A \cup E \setminus \{e\}$ tali che $f_{e,c,a} > -\infty$; definiamo per ogni $e \in E$, $a \in A$ e $c \in C$ gli insiemi

$$T(e) = \{(c, a) : (e, c, a) \in T\} \cup \{(e', c) : (e', c, e) \in T\}$$
$$T(a) = \{(e, c) : (e, c, a) \in T\} \qquad T(c) = \{(e, a) : (e, c, a) \in T\} .$$

Introduciamo inoltre per ogni $(e, c, a) \in T$ la variabile binaria

$$x_{e,c,a} = \begin{cases} 1 & \text{se l'Eroe "}e\text{" viene mandato in battaglia montato sul Cocchio "}c\text{" ed accompagnato dall'Auriga "}a\text{"} \\ 0 & \text{altrimenti} \end{cases}$$

e per ogni $e \in E$ la variabile binaria

$$x_e = \begin{cases} 1 & \text{se l'Eroe "}e\text{" viene mandato in battaglia a piedi} \\ 0 & \text{altrimenti} \end{cases} .$$

Una formulazione del problema è la seguente:

$$\begin{array}{lll} \max & \sum_{(e,c,a)\in T} f_{e,c,a}\, x_{e,c,a} + \sum_{e\in E} f_e x_e & \\ & \sum_{(c,a)\in T(e)} x_{e,c,a} + x_e \leq 1 & e \in E \\ & \sum_{(e,a)\in T(c)} x_{e,c,a} \leq 1 & c \in C \\ & \sum_{(e,c)\in T(a)} x_{e,c,a} \leq 1 & a \in A \\ & x_{e,c,a} \in \{0, 1\} & \texttt{eca} \in \texttt{T} \\ & x_e \in \{0, 1\} & e \in E \end{array}$$

Il primo blocco di vincoli garantisce che ciascun Eroe $e \in E$ scenda in battaglia al più una volta: a piedi, come combattente su di un Cocchio o come Auriga di un Cocchio (per ovvi motivi il vincolo potrebbe essere posto come uguaglianza). Il secondo e terzo blocco di vincoli garantiscono che ciascun Cocchio $c \in C$ e ciascun Auriga $a \in A$ scendano in battaglia al più una volta. I tre blocchi di vincoli insieme garantiscono che ciascuno dei possibili Eroi, Cocchi ed Aurighi sia coinvolto in una sola tripla, ferma restando la possibilità per gli Eroi di combattere da soli. Infine la funzione obiettivo, da massimizzare, rappresenta la forza totale degli Eroi nelle configurazioni prescelte.

Esempio 1.25. Produzione di androidi

La Società Cibernetica Sirio (TM) sta pianificando la costruzione del suo nuovo, grande stabilimento per la produzione di androidi con Personaltà di Persona Vera (TM). La produzione degli androidi è divisa in cinque reparti: montaggio (M), installazione software (I), test (T), lobotomizzazione preventiva (L), e distruzione rapida (D). Nello stabilimento sono presenti cinque aree, numerate da 1 a 5, ciascuna delle quali può ospitare uno qualunque dei reparti. Uno dei principali limiti alla produttività nella Società Cibernetica Sirio è la necessità di spostare gli androidi parzialmente prodotti dall'uno all'altro dei reparti. Per ogni coppia (i, j) di reparti, con $i \neq j$, si conosce il numero s_{ij} di androidi (nell'unità di tempo) che devono spostarsi da i a j. Per ogni coppia (h, k) di aree, con $h \neq k$, si conosce il tempo t_{hk} necessario a spostarsi da h a k. Si formuli come *PLI* il problema di decidere in quale area alloggiare ogni reparto in modo tale da minimizzare il tempo totale speso nel trasporto degli androidi.

Il problema si può formulare come Programmazione Quadratica in variabili 0/1; introducendo variabili $x_{ih} \in \{0, 1\}$ per ogni $i \in R = \{M, I, T, L, D\}$ e $h \in A = \{1, \ldots, 5\}$, col significato che $x_{ih} = 1$ se il reparto i viene assegnato all'area h, una formulazione del problema è

$$\begin{array}{lll} \min & \sum_{i\in R} \sum_{j\in R\setminus\{i\}} \sum_{h\in A} \sum_{k\in A\setminus\{h\}} s_{ij}\, t_{hk}\, x_{ih}\, x_{jk} & \\ & \sum_{i\in R} x_{ih} = 1 & h \in A \\ & \sum_{h\in A} x_{ih} = 1 & i \in R \\ & x_{ih} \in \{0, 1\} & i \in R,\ h \in A \end{array}$$

Infatti, nella funzione obiettivo vengono conteggiati tutti gli spostamenti tra una specifica coppia (i, j) di reparti con il tempo di trasferimento tra una specifica coppia (h, k) di aree se e solo se il reparto i è assegnato all'area h

ed il reparto j è assegnato all'area k, ossia $x_{ih} = x_{jk} = 1$. I vincoli del problema sono i noti vincoli di assegnamento; infatti, un problema in questa forma è chiamato problema di *assegnamento quadratico*.

Tale problema può essere facilmente portato in forma di *PLI* linearizzando i prodotti $x_{ih}x_{jk}$, che notoriamente corrispondono all'"and logico" tra le due variabili. Per questo occorre introdurre variabili binarie y_{ijhk} per $i \in R$, $j \in R \setminus \{ i \}$, $h \in A$ e $k \in A \setminus \{ h \}$, con il significato che $y_{ijhk} = x_{ih}\, x_{jk}$, e riformulare il problema come

$$
\begin{array}{lll}
\min & \sum_{i \in R} \sum_{j \in R \setminus \{ i \}} \sum_{h \in A} \sum_{k \in A \setminus \{ h \}} s_{ij}\, t_{hk}\, y_{ijhk} & \\
& \sum_{i \in R} x_{ih} = 1 & h \in A \\
& \sum_{h \in A} x_{ih} = 1 & i \in R \\
& y_{ijhk} \leq x_{ih} & i \in R\,,\ j \in R \setminus \{ i \}\,,\ h \in A\,,\ k \in A \setminus \{ h \} \\
& y_{ijhk} \leq x_{jk} & i \in R\,,\ j \in R \setminus \{ i \}\,,\ h \in A\,,\ k \in A \setminus \{ h \} \\
& y_{ijhk} \geq x_{ih} + x_{jk} - 1 & i \in R\,,\ j \in R \setminus \{ i \}\,,\ h \in A\,,\ k \in A \setminus \{ h \} \\
& x_{ih} \in \{ 0\,,\, 1 \} & i \in R\,,\ h \in A \\
& y_{ijhk} \in \{ 0\,,\, 1 \} & i \in R\,,\ j \in R \setminus \{ i \}\,,\ h \in A\,,\ k \in A \setminus \{ h \}
\end{array}
$$

I nuovi vincoli del problema garantiscono che $y_{ijhk} = x_{ih}\, x_{jk}$, e quindi permettono di trasformare la funzione obiettivo quadratica in una equivalente lineare.

Esempio 1.26. Programmazione di turni

AirExpress (AE) è una compagnia di trasporto express che garantisce la consegna notturna di pacchi negli Stati Uniti. Il manager dell'hub di AE, situato in Baltimora, deve gestire gli operatori addetti alla consegna dei pacchi. L'hub opera sette giorni alla settimana e, sulla base di stime relative alla domanda di pacchi da consegnare, il manager ha stimato nel modo seguente il numero minimo di operatori necessari in ogni giorno della settimana:

giorno	Domenica	Lunedì	Martedì	Mercoledì	Giovedì	Venerdì	Sabato
operatori	18	27	22	26	25	21	19

Gli operatori di AE lavorano con un contratto che garantisce turni di lavoro di 5 giorni a settimana, con due giorni consecutivi di riposo (i turni ammissibili sono quindi 7). Il salario settimanale per operatore è di 705 dollari se l'operatore non riposa in nessuno dei due giorni del weekend, di 680 dollari se riposa in uno solo dei due, e di 655 dollari se riposa in entrambi. Inoltre il manager di AE può chiamare ogni giorno un numero arbitrario di lavoratori interinali al costo di 125 dollari l'uno. Si aiuti il manager di AE proponendo due diversi modelli *PLI* che decidano quanti operatori di AE assegnare ad ogni turno e/o quanti interinali assegnare ad ogni giorno, in modo tale da:

1. (*primo modello*) minimizzare il salario totale da pagare garantendo il numero minimo di operatori richiesto per ogni giorno della settimana;
2. (*secondo modello*) avendo a disposizione un budget settimanale pari a 23000 dollari per il pagamento dei salari, ma non potendo utilizzare lavoratori interinali, ottenere un carico di lavoro quanto più possibile equo, minimizzando la somma tra il massimo scostamento in eccesso ed il massimo scostamento in difetto tra il numero di operatori assegnati in un giorno e la stima degli operatori necessari quel giorno, posto che il massimo scostamento in difetto deve comunque risultare inferiore a due.

Per descrivere il problema denotiamo i giorni Domenica, Lunedì, ..., Sabato mediante l'insieme di indici $S = \{ 1\,,\, 2\,,\, \ldots\,,\, 7 \}$. Introduciamo quindi, per ogni $j \in S$, la variabile intera x_j che indica il numero di operatori assegnati al turno j, inteso come il turno in cui il riposo inizia il giorno j (quindi il turno 1 è quello in cui il riposo è Domenica e Lunedì, il turno 2 è quello in cui il riposo è Lunedì e Martedì, e così via), e la variabile intera y_j che indica il numero di operatori interinali che vengono convocati il giorno j. Utilizzando tali variabili decisionali, il

primo dei due modelli è

$$
\begin{array}{rrrrrrrrrrrrrrr}
\min & 680x_1 & +705x_2 & +705x_3 & +705x_4 & +705x_5 & +680x_6 & +655x_7 & +125y_1 & +125y_2 & +125y_3 & +125y_4 & +125y_5 & +125y_6 & +125y_7 \\
 & & x_2 & +x_3 & +x_4 & +x_5 & +x_6 & & +y_1 & & & & & & \geq 18 \\
 & & & x_3 & +x_4 & +x_5 & +x_6 & +x_7 & & +y_2 & & & & & \geq 27 \\
 & x_1 & & & +x_4 & +x_5 & +x_6 & +x_7 & & & +y_3 & & & & \geq 22 \\
 & x_1 & +x_2 & & & +x_5 & +x_6 & +x_7 & & & & +y_4 & & & \geq 26 \\
 & x_1 & +x_2 & +x_3 & & & +x_6 & +x_7 & & & & & +y_5 & & \geq 25 \\
 & x_1 & +x_2 & +x_3 & +x_4 & & & +x_7 & & & & & & +y_6 & \geq 21 \\
 & x_1 & +x_2 & +x_3 & +x_4 & +x_5 & & & & & & & & & +y_7 \geq 19 \\
 & x_1, & x_2, & x_3, & x_4, & x_5, & x_6, & x_7, & y_1, & y_2, & y_3, & y_4, & y_5, & y_6, & y_7 \in N
\end{array}
$$

Il primo vincolo specifica infatti che il numero di operatori AE che, in base all'assegnamento degli operatori ai turni ed alle caratteristiche dei turni, risulteranno disponibili la Domenica più il numero di operatori interinali deve essere almeno pari a 18; seguono analoghi *vincoli di copertura* per gli altri giorni della settimana. La funzione obiettivo, da minimizzare, rappresenta il salario totale da pagare. Il secondo modello invece è

$$
\begin{array}{rrrrrrrrrrrrrr}
\min & z & & & & & & & & & & & & +w \\
 & & & 680x_1 & +705x_2 & +705x_3 & +705x_4 & +705x_5 & +680x_6 & +655x_7 & & \leq & 23000 \\
 & z & \geq & & x_2 & +x_3 & +x_4 & +x_5 & +x_6 & & -18 & \geq & -w \\
 & z & \geq & & & x_3 & +x_4 & +x_5 & +x_6 & +x_7 & -27 & \geq & -w \\
 & z & \geq & x_1 & & & +x_4 & +x_5 & +x_6 & +x_7 & -22 & \geq & -w \\
 & z & \geq & x_1 & +x_2 & & & +x_5 & +x_6 & +x_7 & -26 & \geq & -w \\
 & z & \geq & x_1 & +x_2 & +x_3 & & & +x_6 & +x_7 & -25 & \geq & -w \\
 & z & \geq & x_1 & +x_2 & +x_3 & +x_4 & & & +x_7 & -21 & \geq & -w \\
 & z & \geq & x_1 & +x_2 & +x_3 & +x_4 & +x_5 & & & -19 & \geq & -w \\
 & & & & & & & & & & 2 & \geq & w \\
 & z, & & x_1, & x_2, & x_3, & x_4, & x_5, & x_6, & x_7, & & & w \in N
\end{array}
$$

In questo caso, il primo è il *vincolo di budget* sui salari. Al modello sono state aggiunte due *variabili di soglia* z e w per stimare, rispettivamente, il massimo scostamento in eccesso ed il massimo scostamento in difetto tra il numero di operatori disponibili in un giorno e la stima degli operatori necessari quel giorno. I vincoli di copertura, mediante l'uso di tali variabili, diventano *vincoli di soglia*. Poiché la funzione obiettivo minimizza la somma tra z e w, all'ottimo z sarà il massimo scostamento in eccesso e w il massimo scostamento in difetto. L'ultimo vincolo garantisce che il massimo scostamento in difetto non superi mai il valore 2.

Esempio 1.27. Il Finale Verde di Mass Effect (warning: spoiler!)

Il Capitano Jane Shepard (CJS) sta per terminare la sua epica avventura, lunga ben tre RPG, non solo salvando la Galassia dai temibili Razziatori, ma cambiandone per sempre la storia interrompendo il terribile Ciclo della Mietitura con cui veniva garantita la "convivenza" tra organici e sintetici. Nel finale "verde", rinunciando alla sua identità e fondendo la sua energia vitale con quella della Cittadella scatenerà un'onda di energia che, raggiungendo tutti i sistemi stellari S della Galassia, fonderà il DNA degli organici con i circuiti dei sintetici, ottenendo una sintesi perfetta che garantirà un'epoca di pace e prosperità mai viste. Per questo però l'onda di energia deve attraversare la rete dei Portali, rappresentabile come un grafo non orientato $G = (V, E)$, partendo da quello Sol (la Cittadella è in orbita attorno alla Terra) $sol \in V$. Da ogni Portale $i \in V$ l'onda può arrivare a quelli raggiungibili direttamente attraverso i rispettivi collegamenti interspaziali $\{i, j\} \in E$, ed influenzare i sistemi stellari $S(i) \subset S$ ad esso vicini. Ogni Portale raggiunto dall'onda, però, verrà distrutto. Fedele fino in fondo al suo ruolo di Eroe, il CJS nei suoi ultimi millisecondi di vita pensa a rendere più semplice la lunga fase di ricostruzione dei Portali, la fondamentale infrastruttura su cui si basa tutto il trasporto della Galassia, che dovrà seguire. Per questo vuole determinare l'insieme dei Portali di cardinalità minima attraverso cui far passare l'onda (e che quindi verranno distrutti) in modo da assicurare la copertura di tutti i sistemi stellari S. Per rendere il più possibile indolore la ricostruzione, inoltre, il CJS vuole assicurarsi che per qualsiasi portale distrutto ne esista uno ancora attivo a distanza non superiore a D (sono note le distanze d_{ij} per ogni coppia $\{i, j\} \in V \times V$). Aiutate il CJS a compiere il suo destino scrivendo come *PLI* il problema che deve risolvere.

Per modellare il problema consideriamo il classico grafo orientato $G' = (V, A)$ con gli stessi nodi (vertici) di G e tale che A contiene i due archi (i, j) e (j, i) per ciascun lato $\{i, j\} \in E$. Per ciascun Portale $i \in V$ definiamo inoltre l'insieme $D(i) = \{ j \in V : d_{ij} \leq D \}$ dei Portali abbastanza vicini affinché la ricostruzione non sia troppo difficile. Sugli archi di G' introduciamo le classiche variabili di flusso x_{ij}. Per ciascun vertice (nodo) $i \in V$, tranne *sol* (che ovviamente viene distrutto), introduciamo la variabile y_i che indica se il Portale corrispondente viene raggiunto dall'onda. Con questo insieme di variabili, un modello *PLI* del problema del CJS è:

$$
\begin{array}{lll}
\min & \sum_{i \in V \setminus \{ sol \}} y_i & \\
& \sum_{(j,i) \in BS(i)} x_{ji} - \sum_{(i,j) \in FS(i)} x_{ij} = \left\{ \begin{array}{ll} -\sum_{j \in V \setminus \{ sol \}} y_j & i = sol \\ y_i & i \neq sol \end{array} \right. & i \in V \\
& x_{ij} \leq (|V| - 1) y_i & i \in V \setminus \{ sol \} \ , \ (i, j) \in FS(i) \\
& \sum_{i : s \in S(i)} y_i \geq 1 & s \in S \setminus S(sol) \\
& \sum_{j \in D(i)} (1 - y_j) \geq y_i & i \in V \setminus \{ sol \} \\
& \sum_{j \in D(sol)} (1 - y_j) \geq 1 & \\
& x_{ij} \geq 0 & (i, j) \in A \\
& y_i \in \{ 0, 1 \} & j \in V \setminus \{ sol \}
\end{array}
$$

La funzione obiettivo rappresenta il numero di Portali distrutti. Il secondo blocco di vincoli sono i classici vincoli di conservazione del flusso ce assicurano che l'onda di energia si propaghi da *sol*, utilizzando i collegamenti interspaziali esistenti, fino a raggiungere tutti i Portali $i \in V \setminus \{ sol \}$ tali che $y_i = 1$. Per far questo l'onda non può passare per nessuno dei Portali tali che $y_i = 0$ per via dei vincoli del terzo blocco di vincoli, che garantiscono che nessuna unità di flusso possa uscire da i (e quindi, per via della conservazione del flusso, nessuna possa entrarvi) se $y_i = 0$. Il quarto blocco di vincoli garantisce che l'onda di energia arrivi ad almeno uno dei Portali sufficientemente vicini a ogni sistema stellare affinché esso possa esserne influenzato; si noti che il vincolo non riguarda i sistemi stellari vicini al Portale *sol*, in quanto è certo che essi verranno raggiunti. Il quinto blocco di vincoli assicura che se il Portale $i \neq sol$ viene distrutto allora non lo sia almeno uno dei Portali ad esso abbastanza vicini per permettere una facile ricostruzione; il blocco successivo ne è la forma particolare per il Portale *sol*, che certamente viene distrutto. I rimanenti sono i normali vincoli di segno (bounds), ed ove necessario di integralità, delle variabili.

Esempio 1.28. Impaccamento di rettangoli

Sono dati n rettangoli, di altezza e larghezza rispettivamente h_i e w_i per $i = 1, \ldots, n$. I rettangoli devono essere tutti disposti con i lati paralleli (senza rotazioni, nemmeno di $\pi/2$) all'interno del quadrato di lato (e quindi area) più piccolo possibile, rispettando le seguenti condizioni: (i) due rettangoli diversi non possono avere intersezione, a parte eventualmente sulla frontiera; (ii) per $i < j$, il rettangolo j deve essere disposto "più in alto e più a destra" del rettangolo i, ossia l'angolo in basso a sinistra di j deve avere sia la coordinata x che la coordinata y non inferiori a quelle dell'angolo in basso a sinistra di i (considerando gli assi paralleli ai lati, ovviamente). Si scriva il problema corrispondente come *PLI*.

Per modellare il problema introduciamo variabili x_i e y_i che indicano le coordinate dell'angolo in basso a sinistra del i-esimo rettangolo nella disposizione scelta. Poichè il problema è invariante per traslazione possiamo sempre disporre l'origine in basso a sinistra di tutti i rettangoli; per il vincolo sull'ordinamento questo corrisponde ad $x_1 \geq 0$ e $y_1 \geq 0$, poichè $x_i \geq x_1$ e $y_i \geq y_1$ per $i = 1, \ldots, n$.

Sia adesso (i, j) una coppia ordinata di (indici di) rettangoli con $i < j$; è necessario esprimere le condizioni (i) e (ii). La (i) è del tutto ovvia, e corrisponde a $x_j \geq x_i$, $y_j \geq y_i$. Per quanto riguarda la (ii), è immediato verificare che ciò corrisponde al fatto che *almeno una* delle due condizioni

$$(1) \ x_i + w_i \leq x_j \qquad , \qquad (2) \ y_i + h_i \leq y_j$$

debba essere verificata. Ciò può essere espresso mediante la tecnica standard dei *vincoli disgiuntivi*, che prevede di utilizzare una variabile binaria per ogni vincolo; definiamo quindi le variabili $z_{ij}^{(h)}$ per $h = 1, 2$ e $(i, j) \in C$, dove C è l'insieme di tutte le coppie non ordinate di (indici di) rettangoli con $i < j$. Per scrivere i vincoli disgiuntivi abbiamo bisogno di costanti "sufficientemente grandi" tali da renderli inattivi; per questo possiamo notare che si può banalmente trovare una disposizione ammissibile tale per cui $x_i \leq W$ e $y_i \leq H$ con $W = \sum_{i=1}^{n} w_i$ e $H = \sum_{i=1}^{n} h_i$.

Una formulazione del problema è:

$$
\begin{array}{ll}
\min & v \\
& 0 \le x_1 \ , \ 0 \le y_1 \\
& x_{i+1} \ge x_i \ , \ y_{i+1} \ge y_i \qquad i = 1, \ldots, n-1 \\
& x_i + w_i \le v \ , \ y_i + h_i \le v \qquad i = 1, \ldots, n \\
& x_i + w_i \le x_j + W z_{ij}^{(1)} \qquad (i\,,\,j) \in C \\
& y_i + h_i \le y_j + H z_{ij}^{(2)} \qquad (i\,,\,j) \in C \\
& z_{ij}^{(1)} + z_{ij}^{(2)} \le 1 \qquad (i\,,\,j) \in C \\
& z_{ij}^{(h)} \in \{\,0\,,\,1\,\} \qquad h = 1,2 \ , \ (i\,,\,j) \in C
\end{array}
$$

I primi due blocchi di vincoli assicurano la non-negatività delle coordinate (invarianza per traslazione) e la condizione (i) (ordinamento). Il terzo blocco di vincoli assicura che v sia maggiore od uguale della massima coordinata, sia x che y, dell'angolo superiore destro di ciascun rettangolo; pertanto, tutti i rettangoli appartengono al quadrato di lato v avente angolo inferiore sinistro fissato nell'origine. Poiché la funzione obiettivo minimizza v, si determinerà la disposizione che minimizza il lato (e quindi l'area) del quadrato. I successivi vincoli rappresentano le condizioni disgiuntive, modificate per fare in modo che la condizione (h) sia imposta se $z_{ij}^{(h)} = 0$, mentre possa essere violata se $z_{ij}^{(h)} = 1$; il blocco di vincoli finale (esclusi quelli di integralità e bounds sulle variabili) quindi assicurano che almeno una delle due condizioni sia soddisfatta (al massimo una su due sia violata).

Esempio 1.29. Arche spaziali Golgafrinchiane

Il pianeta Golgafrincham è spacciato, anche se sul perché sono diffuse voci contrastanti (colliderà col suo sole, la sua luna cadrà, sarà invaso da un gigantesco sciame di api piranha alte quattro metri, sarà divorato da una enorme capra stellare mutante, ...). Per salvarsi, i Golgafrinchiani stanno costruendo tre Arche capaci di trasportare milioni di persone per colonizzare altri pianeti. Hanno anche mappato il settore galattico locale come un grafo orientato $G = (N\,,\,A)$, in cui $g \in N$ rappresenta Golgafrincham, e gli archi $(i\,,\,j) \in A$ rappresentano i viaggi spaziali che possono essere compiuti con la tecnologia a disposizione. Devono adesso decidere su quali pianeti in $N' = N \setminus \{\,g\,\}$ inviare le Arche. Ciascuna di esse arriverà sicuramente alla propria destinazione e la colonizzerà, ma data l'importanza dell'impresa i Golgafrinchiani non intendono fermarsi a soli tre pianeti. Per questo prepareranno anche k sotto-Arche; ciascuna delle Arche può portarne con sé al massimo $k/3 \le h < k$, ed una volta arrivata al pianeta destinazione inviarle ciascuna a colonizzare un diverso altro pianeta. Data la ridotta dimensione, una sotto-Arca inviata dal pianeta i al pianeta j ha solamente probabilità $P_{ij} \in [\,0\,,\,1\,]$ di arrivarci. È peró possibile inviare più sotto-Arche da pianeti diversi allo stesso pianeta, in modo da aumentare la probabilità di fondarci una colonia; le probabilità di successo sono indipendenti. Si scriva come *PLI* il problema di determinare a quali pianeti mandare le Arche e le sotto-Arche in modo da massimizzare il numero di colonie fondate con probabilità almeno pari ad $\alpha \in [\,0\,,\,1\,]$.

Per costruire il modello introduciamo le variabili binarie x_i per $i \in N'$ che indicano se una delle Arche è inviata al pianeta i, e le variabili binarie y_{ij} per $[\,0\,,\,1\,] \in A$ (tale che $i \neq g$) che indicano se una sotto-Arca viene inviata dal pianeta i al pianeta j. Il punto delicato del modello è esprimere la probabilità p_j di stabilire una colonia sul pianeta j. Questa è comunque sufficiente ($\ge \alpha$) se j è la destinazione di un'Arca; altrimenti, possiamo scrivere che

$$1 - p_j = \textstyle\prod_{i \in N' \setminus \{\,j\,\}} (\,1 - y_{ij} P_{ij}\,) \ .$$

Infatti, l'evento "la colonia *non* viene stabilita" è la congiunzione degli eventi indipendenti "la sotto-Arca proveniente da i non raggiunge j" per ogni $i \in N' \setminus \{\,j\,\}$. A loro volta, la probabilità di ciascuno di tali eventi è 1 se la sotto-Arca non viene inviata, e $1 - P_{ij}$ se invece viene inviata. Il vincolo $1 - p_j \le 1 - \alpha$ è nonlineare (espandendo la definizione di p_j), ma può essere reso lineare prendendo il logaritmo da entrambi i lati. Introducendo ulteriori variabili z_i per $i \in N'$ che indicano se una colonia viene stabilita con sufficiente probabilità sul pianeta i, una formulazione *PLI*

del problema è:

$$
\begin{array}{ll}
\max & \sum_{i\in N'} z_i \\
& \sum_{i\in N'} x_i = 3 \\
& \sum_{(\,i\,,\,j\,)\in A} y_{ij} \le h x_i \qquad\qquad i \in N' \\
& \sum_{i\in N'} \sum_{(\,i\,,\,j\,)\in A} y_{ij} \le k \\
& \log(\,1-\alpha\,) z_j \ge \log(\,1-\alpha\,) x_j + \sum_{i\in N'\setminus\{\,j\,\}} y_{ij} \log(\,1-P_{ij}\,) \quad j \in N' \\
& x_i \in \{\,0\,,\,1\,\}\ ,\ \ z_i \in \{\,0\,,\,1\,\} \qquad\qquad i \in N' \\
& y_{ij} \in \{\,0\,,\,1\,\} \qquad\qquad i \in N' \setminus \{\,g\,\}\ ,\ (\,i\,,\,j\,) \in A
\end{array}
$$

La funzione obiettivo massimizza il numero di variabili z_i poste ad 1, ossia il numero di colonie fondate con sufficiente probabilità; il blocco di vincoli critico è l'ultimo, discusso in dettaglio più sotto, che assicura che queste siano almeno quelle nelle quali vengono mandate o le Arche, che sono esattamente tre per via del primo vincolo, o un numero opportuno di sotto-arche. Il secondo vincolo assicura che ogni arca invii al massimo h sotto-Arche, mentre il terzo assicura che il numero totale di sotto-Arche inviate sia al massimo pari a k; quest'ultimo avrebbe potuto essere scritto come uguaglianza, dato che non cè nessun motivo per non spedire sotto-Arche. Il vincolo critico è ottenuto a partire dalla condizione logica

$$\log(\,1-p_j\,) \le \log(\,1-\alpha\,) \qquad\qquad (*)\ .$$

Si noti innanzi tutto che, data la natura binaria di y_{ij}, le espressioni e "$\log(\,1-y_{ij}P_{ij}\,)$" e "$y_{ij}\log(\,1-P_{ij}\,)$" sono equivalenti (ma la seconda è lineare in y_{ij}). Inoltre, tutte le probabilità sono < 1 (e > 0), e quindi i logaritmi sono < 0 (ma ben definiti). Se $x_i = 1$, è chiaramente possibile porre $z_i = 1$ (il resto del lato destro del vincolo è non-positivo). Se invece $x_i = 0$, ponendo $z_i = 1$ il lato sinistro del vincolo diviene negativo; ciò costringe, per rispettarlo, a porre ad 1 variabili y_{ij} (che hanno coefficiente negativo) affinché $(*)$ sia rispettato.

Come tutti sanno, il modello è solamente parte di un elaborato inganno volto a caricare tutti i membri inutili della società quali opinionisti, parrucchieri e sterilizzatori di telefoni sull'Arca B e liberarsi per sempre di loro, permettendo alla civiltà Golgafrinchiana di vivere una nuova età dell'oro fino al momento in cui non sarà spazzata via da un'epidemia causata da un telefono non sterilizzato.

1.2.11.1 Esercizi di modellazione

Esercizio 1.24. La Fintus produce tre tipi di patatine surgelate, denominati A, B e C. La compagnia acquista patate di due tipi diversi, denominati P_1 e P_2. I diversi tipi di prodotto usano parti diverse della patata originaria, per cui 1Kg di patate acquistato determina la produzione di una certa quantità di tutti e tre i prodotti. I rendimenti dei due tipi di patata sono diversi, come indicato nella seguente tabella:

patata/tipo	A	B	C
P_1	.2	.2	.3
P_2	.3	.1	.3

Il profitto della Fintus è di .03€ al Kg per le patate P_1 e di .025€ al Kg per le patate P_1: la Fintus intende produrre non più di 6000 Kg di A, 4000 Kg di B e 8000 kg di C, massimizzando il profitto. Formulare come *PL* il problema di ottimizzazione corrispondente.

Esercizio 1.25. L'azienda Caramelli produce un olio speciale per cosmetici, ottenuto dalla raffinazione e miscelazione di oli. Gli oli si dividono in due categorie, oli vegetali ed oli non vegetali. Sono disponibili due diversi oli vegetali, che indichiamo con V_1 e V_2, e tre diversi oli non vegetali che indichiamo con N_1, N_2 e N_3. I costi (€/tonnellata) e la densità degli oli sono i seguenti:

	V_1	V_2	N_1	N_2	N_3
Costo	110	120	130	110	115
Densità	8.8	6.1	2.0	4.2	5.0

Gli oli vegetali e quelli non vegetali richiedono differenti linee di produzione per la raffinazione. In ogni mese non è possibile raffinare più di 200 tonnellate di olio vegetale e 250 tonnellate di olio non vegetale. Non vi è perdita di peso nel processo di raffinamento, ed il costo di tale processo può essere ignorato. Vi è invece una restrizione tecnologica sulla densità del prodotto finale: nell'unità di misura

opportuna, questa deve essere compresa tra 3 e 6. Si assume che la densità degli oli si misceli nel prodotto finale in modo lineare. Il prodotto finale sarà venduto a 150€/tonnellata. Formulare come *PL* il problema di produrre il bene massimizzando il profitto.

Esercizio 1.26. Un'industria dolciaria produce 3 diversi tipi di dolci: A, B, C. Si deve stabilire il piano di produzione giornaliero dell'industria, avente una capacità produttiva massima di 10000 dolci al giorno, in modo che la produzione di A non ecceda il 50% della produzione globale giornaliera, e che la produzione di C sia uguale al più al 25% della produzione di B. Sapendo che il guadagno garantito dalla produzione di un dolce di tipo A, B e C è rispettivamente di 0.2€, 0.1€ e 0.4€, si formuli il problema di individuare un piano di produzione che massimizzi il guadagno come *PLI*.

Esercizio 1.27. Un'impresa ha a disposizione tre procedimenti differenti, chiamati P_1, P_2 e P_3, per la produzione di un certo bene. Per produrre una unità di tale bene sono necessarie lavorazioni tre macchine diverse, chiamate A, B e C. I tempo di lavorazione per ogni macchina necessario a produrre un'unità di bene dipende dal procedimento usato, come mostrato nella tabella seguente:

procedimento/macchina	A	B	C
P_1	2	4	3
P_2	1	2	4
P_3	3	3	2

Ogni macchina è disponibile per 50 unità di tempo. Il profitto per la vendita di un'unità di bene dipende dal procedimento usato: è 15 se si è usato il procedimento P_1, 18 se si è usato P_2 e 10 se si è usato P_3 (in €). Formulare come *PL* il problema di minimizzare il numero di unità di tempo di impiego della macchina B, con il vincolo che il profitto sia almeno pari a 200.

Esercizio 1.28. Il direttore amministrativo dell'ospedale Santa Cara deve stabilire i turni ospedalieri delle ostetriche, in modo da garantire un minimo numero di ostetriche presenti in ogni turno (indicato nella tabella). Il direttore vuole utilizzare il minor numero totale di ostetriche, tenendo conto che le ostetriche che si recano in reparto per uno dei primi cinque turni sono obbligate a lavorare per 8 ore consecutive (due turni consecutivi), mentre quelle impiegate nell'ultimo turno (turno 6) lavorano solo 4 ore. Si formuli il problema come *PLI*.

Turno	1	2	3	4	5	6
Orario	6 - 10	10 - 14	14 - 18	18 - 22	22 - 2	2 - 6
N. ostetriche	70	80	50	60	40	30

Esercizio 1.29. Sia data la matrice 3×3 di numeri reali in figura, in cui sono anche indicate le somme degli elementi di ogni riga e di ogni colonna. Si vuole arrotondare ogni elemento della matrice o alla sua parte intera inferiore, oppure alla sua parte intera superiore. Lo stesso procedimento di arrotondamento deve essere applicato alla somma degli elementi di ogni riga ed alla somma degli elementi di ogni colonna. Si vogliono eseguire tali operazioni di arrotondamento in modo che, nel problema trasformato, la somma degli elementi arrotondati in ogni riga (colonna) sia uguale alla rispettiva somma di riga (colonna) arrotondata. Si formuli il problema come *PLI*.

3.1	6.8	7.3	17.2
9.6	2.4	0.7	12.7
3.6	1.2	6.5	11.3
16.3	10.4	14.5	somme

Esercizio 1.30. Un villaggio ha 7 abitanti $\{ a_1 , \ldots , a_7 \}$, 4 clubs politici $\{ C_1 , \ldots , C_4 \}$, e 3 partiti politici $\{ P_1 , \ldots , P_3 \}$. Ogni abitante è membro di almeno un club, ed è iscritto ad un solo partito politico. Più precisamente, i clubs hanno i seguenti membri: $C_1 = \{ a_1 , a_2 \}$; $C_2 = \{ a_2 , a_3 , a_4 \}$; $C_3 = \{ a_4 , a_5 \}$; $C_4 = \{ a_4 , a_5 , a_6 , a_7 \}$, mentre i partiti hanno i seguenti iscritti: $P_1 = \{ a_1 , a_2 \}$; $P_2 = \{ a_3 , a_4 \}$; $P_3 = \{ a_5 , a_6 , a_7 \}$. Ogni club deve nominare uno dei suoi membri come rappresentante

al consiglio del villaggio, costituito da 4 membri. Formulare il problema di decidere se sia possibile costituire un consiglio del villaggio con la proprietà che al più un membro appartenga a P_2 e al più un membro appartenga a P_3.

Esercizio 1.31. Il C.T. Tramattoni, dopo la clamorosa esclusione dell'Itaglia dalla Coppa del Tondo, decide di rivolgersi ad un esperto di Ricerca Operativa per le prossime partite per i Campionati della Moneta Unica. Per le partite di qualificazione ha già deciso di avvalersi di una rosa di n giocatori; di ciascun giocatore i, $i = 1, \ldots, n$, si conosce la bravura b_i e il ruolo che può ricoprire in campo (uno e uno solo per giocatore). Gli n giocatori sono partizionati negli insiemi P dei portieri, D dei difensori, C dei centrocampisti e A degli attaccanti. Si dovrà far giocare almeno un giocatore per ciascun ruolo ma non più di 1 portiere, 6 difensori, 5 centrocampisti e 4 attaccanti. Tramattoni fornisce all'esperto anche la lista L, $|\,L\,| = m$, delle coppie di giocatori che non potranno giocare assieme per incompatibilità tattiche e/o caratteriali. Egli richiede all'esperto di aiutarlo a definire una formazione di 11 giocatori che rispetti sia le limitazioni sui ruoli sia le incompatibilità e che massimizzi la bravura complessiva, data dalla somma delle bravure dei singoli giocatori. Svolgere il ruolo dell'esperto di Ricerca Operativa formulando come *PLI* il problema per Tramattoni.

Esercizio 1.32. La commissione arbitri di calcio ha deciso di formare le terne (un arbitro più due guardialinee) in modo automatico per eliminare sospetti di "combine". Inoltre, per rispettare la legge sulle pari opportunità, la commissione ha deciso che ogni coppia di guardialinee sia formata da un uomo e una donna. Per le n partite in programma domenica prossima sono a disposizione a arbitri (maschi e femmine), m guardialinee maschi e f guardialinee femmine (con $a > n$, $m > n$, $f > n$). Il valore p_i indica la qualità (esperienza, capacità, ...) dell'arbitro o del/della guardialinee i; il valore di una terna è la somma dei valori delle tre persone che formano la terna stessa. Per evitare che si formino terne troppo difformi tra loro, la commissione decide di formare n terne in modo che sia minima la differenza tra il massimo e il minimo valore delle terne. Formulare il problema della commissione come problema di *PLI*.

Esercizio 1.33. Il partito del Limone si sta attivando per le elezioni europee e, per formare la lista dei candidati, ha a disposizione n volontari. Dopo un rapido sondaggio tra i limoncini, si valuta che il volontario i può ricevere c_i preferenze, $i = 1, \ldots, n$; inoltre il Capolimone conosce l'insieme $D(i)$ dei volontari in grado di collaborare col volontario i, $i = 1, \ldots, n$. Dall'insieme dei volontari si vuole selezionare una lista L di candidati tale che ognuno sia in grado di collaborare con gli altri e che la somma delle preferenze ricevute sia massima. Formulare il problema in termini di *PLI*.

Esercizio 1.34. Giro e Tond, i due maggiori produttori di automobili europei, hanno deciso di effettuare una fusione. Ciò comporta una gestione comune degli impianti delle due aziende, che dovranno produrre gli stessi modelli. Indichiamo con I e J rispettivamente gli insiemi degli impianti della Giro e della Tond, con K l'insieme dei mercati in cui dovrà operare la nuova azienda, la GiroTond, con b_k, $k \in K$, la domanda del k-esimo mercato e con c_{ik}, $i \in I \cup J$, $k \in K$, il costo unitario di trasporto dall'impianto i al mercato k. Si vuole assegnare ogni mercato ad uno ed uno solo degli impianti, chiudendo gli impianti in eccesso. Formulare, come problema di *PLI*, il problema dell'assegnamento degli impianti ai mercati, con l'obiettivo di minimizzare i costi di trasporto ed il vincolo che almeno il 50% degli impianti di ciascuno dei due produttori rimanga aperto.

Riferimenti Bibliografici

E. Specht "The best known packings of equal circles in a square (up to N = 10000)"
http://hydra.nat.uni-magdeburg.de/packing/csq/csq.html

J. Lee "A First Course in Linear Optimization v4.06" 2022
https://github.com/jon77lee/JLee_LinearOptimizationBook/blob/master/JLee.4.06.zip

F. Schoen "Optimization Models" free e-book version, 2022
https://webgol.dinfo.unifi.it/OptimizationModels/contents.html

M. Pappalardo, M. Passacantando “Ricerca Operativa” Edizioni Plus, 2013

Capitolo 2

Grafi e reti di flusso

Molti problemi di ottimizzazione sono caratterizzati da una struttura di grafo: in molti casi questa struttura emerge in modo naturale, in altri nasce dal particolare modo in cui i problemi vengono modellati. Ad esempio:

- una rete stradale o ferroviaria è naturalmente rappresentabile come un grafo in cui i nodi sono gli incroci/stazioni e gli archi le strade/ferrovie;
- in una rete di telecomunicazione i nodi sono tipicamente i *routers* mentre gli archi sono i collegamenti (wireless, in fibra ottica, ...) tra di essi;
- in una rete idrica o del gas gli archi sono le condutture ed i nodi sono i raccordi (eventualmente dotati di valvole e compressori) tra di esse;
- nella rete elettrica gli archi sono gli elettrodotti (a corrente continua o alternata) ed i nodi i trasformatori ed i collegamenti tra di essi.

Non è pertanto strano che i settori dei trasporti, delle telecomunicazioni e delle reti infrastrutturali ed energetiche (compresa quella elettrica) siano quelli in cui la teoria dei grafi trovi maggiore applicazione. In molti altri problemi, invece, la struttura di grafo è più nascosta. Abbiamo già visto diversi esempi di problemi con struttura di grafo (più o meno esplicita) nel Capitolo 1, ed altri ne vedremo nel seguito delle dispense.

In questo capitolo studieremo in particolare alcuni problemi di base definiti su grafi e reti, di cui forniremo le proprietà più importanti e per i quali introdurremo alcuni algoritmi risolutivi, in generale i più semplici; per maggiori approfondimenti, si rinvia alla letteratura indicata ed a corsi avanzati dell'area di Ricerca Operativa. La caratteristica principale di tutti i problemi che tratteremo in questo capitolo è di essere "facili", ossia di ammettere algoritmi risolutivi di complessità polinomiale (e molto efficienti in pratica). La conoscenza di algoritmi di base su grafi è un elemento importante nel curriculum di un esperto di informatica, anche al di fuori della Ricerca Operativa, in quanto strutture di grafo si incontrano sovente nel progetto e nella gestione di sistemi informatici, sia hardware che software: si pensi ad esempio alle reti di comunicazione (dalle WAN alle reti di interconnessione di calcolatori paralleli fino alle interconnessioni on-chip tra i molteplici elementi delle moderne CPU e GPU), ai compilatori, ai database, alla bioinformatica, all'apprendimento automatico/intelligenza artificiale, ecc.. Inoltre, molte delle idee algoritmiche che vedremo per questi primi, (relativamente) semplici problemi si ripresenteranno, in forma più sofisticata, per i problemi più complessi quali quelli che studieremo nel Capitolo 3. Infine, gli algoritmi efficienti per problemi "facili" sono comunque la base dei migliori (il meno inefficienti possibile) algoritmi per i problemi "difficili", che discuteremo Capitoli 5, 6 e 7.

2.1 Flussi su reti

In questo capitolo daremo per noti i concetti elementari di teoria dei grafi riassunti nell'Appendice B, facendo specifici rinvii a tale appendice solo quando necessario. Introdurremo ora alcuni concetti

tipici dei problemi di ottimizzazione su grafo, estendendo quanto visto nel §1.2.6.

Con il termine "rete" indichiamo un grafo *orientato* $G = (N, A)$ pesato, cioè ai cui nodi ed archi sono associati valori numerici; nel seguito indicheremo con $n = |N|$ il numero di nodi e con $m = |A|$ il numero di archi. In generale, in una rete gli archi sono interpretabili come canali attraverso cui fluiscono dei beni, che possono essere rappresentati per mezzo di grandezze discrete (ad esempio il numero di auto su una strada, o il numero di messaggi su una rete di comunicazione) o continue (quantità di petrolio che fluisce in un oleodotto) che possono rappresentare dei valori assoluti oppure dei valori relativi (per unità di tempo). In questo contesto, i pesi degli archi rappresentano usualmente delle capacità e dei costi, mentre i pesi dei nodi rappresentano la quantità dei beni che entrano nella rete in quei nodi, o che ne escono. Più precisamente, nei problemi di cui tratteremo:

- ad ogni nodo $i \in N$ è associato un valore reale b_i, detto *deficit* del nodo, che può essere:
  - positivo, e in tal caso rappresenta la quantità del bene che esce dalla rete al nodo i; b_i è allora detto *domanda del nodo*, ed il nodo viene detto *destinazione*, *pozzo* o *nodo di output*;
  - negativo, e in tal caso rappresenta la quantità di bene che entra nella rete al nodo i; $-b_i$ è allora detto *offerta del nodo*, ed il nodo viene detto *origine*, *sorgente* o *nodo di input*;
  - nullo, ed in questo caso i viene detto *nodo di trasferimento*;
- ad ogni arco $a_k = (i, j)$ sono associati un *costo* c_k (o c_{ij}), che indica il costo che viene pagato per ogni unità del bene che attraversi l'arco, ed una *capacità inferiore* l_k (l_{ij}) e *superiore* u_k (u_{ij}), che indicano, rispettivamente, il minimo ed il massimo numero di unità di bene che possono attraversare l'arco. In molte applicazioni la capacità inferiore viene assunta uguale a 0, e quindi viene fornita tra i parametri della rete solamente la capacità superiore.

Nei problemi di flusso la domanda globale, cioè la somma di tutte le domande, è uguale all'offerta globale, cioè alla somma, cambiata di segno, di tutte le offerte; più formalmente, detti D e O rispettivamente gli insiemi dei nodi di domanda e di offerta

$$D = \{ i \in N : b_i > 0 \} \quad \text{e} \quad O = \{ i \in N : b_i < 0 \} \quad \text{si ha} \quad \textstyle\sum_{i \in D} b_i = -\sum_{i \in O} b_i \ . \tag{2.1}$$

In altre parole, il vettore b, detto vettore dei *bilanci* dei nodi, deve soddisfare la relazione $\sum_{i \in N} b_i = 0$. Questa, come vedremo, è una condizione necessaria (ma non sufficiente) affinché esista un vettore $x = [x_{ij}]_{(i,j) \in A} \in \mathbb{R}^m$ che soddisfa i *vincoli di conservazione del flusso* (cf. §1.2.6)

$$\textstyle\sum_{(j,i) \in BS(i)} x_{ji} - \sum_{(i,j) \in FS(i)} x_{ij} = b_i \qquad i \in N \ , \tag{2.2}$$

dove $BS(i)$ e $FS(i)$ sono rispettivamente la stella entrante e la stella uscente di $i \in N$ (si veda l'Appendice B); un siffatto x si dice un *flusso* su G, ed il valore x_k (o x_{ij}) è detto *flusso dell'arco* $a_k = (i, j)$. Un flusso è poi detto *ammissibile* se sono verificati i *vincoli di capacità sugli archi*

$$l_{ij} \le x_{ij} \le u_{ij} \qquad (i, j) \in A \ . \tag{2.3}$$

Associando adesso ad ogni flusso x un costo dato dalla somma dei flussi degli archi per il loro costo

$$cx = \textstyle\sum_{(i,j)) \in A} c_{ij} x_{ij}$$

possiamo definire il *problema del flusso di costo minimo* (MCF, da *Min Cost Flow* problem) come

$$\text{(MCF)} \qquad \min \left\{ cx \ : \ Ex = b \, , \ l \le x \le u \right\} \tag{2.4}$$

dove $Ex = b$ rappresenta in forma vettoriale i vincoli di conservazione del flusso (2.2) utilizzando la *matrice di incidenza* del grafo G (si veda il §B.2.1) e $b = [b_i]_{i \in N}$, mentre $l \le x \le u$ rappresenta in forma vettoriale i vincoli di capacità (2.3) dove $l = [l_{ij}]_{(i,j) \in A}$ e $u = [u_{ij}]_{(i,j) \in A}$.

Esempio 2.1. Un'istanza di (MCF)
Sia dato il grafo orientato in Figura 2.1, in cui sono riportati domande, offerte, costi e capacità superiori (si suppone che le capacità inferiori siano nulle).

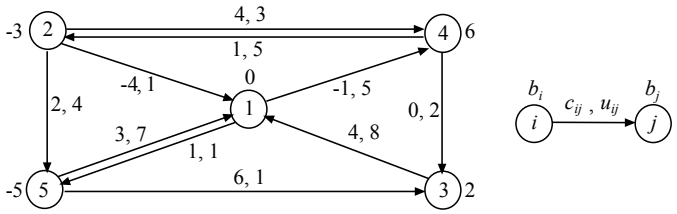


Figura 2.1: Un grafo con domande, offerte, costi e capacità

I vincoli di conservazione del flusso dei 5 nodi sono i seguenti:

$$\begin{array}{rrrrrrrrrrcr}
-x_{14} & -x_{15} & +x_{21} & & & +x_{31} & & & +x_{51} & & = & 0 \\
 & & -x_{21} & -x_{24} & -x_{25} & & +x_{42} & & & & = & -3 \\
 & & & & & -x_{31} & & +x_{43} & & +x_{53} & = & 2 \\
+x_{14} & & & +x_{24} & & & -x_{42} & -x_{43} & & & = & 6 \\
 & +x_{15} & & & +x_{25} & & & & -x_{51} & -x_{53} & = & -5
\end{array}$$

Si noti la forma della matrice di incidenza E. I vincoli di capacità degli archi sono

$$\begin{array}{lllll}
0 \le x_{14} \le 5 & 0 \le x_{15} \le 1 & 0 \le x_{21} \le 1 & 0 \le x_{24} \le 3 & 0 \le x_{25} \le 4 \\
0 \le x_{31} \le 8 & 0 \le x_{42} \le 5 & 0 \le x_{43} \le 2 & 0 \le x_{51} \le 7 & 0 \le x_{53} \le 1
\end{array}$$

mentre il costo del flusso è

$$cx = -x_{14} + x_{15} - 4x_{21} + 4x_{24} + 2x_{25} + 4x_{31} + x_{42} + 3x_{51} + 6x_{53} \ .$$

Esercizio 2.1. Determinare un flusso ammissibile per la rete in Figura 2.1 e valutarne il costo.

2.1.1 Alcuni modelli di flusso

Esiste un gran numero di problemi reali, in ambiti molto vari, che si modellano efficacemente come problemi di flusso: ne riportiamo di seguito alcuni esempi, caratterizzati dal fatto che il grafo sul quale il flusso è definito non è immediatamente evidente dalla descrizione del problema.

Esempio 2.2. Schedulazione della produzione

L'industria dolciaria PereCani, nota produttrice di panettoni, deve decidere come utilizzare al meglio, nel corso dell'anno, la sua linea di produzione per tali dolci. La PereCani conosce una stima b_i, $i = 1, \ldots, 12$, del numero di panettoni che venderà in ogni mese dell'anno. Il costo unitario di produzione e la massima quantità di panettoni producibile in un mese variano anch'esse, a seconda di alcuni fattori quali il prezzo delle materie prime e la disponibilità di personale impegnato anche su altre linee di produzione, e anche di esse si hanno le stime, rispettivamente, c_i ed u_i, $i = 1, \ldots, 12$. I panettoni prodotti al mese i possono essere venduti immediatamente, oppure immagazzinati per essere poi venduti nei mesi successivi: il magazzino ha una capacità massima di U, ed un costo unitario di immagazzinamento pari a C. All'inzio il magazzino contiene b_0 panettoni, e si desidera che alla fine dell'anno ne contenga b_{13}.

Il problema della PereCani, noto in letteratura come problema di *Lot Sizing*, può essere formulato come un problema di flusso di costo minimo come mostrato in Figura 2.2. Gli archi dal nodo fittizio 0 ai nodi 1, ..., 12 rappresentano la produzione, mentre gli archi di tipo $(i, i+1)$ rappresentano il magazzino. I bilanci ai nodi sono scelti in modo da rappresentare la vendita di panettoni in ogni mese, al netto dei panettoni già presenti in magazzino (per il nodo 1) o di quelli che dovranno esservi lasciati (per il nodo 12); il bilancio al nodo 0 è la produzione totale di panettoni durante l'anno.

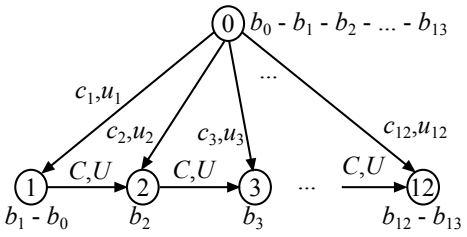


Figura 2.2: Il problema della PereCani

Esempio 2.3. Schedulazione di viaggi di camion

La ditta di trasporti Andamiento Lento deve organizzare una giornata lavorativa per i suoi camion. La ditta deve effettuare n viaggi, ognuno caratterizzato da un tempo di inizio, una località di origine, un tempo di percorrenza ed una località di destinazione. I camion della ditta, tutti uguali, all'inizio della giornata sono tutti nello stesso deposito, e devono tutti trovarsi nel deposito alla fine della giornata. La località di origine del generico viaggio i, $i = 1, \ldots, n$, può essere raggiunta da un camion che parte dal deposito prima dell'istante di partenza del viaggio corrispondente, ed è noto il costo c_i^I (del carburante) che questo comporta; analogamente, è sempre possibile raggiungere il deposito dalla località di destinazione del viaggio i, $i = 1, \ldots, n$, prima della fine della giornata lavorativa, ed è noto il costo c_i^F che questo comporta. Una coppia di viaggi (i, j) è detta *compatibile* se essi possono essere effettuati, nell'ordine dato, dallo stesso camion; cioè se è possibile per il camion che ha effettuato il viaggio i, partire dalla

località di destinazione di i e raggiungere la località di origine del viaggio j, prima del tempo di inizio di j. Per ogni coppia (i, j) di viaggi compatibile, è noto il costo c_{ij} del viaggio (a vuoto) tra la località di destinazione di i e la località di origine di j. Inoltre, esiste un costo fisso C che deve essere pagato per ogni camion che viaggia durante la giornata, indipendentemente dal numero di viaggi effettuati e dai km percorsi. Supponendo che la Andamiento Lento possegga abbastanza camion per eseguire tutti i viaggi (ovviamente non ne serviranno più di n), si vuole formulare il problema di scegliere quali viaggi far effettuare dallo stesso camion in modo da minimizzare il costo complessivo, dato dalla somma dei costi dei movimenti a vuoto (dal deposito alla prima origine, dalle destinazioni alle origini dei viaggi successivi e dalla destinazione dell'ultimo viaggio nuovamente al deposito) e dei costi fissi.

Questo problema può essere formulato come un problema di flusso di costo minimo nel seguente modo. Il grafo G ha $2n + 2$ nodi: un nodo origine s, un nodo destinazione t, e n coppie di nodi (i', i''), uno per ogni viaggio i, $i = 1, \ldots, n$. I nodi i' hanno bilancio 1, i nodi i'' hanno bilancio -1 mentre s e t hanno bilancio 0. Esistono archi (s, i') per ogni $i = 1, \ldots, n$, con costo c_i^I e capacità 1; analogamente, esistono archi (i'', t) per ogni $i = 1, \ldots, n$, con costo c_i^F e capacità 1. Per ogni coppia di viaggi (i, j) compatibile esiste un arco (i'', j') con costo c_{ij} e capacità 1. Infine, esiste un arco (t, s) con costo C e capacità infinita (anche se il flusso non può superare n). Un esempio di grafo per un problema con 5 viaggi è mostrato in Figura 2.3, in cui il viaggio 1 è compatibile col 2 e col 3, il viaggio 2 è compatibile col 3 e col 5, il viaggio 4 è compatibile col 5 mentre i viaggi 3 e 5 non sono compatibili con nessun viaggio. Per semplicità, nella figura sono mostrati i bilanci ai nodi ma non i costi e le capacità degli archi. I nodi i' ed i'' rappresentano rispettivamente l'inizio e la fine del viaggio i, ed il flusso rappresenta i camion: un'unità di flusso su un arco (s, i') indica che il viaggio i viene effettuato da un camion appena uscito dal deposito, un'unità di flusso su un arco (i'', j') indica che i viaggi i e j vengono effettuati (in sequenza) dallo stesso un camion mentre un'unità di flusso su un arco (j'', t) indica che il camion che effettua il viaggio j torna immediatamente dopo al deposito. I vincoli di equilibrio ai nodi i' ed i'' garantiscono rispettivamente che il viaggio i sia compiuto da esattamente un camion (o proveniente dal deposito oppure a seguito di un altro viaggio) e che il camion che ha compiuto i torni al deposito oppure compia un altro viaggio. I vincoli di equilibrio ai nodi s e t garantiscono che tutti i camion che escono dal deposito vi rientrino; il numero di camion usati è pari al flusso sull'arco (t, s).

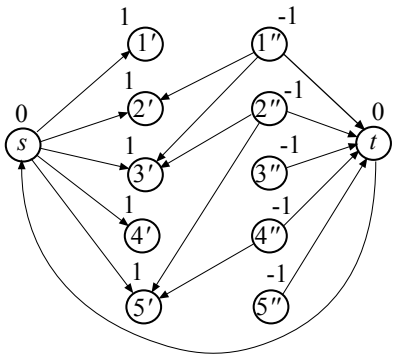


Figura 2.3: Schedulazione di viaggi

2.1.2 Trasformazioni equivalenti

Molti problemi reali possono essere formulati come problemi di flusso di costo minimo; questo è in parte dovuto alla grande flessibilità del modello stesso. Infatti, sui problemi di (MCF) si possono fare alcune assunzioni che ne semplificano la descrizione e l'analisi: se tali assunzioni non sono rispettate dall'istanza che effettivamente occorre risolvere, è sempre possibile costruire un'istanza di (MCF), equivalente a quella data, che le rispetti. Tali assunzioni sono:

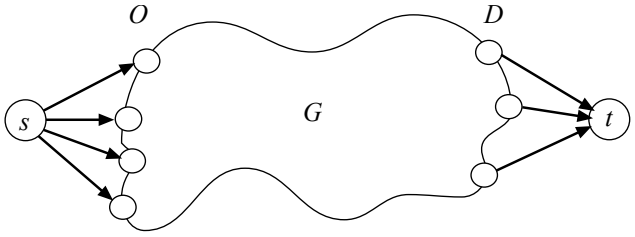


Figura 2.4: La rete ampliata G'

- *Singola sorgente – singolo pozzo.* Se si hanno più sorgenti e/o pozzi, è possibile introdurre una *rete ampliata* $G' = (N', A')$ in cui $N' = N \cup \{s, t\}$ ed $A' = A \cup \{(s, j) : j \in O\} \cup \{(i, t) : i \in D\}$, dove i nodi fittizi s e t sono "la nuova sorgente e il nuovo pozzo". Ad ogni arco fittizio $(s, j) \in FS(s)$ viene associata una capacità $u_{sj} = -b_j$, uguale cioè al flusso in ingresso a $j \in O$ nel problema originario; analogamente, ad ogni arco fittizio $(i, t) \in BS(t)$ viene associata una capacità $u_{it} = b_i$, uguale cioè al flusso in uscita da $i \in D$ nel problema originario; tutti gli archi fittizi hanno costo 0. L'offerta di s e la domanda di t sono date da

$$b_s = \sum_{j \in O} b_j \qquad b_t = \sum_{i \in D} b_i$$

  mentre tutti gli altri nodi sono di trasferimento ($b_i = 0$ per $i \neq s, t$). Un esempio di trasformazione della rete G nella rete ampliata G' è mostrato in Figura 2.4. È facile dimostrare che ad ogni flusso ammissibile x' di G' corrisponde un flusso ammissibile x di G, e viceversa; infatti x' satura tutti gli archi (s, j) (e, di conseguenza, tutti gli archi (i, t)) e quindi le offerte e le domande ai nodi in O e in D sono rispettate.

- *Capacità inferiori nulle.* Se un arco (i, j) ha capacità inferiore $l_{ij} \neq 0$, questo vuol dire che

il flusso x_{ij} dovrà essere almeno pari a l_{ij}. Quindi, è possibile costruire un'istanza equivalente sottraendo l_{ij} a b_j e u_{ij}, sommando l_{ij} a b_i, ed aggiungendo un termine costante $c_{ij}l_{ij}$ alla funzione obiettivo: questo equivale a sostituire la variabile di flusso x_{ij} con $x'_{ij} = x_{ij} - l_{ij}$, e considerare l_{ij} unità di flusso permanentemente presenti sull'arco (i, j). È facile verificare che il problema dell'Esempio 2.3 è equivalente al problema di (MCF) in cui tutti i bilanci ai nodi sono nulli ed esistono archi (i' , i'') con capacità sia inferiore che superiore pari a 1.

- *Nessuna capacità associata ai nodi.* In alcuni casi esiste una limitazione superiore u_i e/o una limitazione inferiore l_i alla massima quantità di flusso che può attraversare un nodo $i \in N$. Per questo, è sufficiente costruire un nuovo grafo G' in cui il nodo i è sostituito da due nodi i' ed i''. Tutti gli archi $(j , i) \in BS(i)$ vengono sostituiti con archi del tipo (j , i'), mentre gli archi $(i , j) \in FS(i)$ vengono sostituiti con archi del tipo (i'' , j), con costi e capacità uguali agli archi originali. Inoltre, viene aggiunto un arco (i' , i'') con costo 0 e capacità superiore ed inferiore rispettivamente u_i e l_i. La domanda del nodo i viene attribuita ad i' se è positiva ed a i'' se negativa.

- *Eccesso di domanda o di offerta.* In alcuni problemi, il valore $-b_i$ per $i \in O$ non rappresenta l'offerta al nodo i, ma la *massima* offerta che il nodo i può fornire alla rete. Se i valori b_j per $j \in D$ rappresentano effettivamente la domanda di flusso dei nodi pozzo, il problema che si vuole risolvere è quello di determinare un flusso che soddisfi tutte le domande e per cui ogni nodo $i \in O$ fornisca al più $-b_i$ unità di flusso; chiaramente dovrà risultare $-\sum_{i \in O} b_i \geq \sum_{i \in D} b_i$. Una formulazione del problema si ottiene da (MCF) modificando i vincoli (2.2) per $i \in O$ in

$$\sum_{(j , i) \in BS(i)} x_{ji} - \sum_{(i , j) \in FS(i)} x_{ij} \geq b_i \qquad i \in O \ .$$

  È facile trasformare questo problema in un'istanza di (MCF) ricorrendo ancora una volta ad una rete ampliata $G' = (N' , A')$, in cui $N' = N \cup \{ s \}$ e $A' = A \cup \{ (s , i) : i \in O \}$. In altri termini, si ha un nuovo nodo sorgente s la cui offerta coincide con la domanda globale dei nodi pozzo ($b_s = -\sum_{j \in D} b_j$), mentre si pone $b_i = 0$ per ogni $i \in O$ in modo che risulti $\sum_{i \in N'} b_i = 0$. Per ogni $i \in O$, l'arco fittizio (s , i) ha costo 0 e capacità $u_{si} = -b_i$. La trasformazione effettuata permette, una volta calcolati i flussi su G', di conoscere, per ciascuna sorgente $i \in O$, la capacità produttiva effettivamente utilizzata, x_{si}, e quella non utilizzata, $u_{si} - x_{si}$. In modo analogo si tratta il caso in cui per ciascun nodo $i \in O$ è data l'offerta $-b_i$, mentre per ciascun nodo $j \in D$ è data una limitazione superiore b_j del valore che la domanda può assumere.

È opportuno enfatizzare che (MCF) è uno tra i più generali problemi di ottimizzazione su reti che ammettano algoritmi polinomiali. Questa generalità implica però che gli algoritmi risolutivi per (MCF) sono tra i più complessi fra quelli per problemi di ottimizzazione su reti. Nel seguito introdurremo prima i problemi "più facili", per i quali si possono definire algoritmi risolutivi relativamente semplici. Tali algoritmi verranno poi utilizzati all'interno di approcci per (MCF).

2.2 Cammini di costo minimo

Il problema della determinazione di cammini di costo minimo, detti anche *cammini minimi*, è uno tra i più semplici, ma allo stesso tempo tra i più importanti problemi di ottimizzazione su reti. Ad esempio, il calcolo dei percorsi all'interno dei dispositivi GPS, negli smartphones e nei servizi quali Google Maps richiede la soluzione di problemi di cammino minimo su grafi di dimensione molto grande (milioni di nodi ed archi), rappresentanti la rete stradale e/o di trasporto pubblico, in tempi estremamente brevi. Questo però è solamente un esempio di possibile utilizzo, nel quale la formulazione in termini di problema di cammini minimi è una naturale e diretta conseguenza delle caratteristiche della realtà modellata. Come già osservato per (MCF), un problema di cammini minimi può servire anche a formulare e risolvere problemi che apparentemente non hanno alcun rapporto con i grafi e le reti.

Esempio 2.4. Ispezioni su una linea di produzione

Una linea di produzione h n celle di lavorazione. Ogni lotto è costituito da B pezzi che passano attraverso le n celle ed in ciascuna di esse subiscono una lavorazione. La probabilità di produrre un difetto in un pezzo nella cella i è p_i. Possono essere fatte ispezioni alla fine di ogni lavorazione: le ispezioni vengono fatte su tutti i pezzi del lotto e quelli trovati difettosi vengono scartati. Non essendo accettabile l'invio ai clienti di pezzi difettosi, viene comunque fatta una ispezione alla fine; tuttavia può essere conveniente effettuare ispezioni già dopo le prime lavorazioni in modo da evitare il costo di lavorazioni effettuate su pezzi difettosi e quindi da scartare. Sono dati il costo unitario q_i di lavorazione alla cella i, il costo fisso f_{ij} di ispezione di un lotto all'uscita della cella j nell'ipotesi che la precedente ispezione fosse stata effettuata all'uscita della cella $i(< j)$, ed il costo unitario h_{ij} di una ispezione effettuata all'uscita della cella j, nell'ipotesi che la precedente ispezione fosse stata effettuata all'uscita della cella $i(< j)$. Il numero atteso di pezzi non difettosi alla fine della lavorazione i è dato da

$$B_i = B \prod_{k=1}^{i} (1 - p_k);$$

B_i è il numero di pezzi su cui si effettueranno le lavorazioni nelle celle successive alla cella i, sino a quella in cui si effettuerà una nuova ispezione. Il costo di un'ispezione effettuata alla cella j nell'ipotesi che la precedente sia stata effettuata alla cella $i(< j)$ è dato da $f_{ij} + B_i h_{ij}$; sommando ad esso il costo di lavorazione dei pezzi in tutte le celle da $i+1$ a j comprese, si ottiene il costo globale (produzione e ispezione) nel segmento produttivo da i escluso a j compreso:

$$c_{ij} = f_{ij} + B_i h_{ij} + B_i \sum_{k=i+1}^{j} q_k \ .$$

Nel seguito con 0 indicheremo una cella fittizia, che precede l'inizio del processo produttivo, affinché siano definiti i valori f_{0j}, h_{0j}, c_{0j} e $B_0 = B$ relativi al caso in cui la prima ispezione sia effettuata nella cella j.

Il problema di determinare il piano di ispezioni ottimo, cioè decidere quando effettuare le ispezioni in modo da minimizzare il costo globale (la somma dei costi di produzione e di quelli di ispezione), può essere formulato come il problema di cercare un cammino di costo minimo dal nodo 0 al nodo n, nel grafo $G = (N, A)$, con $N = \{0, 1, \ldots, n\}$ e $A = \{(i, j) : i \in N \setminus \{n\}, j > i\}$, in cui ad ogni arco (i, j) è associato il costo c_{ij} sopra definito. In Figura 2.5 è mostrato il grafo nel caso di $n = 4$.

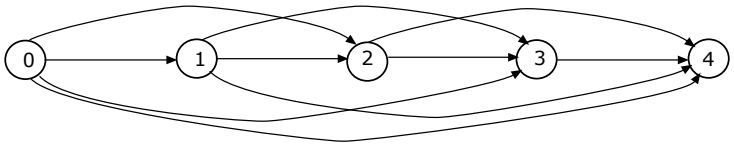


Figura 2.5: Grafo associato al problema delle ispezioni

È facile dimostrare che ogni cammino orientato del grafo da 0 a n corrisponde a un piano di ispezione e, viceversa, qualunque piano di ispezione è rappresentato da un cammino orientato da 0 a n, il cui costo (come somma dei costi degli archi) è proprio il costo globale di produzione e ispezione.

2.2.1 Il problema

Sia $G = (N, A)$ un grafo orientato e pesato dove ad ogni arco $(i, j) \in A$ è associato un costo $c_{ij} \in \mathbb{R}$. Per ogni cammino P in G, il costo $C(P) = \sum_{(i,j) \in P} c_{ij}$ è dato dalla somma dei costi degli archi che lo costituiscono. Dati due nodi r e t, definiamo $\mathcal{P}_{rt}$ l'insieme dei cammini (orientati) che connettono r a t. Il *problema del cammino minimo da r a t* ((SP), da *Shortest Path* problem) è quindi

$$\text{(SP)} \qquad \min \left\{ C(P) \ : \ P \in \mathcal{P}_{rt} \right\} \ . \tag{2.5}$$

Come già anticipato nell'Esempio 1.16, il problema (2.5) può essere formulato come un particolare problema di flusso di costo minimo su G, in cui gli archi hanno capacità infinita, i costi sono quelli del problema del cammino minimo, r è l'unica sorgente, che produce un'unità di flusso, t è l'unico pozzo, che richiede un'unità di flusso, mentre ogni altro nodo è di trasferimento. In altri termini, (2.4) in cui

$$u_{ij} = +\infty \qquad (i, j) \in A \qquad , \qquad b_i = \begin{cases} -1 & \text{se } i = r \\ 1 & \text{se } i = t \\ 0 & \text{altrimenti} \end{cases} \qquad i \in N \ .$$

Infatti, il modo migliore per soddisfare la richiesta di un'unità di flusso da parte di t è inviarla lungo un cammino di costo minimo da r a t. Si noti che, nel caso in cui esistano più cammini aventi uguale lunghezza minima, si possono inviare frazioni dell'unità di flusso lungo cammini diversi; il flusso ottimo può cioè, in questo caso, corrispondere a più di un cammino minimo. Per evitare questo inconveniente si può imporre un *vincolo di integralità sul flusso*, ossia richiedere che $x \in \mathbb{Z}^m$ (il che in effetti significa $x \in \{0, 1\}^m$), come appunto fatto nell'Esempio 1.16. Questo normalmente renderebbe il problema

"difficile" (cf. il Capitolo 4), ma in questo caso, come vedremo, la condizione viene "naturalmente" soddisfatta dagli algoritmi che svilupperemo.

È possibile che (2.5) sia vuoto: questo accade se e solo se in G non esiste nessun cammino da r a t, il che può essere facilmente verificato mediante una visita del grafo (si veda il §B.3). È anche possibile che (2.5) sia inferiormente illimitato: questo accade se e solo se in G esiste un *ciclo negativo*, cioè un ciclo orientato il cui costo sia negativo, raggiungibile da r e dal quale t sia raggiungibile. Infatti, consideriamo un cammino da r a t che includa un tale ciclo: percorrendo il ciclo si ottiene un cammino *non semplice*, e più volte si percorre il ciclo più si riduce il costo del cammino, mostrando così che il problema non è inferiormente limitato. Vedremo nel seguito che è possibile verificare in tempo polinomiale se un grafo contiene oppure no un ciclo negativo.

Se il grafo non contiene cicli negativi, rimuovendo i cicli da un cammino non semplice si ottiene un cammini semplice non più costoso. Pertanto, esiste sempre una soluzione ottima che è un cammino semplice. In altri termini, *in assenza di cicli negativi* (2.5) *coincide col problema del cammino semplice di costo minimo* (o cammino semplice minimo). Invece, *in presenza di cicli negativi il problema del cammino semplice minimo è $\mathcal{NP}$-arduo.* È infatti possibile esprimere in questo modo il problema di determinare un *cammino Hamiltoniano minimo* su un grafo, problema che è notoriamente $\mathcal{NP}$-arduo.

Esercizio 2.2. Si riconduca il problema del commesso viaggiatore (cf. l'Esempio 1.7) al problema del cammino Hamiltoniano minimo.

Per questo si definiscano, per lo stesso grafo G, dei nuovi costi $c'_{ij} = c_{ij} - M$ dove M è un numero "molto grande", ad esempio $M = (n-1)c_{max} + 1$ dove c_{max} è il massimo dei valori assoluti dei costi (originali) degli archi. Se esiste un cammino Hamiltoniano da r a t, il cammino semplice di costo minimo da r a t con i nuovi costi c' sarà sicuramente Hamiltoniano: infatti i cammini Hamiltoniani sono i più lunghi possibili (in termini di numero di archi) tra i cammini semplici su un grafo, ed i costi c' sono tali per cui qualsiasi cammino con $k+1$ archi ha un costo inferiore di qualsiasi cammino con k archi. Pertanto, se il cammino semplice di costo minimo individuato su G con costi c' non è Hamiltoniano, allora il problema del cammino Hamiltoniano minimo è vuoto; altrimenti il cammino semplice di costo minimo su G con costi c' corrisponde ad un cammino Hamiltoniano minimo con i costi originari c. Questo è dovuto al fatto che tutti i cammini Hamiltoniani hanno $n-1$ archi, e quindi la differenza del costo di un cammino Hamiltoniano usando i costi c' ed i costi c è la costante $(n-1)M$. Pertanto la modifica dei costi non cambia l'ordinamento dei cammini Hamiltoniano rispetto ai costi originari, e quindi assicura che un cammino semplice minimo sia un cammmino Hamiltoniano minimo (qualora ne esista uno).

Esercizio 2.3. Si noti che quando non esistono cicli negativi si possono aggiungere vincoli di capacità $x_{ij} \leq 1$ su tutti gli archi senza cambiare la soluzione: questo rende il problema limitato anche in presenza di cicli negativi. In aggiunta, è possibile utilizzare le trasformazioni viste nel paragrafo 2.1.2 per modificare il grafo in modo tale da imporre il vincolo che per ciascun nodo passi al più un'unità di flusso. Si discuta perché il problema di flusso così ottenuto (che può essere risolto in tempo polinomiale) *non* è una formulazione appropriata del problema del cammino semplice minimo.

È possibile considerare un problema più generale rispetto a (2.5): data una radice r, determinare in G un cammino di costo minimo da r a i, per ogni $i \neq r$. È facile vedere che il problema può essere formulato come

$$\min \left\{ \sum_{i \neq r} C(P_i) \ : \ P_i \in \mathcal{P}_{ri} \quad i \neq r \right\} \ . \tag{2.6}$$

Infatti, la scelta del cammino per un dato nodo i non influenza la scelta del cammino per tutti gli altri nodi; quindi il modo migliore per minimizzare la somma dei costi di tutti i cammini è quella di selezionare per ogni nodo il cammino di costo minimo. Il motivo per cui usualmente si considera (2.6) invece di (2.5) è che, come vedremo, nel caso peggiore la determinazione di un cammino minimo per un solo nodo destinazione richiede di determinare anche tutti gli altri cammini minimi. Inoltre, in molte applicazioni si debbono calcolare più cammini minimi aventi un'origine comune. Valgono per (2.6) molte delle considerazioni fatte in precedenza per (2.5); in particolare, il problema può essere formulato come un problema di flusso di costo minimo su G in modo analogo, con l'unica differenza

che la radice r è la sorgente di $n-1$ unità di flusso, mentre ogni altro nodo i è un nodo pozzo che richiede un'unità di flusso, ossia

$$b_i = \begin{cases} -(n-1) & \text{se } i = r \\ 1 & \text{altrimenti} \end{cases} \qquad i \in N \ .$$

Si noti che la funzione obiettivo è la *somma* delle lunghezze di tutti i cammini P_i, $i \neq r$. Quindi, se nella soluzione ottima un certo arco (i, j) è contenuto in k distinti cammini di costo minimo da r a k diversi nodi, il costo di quell'arco viene conteggiato k volte, il che corrisponde al fatto che nella formulazione di flusso si ha $x_{ij} = k$. Se G è privo di cicli negativi, (2.6) ha una soluzione ottima finita $\{ P_i \ , \ i \neq r \}$ se e solo se esiste almeno un cammino da r a ciascun altro nodo i. In effetti si può assumere senza perdita di generalità che questo accada: infatti, è sempre possibile aggiungere un arco fittizio (r, i), per ogni $i \neq r$ tale che $(r, i) \notin A$, con costo "elevato" $c_{ri} = M$, ad esempio $M = (n-1)c_{max}+1$, dove $c_{max} = \max\{ |c_{ij}| : (i, j) \in A \}$. Il cammino di costo minimo sarà costituito dal solo arco fittizio (r, i) solo se non esiste alcun cammino da r a i; infatti, se tale cammino esistesse il suo costo sarebbe certamente inferiore a M.

È facile verificare che, tra tutte le soluzioni ottime di (2.6), ne esiste almeno una in cui *l'unione dei cammini P_i forma un albero di copertura per G radicato in r e orientato*, ossia un albero di radice r i cui archi sono orientati da r verso le foglie. Infatti, se P_i è un cammino minimo da r a i e j è un nodo interno al cammino, allora il sottocammino di P_i che arriva sino a j è a sua volta un cammino minimo da r a j. Quindi, se esistono più cammini minimi da r ad un certo nodo i, è possibile selezionarne uno, P_i, ed imporre che i cammini minimi da r verso altri nodi, aventi i come nodo intermedio, abbiano P_i come sottocammino da r ad i. Ogni soluzione ottima di (2.6) che possa essere rappresentata mediante un albero di copertura orientato di radice r è detta un *albero di cammini minimi di radice r*. Nel seguito considereremo quindi la seguente forma equivalente di (2.6): determinare in G un albero di cammini minimi di radice r. Questo viene detto *problema dell'albero di cammini minimi* ((SPT), da Shortest Path Tree), o, più semplicemente, *problema dei cammini minimi.*

2.2.2 Alberi, etichette e condizioni di ottimo

Sia $T = (N, A_T)$ una soluzione ammissibile per (SPT), ossia $A_T \subseteq A$ con $|A_T| = n-1$ è un albero di copertura radicato in r e orientato. Vogliamo verificare se T sia una soluzione ottima; per questo non deve accadere che per qualche nodo $i \neq r$, esista un cammino orientato da r ad i di costo minore di $C(P_i^T)$, dove P_i^T è l'unico cammino da r ad i in T. Per fare questo calcoliamo il costo dei cammini in T: costruiamo quindi un vettore di *etichette dei nodi* $d \in \mathbb{R}^n$ tale che $d_i = C(P_i^T)$, per $i \neq r$, e $d_r = 0$. Il vettore d può essere facilmente determinato per mezzo di una procedura di visita dell'albero a partire dalla radice r. Si noti che, se l'albero contiene l'arco (i, j), allora $d_i + c_{ij} = d_j$; infatti, l'unico cammino da r a j è formato dal sottocammino da r ad i, di costo d_i, e dall'arco (i, j), che ha costo c_{ij}.

Esercizio 2.4. Si particolarizzi la procedura *Visita* in modo che, dato un albero T con costi sugli archi, calcoli il vettore di etichette d.

Dato il vettore delle etichette d corrispondente a T, è possibile verificare se qualche arco $(i, j)) \notin A_T$ può essere utilizzato per costruire un cammino da r a j migliore di P_j^T. Infatti, supponiamo che per un qualche arco (i, j) risulti $d_i + c_{ij} < d_j$, e sia h il predecessore di j in T (il nodo immediatamente precedente j nel cammino P_j^T): sostituendo nell'albero l'arco (h, j) con (i, j) si ottiene un nuovo albero T' in cui il nodo j è raggiunto con un cammino di costo inferiore, come mostrato in Figura 2.6. Se invece ciò non accade per alcun arco, allora T è una soluzione ottima per (SPT). Per dimostrarlo utilizziamo il seguente lemma:

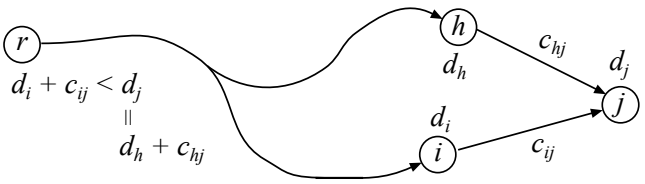


Figura 2.6: Un cammino di costo inferiore per j

Lemma 2.1. Sia $d \in \mathbb{R}^n$ un vettore di etichette dei nodi che verifica le *condizioni di Bellman*

$$d_i + c_{ij} \geq d_j \qquad (i\,,\,j) \in A \tag{2.7}$$

e $d_r = 0$; allora, per ogni $i \neq r$, d_i è una valutazione inferiore del costo del cammino minimo da r a i.

Dimostrazione Sia $P_i = \{ j_1\,,\, j_2\,,\, \ldots\,,\, j_k \}$ un qualsiasi cammino, non necessariamente semplice, da r a i (quindi $r = j_1$ e $j_k = i$). Per ipotesi si ha

$$\begin{array}{rcl} d_{j_k} & \leq & d_{j_{k-1}} + c_{j_{k-1}j_k} \\ d_{j_{k-1}} & \leq & d_{j_{k-2}} + c_{j_{k-2}j_{k-1}} \\ \vdots & & \vdots \\ d_{j_2} & \leq & d_{j_1} + c_{j_1 j_2} \end{array} \quad .$$

Sommando membro a membro, e sapendo che $d_{j_1} = d_r = 0$, si ottiene

$$d_{j_k} = d_i \leq C(P_i) = \textstyle\sum_{\ell=1}^{k-1} c_{j_\ell j_{\ell+1}} \quad ,$$

il che, essendo vero per ogni cammino $P_i \in \mathcal{P}_{ri}$, è vero in particolare per il cammino di costo minimo. ◇

Quindi, se il vettore di etichette d corrispondente a T verifica (2.7), allora T è chiaramente ottimo: per il Lemma 2.1, il cammino minimo da r ad un qualsiasi nodo $i \neq r$ non può costare meno di d_i, ed il cammino P_i^T ha esattamente quel costo. Se invece il vettore di etichette d corrispondente a T non verifica (2.7) allora, come abbiamo visto (cf. Figura 2.6), T non può essere ottimo. Si ottiene quindi il seguente risultato:

Teorema 2.1. Sia $T = (N\,,\, A_T)$ un albero di copertura radicato in r e orientato, e sia d il corrispondente vettore di etichette: T è un albero dei cammini minimi di radice r se e solo se d verifica le condizioni di Bellman (2.7).

Si noti che, per verificare l'ottimalità di T, abbiamo associato un'etichetta ad ogni nodo: anche se fossimo interessati solamente al cammino minimo da r ad un dato nodo t, per dimostrarne l'ottimalità attraverso le condizioni di Bellman dovremmo comunque associare un'opportuna etichetta anche a tutti gli altri nodi. Inoltre, un vettore di etichette che rispetta le condizioni di Bellman fornisce informazione sul costo del cammino minimo da r ad ogni altro nodo. Questo giustifica perché, usualmente, venga studiato il più generale problema dell'albero dei cammini minimi.

Se nel grafo esistesse un ciclo orientato di costo negativo, allora non esisterebbe (almeno per alcuni nodi del grafo) nessun limite inferiore al costo dei cammini minimi, e quindi non potrebbe esistere nessun vettore di etichette che rispetti le condizioni di Bellman. Infatti, sia $C = \{ j_1\,,\, j_2\,,\, \ldots\,,\, j_k\,,\, j_1 \}$ un ciclo negativo, e supponiamo per assurdo che esista un vettore d che rispetta (2.7): si ha

$$\begin{array}{rcl} d_{j_k} & \leq & d_{j_{k-1}} + c_{j_{k-1}j_k} \\ d_{j_{k-1}} & \leq & d_{j_{k-2}} + c_{j_{k-2}j_{k-1}} \\ \vdots & & \vdots \\ d_{j_2} & \leq & d_{j_1} + c_{j_1 j_2} \\ d_{j_1} & \leq & d_{j_k} + c_{j_k j_1} \end{array}$$

da cui, sommando membro a membro, si ottiene la contraddizione $0 \leq C(C) < 0$.

2.2.3 L'algoritmo *SPT*

Le condizioni di ottimalità presentate nel precedente paragrafo suggeriscono in modo naturale il seguente algoritmo per la determinazione di un albero dei cammini minimi:

- mediante una visita del grafo dei nodi del grafo, $O(n)$, si determina un albero di copertura radicato in r ed orientato, rappresentato dal vettore $p[\cdot]$ dei predecessori dei nodi, e le relative *etichette* d_i, che rappresentano il costo dell'unico cammino dell'albero da r a i;
- si fa partire un processo iterativo che:
  - mediante una visita degli archi del grafo, $O(m)$, controlla se esiste un arco $(i\,,\,j) \in A$ tale che $d_i + c_{ij} < d_j$, se ció non accade l'algoritmo termina in avendo ottenuto un albero ottimo, contenuto in $p[\cdot]$, insieme ad un *certificato* (le etichette che verificano le condizioni di Bellman) che ne dimostra in modo incontrovertibile l'ottimalità;

– se invece si determina in arco (i, j) che viola le condizioni di Bellman si modifica l'albero sostituendolo a $(p[j], j)$ (ossia ponendo $p[j] = i$), si calcola il vettore delle etichette corrispondente al nuovo albero e si itera; è facile vedere che il ricalcolo delle etichette va effettuato solo per i nodi nel sottoalbero di radice j, in cui tutte le etichette diminuiscono della stessa quantità $d_j - d_i - c_{ij} > 0$, e questo sarebbe in principio $O(n)$ se non fosse per il fatto che la funzione predecessore $p[\cdot]$ non permette di determinare efficientemente i soli nodi nella stella uscente che appartengono all'albero (senza guardarli tutti), il che rende anche questa operazione $O(m)$ al caso pessimo.

Questo algoritmo è un algoritmo di *ricerca locale* (si veda il §5.2) in cui l'intorno è rappresentato dall'insieme di tutti gli alberi di copertura radicati in r ed orientati che differiscono da quello corrente per al più un arco.

Per ottenere un algoritmo più efficiente è possibile integrare tutte e tre le visite in un'unica procedura che assomiglia a quella di visita, ma nella quale ciascun nodo può essere visitato più di una volta. L'idea fondamentale è quella dell'*aggiornamento differito delle etichette*. L'algoritmo mantiene cioè ad ogni iterazione una soluzione ammissibile, rappresentata da un vettore di predecessori $p[\cdot]$, una struttura, che indicheremo con Q, che contiene tutti i nodi i cui archi uscenti potrebbero violare le condizioni di Bellman (2.7), ed un vettore di etichette $d[\cdot]$ in cui $d[i]$, in questo caso, rappresenta in generale un'*approssimazione superiore* del costo dell'unico cammino dell'albero da r a i. All'inizio l'albero è formato da archi fittizi (r, i) aventi costo M molto elevato ($p[i] = r$ e $d[i] = M$ per $i \neq r$).

```
procedure (p, d) = SPT (G, c, r) {
  foreach( i ∈ N ) do { p[i] = r; d[i] = M; }
  d[r] = 0; Q = { r };
  do { select i from Q; Q = Q \ { i };
       foreach( (i, j) ∈ FS(i) ) do
         if( d[i] + c_ij < d[j] ) then {
           d[j] = d[i] + c_ij; p[j] = i; Q = Q ∪ { j };
           }
       } while( Q ≠ ∅ );
  }
```

Procedura 2.1: Algoritmo *SPT*

L'algoritmo *SPT* controlla se le condizioni di Bellman sono verificate e, ogni volta che trova un arco (i, j) per cui esse sono violate, cioè per cui $d_i + c_{ij} < d_j$, modifica il predecessore di j, ponendo $p[j] = i$, e l'etichetta di j ponendo $d_j = d_i + c_{ij}$. A seguito della diminuzione di d_j, tutti gli archi uscenti da j possono violare le condizioni di Bellman; j viene detto per questo *nodo candidato*, e viene inserito in Q. Ad ogni iterazione si verifica se Q è vuoto. In questo caso l'algoritmo termina avendo determinato una soluzione ottima; infatti il vettore $d[\cdot]$ rispetta le condizioni di Bellman e contiene le etichette dell'albero T rappresentato dal vettore $p[\cdot]$. Altrimenti, si estrae un nodo i da Q e si controlla se le condizioni (2.7) valgono per ciascun arco della sua stella uscente: per ognuno di tali archi (i, j) per cui ció non accade si pone $p[j] = i$ e l'etichetta di j viene aggiornata. Non si effettua però l'aggiornamento delle etichette di tutti i nodi del sottoalbero di radice j, ma si inserisce j in Q in modo che ciò avvenga, gradualmente, nelle iterazioni successive.

Esempio 2.5. Esecuzione dell'algoritmo *SPT*

Si vuole determinare un albero dei cammini minimi di radice $r = 1$ sul grafo in Figura 2.7(a), applicando l'algoritmo *SPT* in cui Q è implementato come una fila. Osserviamo preliminarmente che quando l'ordine di visita dei nodi dipende da quello in cui sono inseriti in Q, come in questo caso (ed in altri, cf. §2.2.5), diviene anche rilevante l'ordine in cui si visita la stella uscente del nodo i ad ogni iterazione, perché questo influenza l'ordine di inserimento in Q. Nelle implementazioni reali questo dipende tipicamente dai dettagli dell'implementazione delle strutture dati che rappresentano la $FS(\cdot)$, ed un ultima analisi spesso dal modo in cui vengono forniti in input i dati. Nelle dispense, per evitare ogni ambiguità useremo l'assunzione—utile ai soli fini didattici—che la stella uscente (ed entrante, ove rilevante) sia ordinata per nodo coda crescente, ossia, ad esempio, l'arco $(1, 3)$ sia esaminato prima dell'arco $(1, 3)$. Descriviamo adesso le iterazioni compiute dall'algoritmo, una ad una.

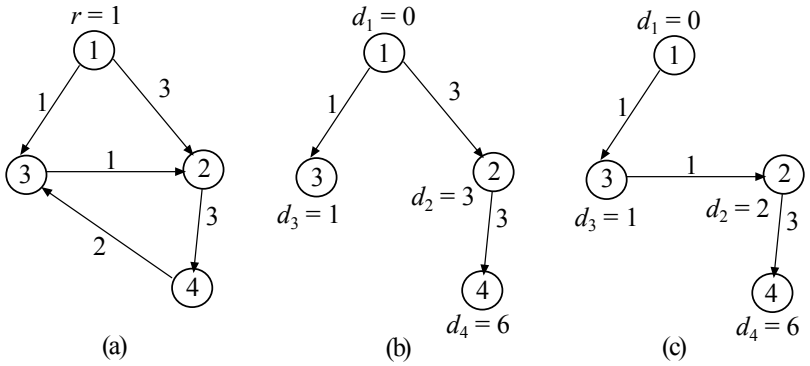


Figura 2.7: Un passo della procedura *SPT*

it. 0 Si pone $p[1] = p[2] = p[3] = p[4] = 1$, $d_1 = 0$ e $d_2 = d_3 = d_4 = (n-1)C_{max} + 1 = 3 \cdot 3 + 1 = 10$. Si pone $Q = \{1\}$.

it. 1 Si estrae $i = 1$ da Q (c'è solo quello) lasciando $Q = \emptyset$. Esaminando la stella uscente del nodo 1 si trova che $d_1 + c_{12} = 0 + 3 < d_2 = 10$ e $d_1 + c_{13} = 0 + 1 < d_3 = 10$; pertanto si pone $p[2] = p[3] = 1$ (il che non cambia in effetti il valore corrente), $d_2 = 3$, $d_3 = 1$, ed al termine della prima iterazione si ha dunque $Q = \{2, 3\}$.

it. 2 Si estrae $i = 2$ da Q, ed esaminando la sua stella uscente si trova che $d_2 + c_{24} = 3 + 3 < d_4 = 6$; pertanto si pone $p[4] = 2$ e $d_4 = 6$, come mostrato in Figura 2.7(b), terminando l'iterazione con $Q = \{3, 4\}$.

it. 3 Si estrae $i = 3$ da Q. La sua stella uscente contiene il solo arco $(3, 2)$, che viola le condizioni di Bellman in quanto $d_3 + c_{32} = 1 + 1 < d_2 = 3$. Si modificano pertanto l'etichetta ($d_2 = d_3 + c_{32} = 2$) e il predecessore del nodo 2 ($p[2] = 3$) e si inserisce nuovamente 2 in Q—il che mostra come la procedura *non* sia una visita—ottenendo l'albero di Figura 2.7(c) e $Q = \{4, 2\}$. L'etichetta del nodo 4 non viene modificata, e quindi non rappresenta il costo del cammino da 1 a 4 nell'albero rappresentato dal vettore $p[\cdot]$. Poiché però $2 \in Q$, questo avverrà in una qualche iterazione successiva.

it. 4 Si estrae $i = 4$ da Q. La sua stella uscente contiene il solo arco $(4, 3)$, che non viola le condizioni di Bellman in quanto $d_4 + c_{43} = 6 + 2 \geq d_3 = 1$, quindi non avviene nessuna modifica ai predecessori ed alle etichette. Si noti che il controllo viene fatto utilizzando il valore "sbagliato" $d_4 = 6$ dell'etichetta e quindi non c'è garanzia che l'arco non violi in effetti le condizioni di Bellman con il valore corretto della stessa, ma ancora $2 \in Q$ e quindi il controllo corretto avverrà in una qualche iterazione successiva.

it. 5 Si estrae nuovamente $i = 2$ (l'unico nodo rimanente) da Q. L'arco $(2, 4)$ della sua stella uscente viola le condizioni di Bellman in quanto $d_2 + c_{24} = 2 + 3 < d_4 = 6$; si modifica pertanto l'etichetta ($d_4 = d_2 + c_{24} = 5$), che adesso rappresenta correttamente il costo del cammino $P_4 = \{1, 3, 2, 4\}$ da $r = 1$ a 4 rappresentato in $p[\cdot]$. Formalmente l'algoritmo cambia anche il predecessore del nodo 4 ($p[4] = 2$), ma questo non ha alcun effetto perché il predecessore era già corretto: l'iterazione ha quindi avuto come unico effetto l'aggiornamento (differito) dell'etichetta, oltre naturalmente al fatto che adesso $Q = \{4\}$.

it. 6 Si estrae nuovamente $i = 4$ da Q, che quindi rimane vuota. La sua stella uscente contiene il solo arco $(4, 3)$, che come all'iterazione 4 non viola le condizioni di Bellman, ma adesso con il "giusto" valore dell'etichetta: $d_4 + c_{43} = 5 + 2 \geq d_3 = 1$. Avendo verificato questo non è necessario operare alcun cambiamento, ed essendo $Q = \emptyset$ al termine dell'iterazione l'algoritmo termina.

Per dimostrare la terminazione dell'algoritmo *SPT* abbiamo bisogno del seguente risultato:

Teorema 2.2. Ad ogni passo della procedura *SPT*, per ogni $i \in N$ il valore d_i dell'etichetta del nodo i rappresenta il costo di un cammino da r a i nel grafo G (oppure è M).

Dimostrazione La dimostrazione è per induzione sul numero delle iterazioni. La tesi è certamente vera alla prima iterazione, subito dopo la fase di inizializzazione. Assumendo che la tesi sia vera all'iterazione k, verifichiamo che lo sia anche all'iterazione $k + 1$. Sia j un nodo la cui etichetta è migliorata all'iterazione $k + 1$: il nuovo valore sarà $d_j = d_i + c_{ij}$ per qualche nodo i. Poiché l'etichetta di i è stata modificata in un'iterazione precedente, per ipotesi induttiva essa rappresenta il costo di un cammino da r ad i. Pertanto d_j è il costo di un cammino del grafo costituito da un sottocammino da r ad i, di costo d_i, e dall'arco (i, j). ◇

A questo punto possiamo dimostrare che, se il grafo non contiene cicli di costo negativo, la procedura *SPT* termina dopo un numero finito di passi. Infatti, per il Teorema 2.2 il valore d_i dell'etichetta di qualsiasi nodo i è sempre uguale al costo di un cammino del grafo G da r ad i. Osserviamo che il nodo i è inserito in Q solo quando la sua etichetta diminuisce: poiché il numero di cammini semplici

da r ad i è finito, d_i può diminuire solamente un numero finito di volte, e quindi il nodo i potrà essere inserito in Q solamente un numero finito di volte. Di conseguenza, dopo un numero finito di iterazioni si avrà $Q = \emptyset$ e la procedura terminerà. Il prossimo esempio mostra che, se invece il grafo contiene un ciclo negativo (e almeno uno dei nodi del ciclo è raggiungibile da r), allora la procedura *SPT non termina*.

Esempio 2.6. Effetto di un ciclo negativo

Si consideri nuovamente il problema di Figura 2.7(a), in cui però l'arco $(4, 3)$ abbia costo -5. Le prime iterazioni dell'algoritmo fino alla terza inclusa rimangono identiche, ma dalla quarta in poi il l'esecuzione cambia:

it. 4 Si estrae $i = 4$ da Q. La sua stella uscente contiene il solo arco $(4, 3)$, che non viola le condizioni di Bellman in quanto $d_4 + c_{43} = 6 - 5 \geq d_3 = 1$. Si noti come in questo caso le condizioni di Bellman valgano con l'uguaglianza (come per gli archi dell'albero, ma questo non lo è), il che rende ovviamente rilevante il fatto che il controllo sia siato fatto utilizzando il valore "sbagliato" $d_4 = 6$ dell'etichetta: se si fosse utilizzato il valore "giusto" si sarebbe scoperta una violazione delle condizioni di Bellman, cosa che non è accaduta ma accadrà in un'iterazione successiva.

it. 5 Questa iterazione rimane uguale: si estrae $i = 2$, l'arco $(2, 4)$ viola le condizioni di Bellman ($d_2 + c_{24} = 2+3 < d_4 = 6$), si modifica l'etichetta di 4 ($d_4 = d_2 + c_{24} = 5$) ma non di fatto il suo predecessore, si inserisce 4 in Q.

it. 6 Estraendo nuovamente $i = 4$ da Q si ottiene, a differenza dell'Esempio precedente, la violazione delle condizioni di Bellman: $d_4 + c_{43} = 4 - 5 < d_3 = 1$, il che causa il rietichettamento di 3 ($d_3 = 0$), il cambio di predecessore ($p[3] = 4$) e la re-inserzione di 3 in Q. Si noti che al termine di questa iterazione il vettore $p[\cdot]$ non descrive più un albero; infatti si ha $p[3] = 4$, $p[4] = 2$ e $p[2] = 3$, che individua il ciclo $C = \{3, 2, 4, 3\}$, non per caso di costo negativo.

it. 7 Si estrae $i = 3$ da Q, e l'arco $(3, 2)$ viola nuovamente le condizioni di Bellman in quanto $d_3 + c_{32} = 0 + 1 < d_2 = 2$. Si modifica pertanto l'etichetta ($d_2 = d_3 + c_{32} = 1$), ma non il predecessore, e si inserisce nuovamente 2 in Q.

it. 8 Si estrae $i = 2$, e l'arco $(2, 4)$ viola le condizioni di Bellman in quanto $d_2 + c_{24} = 1 + 3 < d_4 = 5$; si modifica pertanto l'etichetta ($d_4 = 1 + c_{24} = 4$), ma non il predecessore, e si inserisce nuovamente 4 in Q.

it. 9 Si estrae $i = 4$ da Q, e l'arco $(4, 3)$ viola le condizioni di Bellman in quanto $d_4 + c_{43} = 4 - 5 < d_3 = 0$; si rietichetta quindi 3 ($d_3 = -1$) e lo si inserisce nuovamente in Q.

È chiaro come a questo punto l'algoritmo sia "entrato in loop": i nodi del ciclo negativo $C = \{3, 2, 4, 3\}$ vengono inseriti in Q un numero infinito di volte, mentre il valore delle loro etichette diminuisce indefinitamente.

L'algoritmo *SPT* è un algoritmo molto generale il cui effettivo comportamento dipende dal modo con cui viene implementato l'insieme Q dei nodi candidati. In effetti, ad implementazioni diverse corrispondono comportamenti molto diversi in termini di complessità computazionale. Ad alto livello possiamo pensare a due scelte alternative:

1. Q è una *coda di priorità*, cioè un insieme in cui ogni elemento ha associato un valore (chiave), e la scelta dell'elemento da estrarre avviene sulla base di questo valore;
2. Q viene implementato come una *lista* e la scelta dell'elemento da estrarre è determinata dalla posizione dell'elemento nella lista.

Tali scelte corrispondono a strategie implementative diverse, realizzabili in modi molto differenti fra loro: nel seguito discuteremo alcune di queste possibili implementazioni e le conseguenze che esse hanno sull'efficienza dell'algoritmo.

2.2.4 Algoritmi a coda di priorità

L'insieme Q viene implementato come coda di priorità; ad ogni elemento i è cioè associata una chiave di priorità, che nel nostro caso è l'etichetta d_i, e la priorità di i cresce al decrescere di d_i. Le operazioni elementari eseguibili su Q sono:

- inserimento di un elemento con l'etichetta associata,
- modifica (riduzione) dell'etichetta di un elemento di Q,
- selezione dell'elemento con etichetta minima e sua rimozione da Q.

Chiamiamo *SPT.S* (da *Shortest-first*) la versione di *SPT* in cui ad ogni iterazione si estrae da Q un elemento ad etichetta minima. L'operazione "select i from Q;" viene pertanto realizzata come

$$\text{select } i \text{ from } Q \text{ such that } d_i = \min\{\, d_h \;:\; h \in Q \,\} \;.$$

Vale il seguente Teorema:

Teorema 2.3. [Dijkstra, 1959] Nel funzionamento di *SPT.S* su grafi con costi non negativi, ogni nodo viene inserito in Q (e rimosso da esso) al più una volta.

Dimostrazione Indichiamo con u_k e $d^k[\cdot]$ rispettivamente il nodo estratto da Q ed il vettore delle etichette all'iterazione k ($u_1 = r$). Vogliamo innanzitutto dimostrare che la successione dei valori delle etichette dei nodi estratti da Q è non decrescente, ossia che $d^{k+1}[\,u_{k+1}\,] \geq d^k[\,u_k\,]$, per ogni $k \geq 1$. Per questo basta considerare due casi:

- $d^{k+1}[\,u_{k+1}\,] = d^k[\,u_{k+1}\,]$, ossia l'etichetta di u_{k+1} non è cambiata durante la k-esima iterazione. In questo caso u_{k+1} apparteneva certamente a Q all'inizio della k-esima iterazione (un nodo può entrare in Q solo se il valore della sua etichetta diminuisce), e quindi $d^{k+1}[\,u_{k+1}\,] = d^k[\,u_{k+1}\,] \geq d^k[\,u_k\,]$ poiché u_k è uno dei nodi di Q con etichetta di valore minimo al momento in cui viene estratto.
- $d^{k+1}[\,u_{k+1}\,] < d^k[\,u_{k+1}\,]$, ossia l'etichetta di u_{k+1} è cambiata durante la k-esima iterazione. Questo significa che u_k è il predecessore di u_{k+1} all'inizio della $k+1$-esima iterazione, e quindi $d^{k+1}[\,u_{k+1}\,] = d^k[\,u_k\,] + c_{u_k,u_{k+1}}$. Da questo, dato che $c_{u_k,u_{k+1}} \geq 0$, si ottiene ancora una volta $d^{k+1}[\,u_{k+1}\,] \geq d^k[\,u_k\,]$.

Poiché i nodi vengono inseriti in Q quando il valore della loro etichetta decresce, se un nodo entrasse in Q una seconda volta, la sua etichetta avrebbe un valore inferiore a quello che aveva nel momento della sua prima estrazione. Ciò contraddice quanto appena dimostrato. ◇

Il fatto che un nodo non possa essere inserito in Q, e quindi estratto, più di una volta fa sì che il generico arco $(\,i\,,\,j\,)$ possa essere esaminato al più una volta, cioè quando viene selezionato il nodo i, e pertanto *SPT.S* ha complessità polinomiale. Inoltre, questa proprietà si dimostra molto utile nel caso in cui si sia in effetti interessati a determinare solamente il cammino minimo da r ad uno specifico nodo t, o comunque verso un sottoinsieme proprio dei nodi. Infatti, è immediato verificare che in questo caso si può far terminare l'algoritmo non appena tutti i nodi che si desidera raggiungere sono stati estratti dalla coda perché, per la proprietà appena mostrata, non appena ciò si verifica tutti i cammini minimi relativi risultano essere stati determinati.

Esercizio 2.5. Dimostrare, per *SPT.S*, che se i costi degli archi sono non negativi allora, ad ogni iterazione, il valore dell'etichetta di un nodo è il costo del cammino di T che va dall'origine a quel nodo, e non una sua approssimazione superiore.

Si può dimostrare invece che, nel caso di *costi negativi*, esistono grafi per i quali l'algoritmo esegue un numero esponenziale di iterazioni; per i dettagli si rimanda alla letteratura citata.

Esempio 2.7. Esecuzione dell'algoritmo *SPT.S*

Si vuole determinare l'albero dei cammini minimi di radice $r = 1$ sul grafo di Figura 2.8(a) con la procedura *SPT.S*.

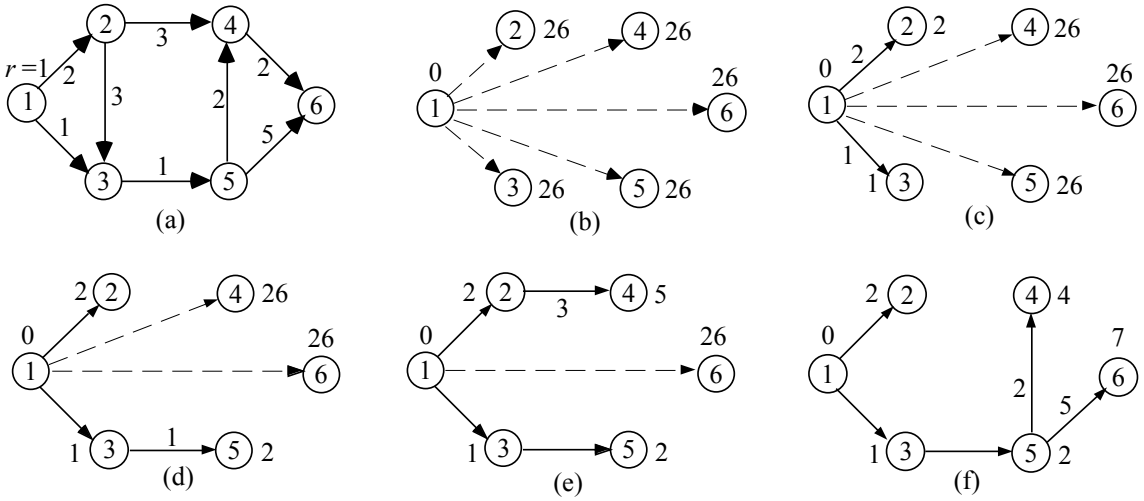
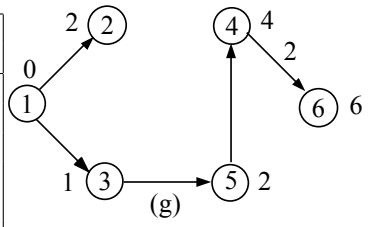


Figura 2.8: Alberi generati dall'Algoritmo *SPT.S*

Gli alberi da (b) a (g) sono quelli che si ottengono nell'inizializzazione e come risultato delle iterazioni dell'algoritmo riportate nella seguente tabella. Gli archi disegnati in queste figure corrispondono a quelli indicati dal vettore dei predecessori; quelli tratteggiati corrispondono agli archi fittizi di costo $M = 26$. I valori delle etichette sono indicati accanto ai nodi. Nell'ultima iterazione si seleziona il nodo 6, ma non si hanno modifiche di etichette e quindi l'albero rimane inalterato.

Iter.	Q	u	Archi esaminati	Etichette modificate	Predecessori modificati	Albero
1	$\{1\}$	1	$(1,2)$, $(1,3)$	d_2, d_3	p_2, p_3	(c)
2	$\{2,3\}$	3	$(3,5)$	d_5	p_5	(d)
3	$\{2,5\}$	2	$(2,3)$, $(2,4)$	d_4	p_4	(e)
4	$\{4,5\}$	5	$(5,4)$, $(5,6)$	d_4, d_6	p_4, p_6	(f)
5	$\{4,6\}$	4	$(4,6)$	d_6	p_6	(g)
6	$\{6\}$	6				(g)



Esercizio 2.6. Applicare *SPT.S* con radice $r = 1$ al grafo di Figura 2.9.

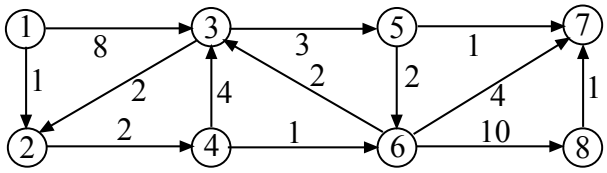


Figura 2.9: Un'istanza del problema (SPT)

Sono possibili diverse implementazioni dell'algoritmo *SPT.S*, che differiscono per il modo in cui è implementata la coda di priorità Q. La scelta dell'implementazione di Q non cambia il comportamento dell'algoritmo (si può pensare che la sequenza di estrazioni da Q sia indipendente da tale scelta), ma ne influenza la complessità e l'efficienza computazionale. Nel seguito discuteremo brevemente alcune possibili implementazioni.

Lista non ordinata

Q è implementata come una lista non ordinata, ad esempio mediante un vettore a puntatori, in cui i nodi vengono inseriti in testa o in coda e la selezione del nodo di etichetta minima viene effettuata per mezzo di una scansione completa della lista. Le operazioni elementari hanno quindi la seguente complessità:

inizializzazione delle etichette e della lista Q:	$O(n)$
selezione del nodo di etichetta minima:	$O(n)$
rimozione da Q del nodo di etichetta minima:	$O(1)$
inserzione di un nodo o modifica della sua etichetta:	$O(1)$

L'algoritmo risultante è noto come *algoritmo di Dijkstra*: è facile verificare che, su grafi con costi non negativi, questo algoritmo ha complessità $O(n^2)$. Infatti, dal Teorema (2.3) discende che non verranno estratti più di n nodi da Q: ad ogni estrazione del nodo di etichetta minima si scandisce l'intera lista in tempo $O(n)$ e quindi il costo totale delle operazioni di gestione della lista è $O(n^2)$. Siccome ogni nodo viene estratto da Q al più una volta, ogni arco (i, j) viene esaminato, una sola volta, se e quando i viene estratto da Q. Le operazioni sui singoli archi sono effettuate in tempo costante, e quindi costano complessivamente $O(m)$; dato che $m < n^2$, la complessità in tempo dell'algoritmo è $O(n^2)$.

Esercizio 2.7. In generale un grafo può avere "archi paralleli", ossia più copie dello stesso arco (i, j) con costi diversi; si esamini la complessità dell'algoritmo di Dijkstra in questo caso.

Heap binario bilanciato

Come abbiamo visto, la complessità dell'algoritmo di Dijkstra è fondamentalmente dovuta alla gestione dell'insieme Q: per questo, sono state proposte diverse strutture di dati per implementare Q in modo efficiente mantenendo l'insieme Q parzialmente ordinato. Una delle alternative più utilizzate è quella di realizzare Q mediante uno *heap binario blianciato*, in modo che il costo delle operazioni su Q divenga

inizializzazione delle etichette e dello heap Q:	$O(n)$
selezione del nodo di etichetta minima:	$O(1)$
rimozione da Q del nodo di etichetta minima:	$O(\log n)$
inserzione di un nodo o modifica della sua etichetta:	$O(\log n)$

Se i costi degli archi sono non negativi, le operazioni di ordinamento dello heap a seguito di inserimenti o rimozioni di nodi da Q sono al più $m+n$: pertanto, la versione di *SPT.S* che utilizza un heap binario ha complessità $O(m \log n)$. Si noti che tale complessità è migliore di quella dell'algoritmo di Dijkstra nel caso di grafi *sparsi* ($m \approx n$), mentre è peggiore di quella dell'algoritmo di Dijkstra nel caso di grafi *densi* ($m \approx n^2$). Sono stati proposte molte implementazioni di Q basate su differenti implementazioni di code di priorità, quali ad esempio i *Buckets*, i *Radix Heaps* ed i *Fibonacci Heaps*; per ulteriori dettagli si rinvia alla letteratura citata.

2.2.5 Algoritmi a selezione su lista

In questi algoritmi l'insieme Q viene implementato come una *lista*, cioè una *sequenza* di elementi su cui possono essere effettuate operazioni di rimozione ed inserzione alle estremità della sequenza, chiamate rispettivamente *testa* e *coda* della lista. Si noti che l'aggiornamento dell'etichetta di un nodo che appartiene a Q non influisce sulla posizione dell'elemento nella lista, ossia non causa la rimozione del nodo da Q ed il suo reinserimento in una posizione diversa (formalmente, aggiungere ad un insieme un elemento già presente non lo cambia). Esistono diversi tipi di liste; nel nostro caso, hanno particolare rilevanza

***fila*:** l'inserzione viene effettuata in coda e la rimozione dalla testa (regola *FIFO*);

***pila*:** l'inserzione e la rimozione vengono effettuate in testa (regola *LIFO*);

***deque*:** (double-ended queue, o lista a doppio ingresso) l'inserzione viene effettuata sia in testa che in coda e la rimozione solo dalla testa.

Indichiamo nel seguito con *SPT.L* le versioni di *SPT* nelle quali l'insieme Q è implementato come lista. La lista può essere realizzata in diversi modi (lista a puntatori, vettore di puntatori, lineare o circolare, semplice o doppia, ecc.), ed è sempre possibile fare in modo che le operazioni elementari ed il controllo di appartenenza di un elemento alla lista abbiano complessità costante, $O(1)$. La complessità di *SPT.L*, anche nel caso di costi negativi, dipende quindi linearmente dal numero di controlli delle condizioni di Bellman sugli archi uscenti dai nodi estratti da Q.

Fila

Esaminiamo ora l'algoritmo che si ottiene realizzando la lista Q come *fila* (*queue*), conosciuto in letteratura come *algoritmo di Bellman*: l'inserzione dei nodi avviene in coda e la rimozione dalla testa (regola FIFO).

Esempio 2.8. Esecuzione di *SPT.L.Queue*

Si vuole determinare l'albero dei cammini minimi di radice $r = 1$ sul grafo in Figura 2.8(a) con *SPT.L* e Q implementata come una *fila*. Gli alberi che vengono man mano costruiti sono indicati in Figura 2.10. La simbologia nelle figure e nella tabella seguente coincide con quella utilizzata nell'esempio 2.7.

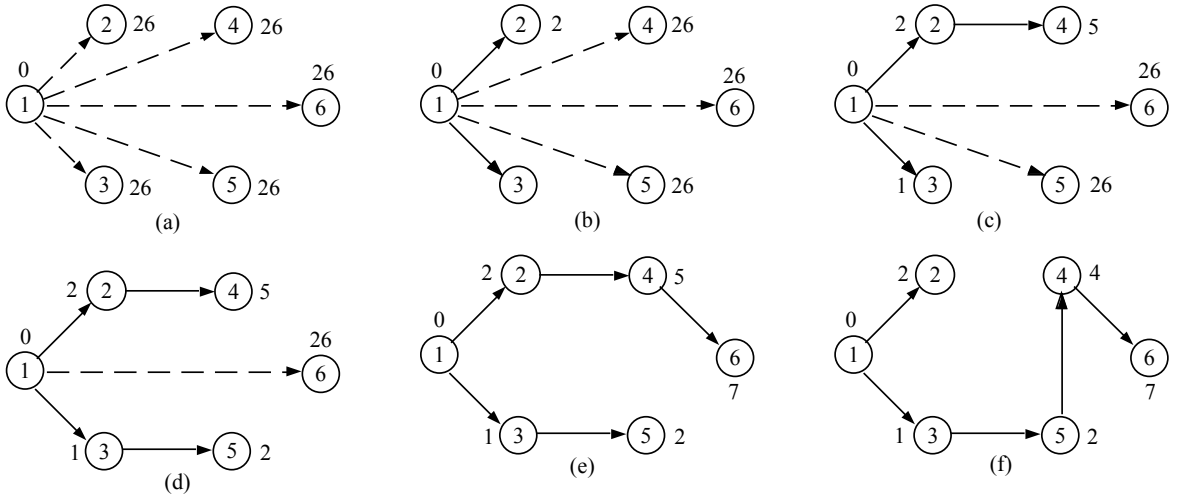
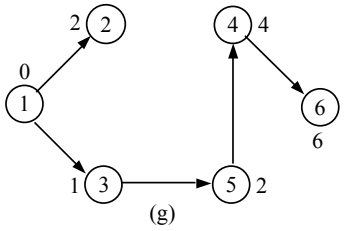


Figura 2.10: Alberi generati da *SPT.L.Queue*

Iter.	Q	u	Archi esaminati	Etichette modificate	Predecessori modificati	Albero
1	$\{1\}$	1	$(1,2),(1,3)$	d_2, d_3	p_2, p_3	(b)
2	$\{2,3\}$	2	$(2,3),(2,4)$	d_4	p_4	(c)
3	$\{3,4\}$	3	$(3,5)$	d_5	p_5	(d)
4	$\{4,5\}$	4	$(4,6)$	d_6	p_6	(e)
5	$\{5,6\}$	5	$(5,4),(5,6)$	d_4	p_4	(f)
6	$\{6,4\}$	6				(f)
7	$\{4\}$	4	$(4,6)$	d_6	p_6	(g)
8	$\{6\}$	6				(g)



Esercizio 2.8. Applicare *SPT.L.Queue* al grafo di figura 2.9 con radice $r = 1$.

L'utilizzo di una strategia FIFO corrisponde ad una "visita a ventaglio" (bfs) del grafo. Dimostreremo ora un'utile conseguenza di questo tipo di visita, ossia che l'algoritmo determina in sequenza, per uno stesso nodo i, i cammini di costo minimo da r ad i con un numero crescente di archi. Per questo dobbiamo definire il concetto di *fasi* dell'algoritmo: la fase 0 corrisponde all'inizializzazione, in cui $Q = \{ r \}$, mentre la generica fase $k+1$ inizia quando viene estratto da Q il primo nodo che vi è stato inserito nella fase k, e termina quando viene estratto da Q l'ultimo nodo che vi è stato inserito nella fase k (quindi la fase 1 coincide con la prima iterazione, al termine della quale sono stati visitati e messi in Q tutti i nodi raggiungibili direttamente da r). Si noti che, per la specifica scelta di Q come fila, i nodi inseriti in Q durante la fase k vengono estratti dopo quelli che erano già presenti in Q all'inizio della fase. Definendo $\bar{d}_i^k$ come la lunghezza minima tra tutte quelle dei cammini da r ad i che contengono al più k archi ($\bar{d}_i^k = +\infty$, o equivalentemente $\bar{d}_i^k = M$, se non ne esiste nessuno), si può dimostrare il seguente risultato:

Teorema 2.4. Sia $d^k[\cdot]$ il vettore delle etichette al termine della fase k: allora $d^k \leq \bar{d}^k$.

Dimostrazione Il teorema si basa sulla seguente caratterizzazione di $\bar{d}_i^k$, la cui correttezza è facile dimostrare:

$$\bar{d}_j^{k+1} = \min \left\{ \bar{d}_j^k \, , \, \left\{ \bar{d}_i^k + c_{ij} \; : \; (i\,,\,j) \in A \right\} \right\} \tag{2.8}$$

Infatti il cammino minimo con al più $k+1$ archi fino a j o ha in effetti al più k archi, oppure è formato da un arco $(i\,,\,j)$ e poi dal cammino minimo con k archi da r ad i. Durante la dimostrazione ci sarà utile assumere che $M = +\infty$, cosa che è sicuramente possibile, per semplificare gli argomenti evitando di distinguere il caso in cui un nodo sia oppure no già stato visitato in precedenza.

Il teorema può adesso mostrato per induzione sulle fasi. La proprietà è certamente vera al termine della fase 1, nella quale, come abbiamo già notato, si visitano i nodi raggiungibili direttamente da r, costruendo così tutti e soli i cammini di lunghezza 1, che sono ovviamente minimi. Assumiamo adesso che la proprietà sia vera al termine della fase k, e dimostriamo che allora è vera anche al termine della fase $k+1$. Indicheremo quindi con $d^k[\cdot]$ e Q^k rispettivamente il valore delle etichette e Q al termine della fase k. La relazione fondamentale che è necessario dimostrare è

$$d^{k+1}[\,j\,] \leq d^k[\,i\,] + c_{ij} \qquad \forall (\,i\,,\,j\,) \in A \;\; . \tag{2.9}$$

Si tratta di una "versione approssimata delle condizioni di Bellman". Infatti, poiché $d^{k+1}[\,i\,] \leq d^k[\,i\,]$, la condizione (2.9) è "più facile da soddisfare" delle condizioni di Bellman: queste non sono necessariamente soddisfatte al termine di ogni fase (anche perché in questo caso l'algoritmo terminerebbe), ma lo sarebbero se al posto di $d^{k+1}[\,i\,]$ si considerasse $d^k[\,i\,]$.

Per dimostrare (2.9) dobbiamo distinguere due casi: quello in cui i viene estratto da Q durante la fase $k+1$, e quello in cui ciò non succede. Per il funzionamento dell'algoritmo, il primo caso corrisponde a $i \in Q^k$, mentre il secondo a $i \notin Q^k$. Per quest'ultimo caso, possiamo affermare che

$$d^k[\,j\,] \leq d^k[\,i\,] + c_{ij} \qquad \forall(\,i\,,\,j\,) \in FS(i) \ .$$

Infatti la proprietà vale se i non è mai stato visitato, e quindi $d^k[\,i\,] = +\infty$, indipendentemente dal valore di $d^k[\,j\,]$. Se invece i è in un qualche momento stato in Q e poi ne è uscito (anche più volte), per il funzionamento dell'algoritmo la proprietà valeva al termine dell'ultima iterazione in cui i è uscito da Q. Ma dopo quel momento $d[\,i\,]$ non è più cambiata (altrimenti i sarebbe rientrato in Q) mentre le $d[\,j\,]$ possono solo essere diminuite, e quindi la proprietà è ancora valida. Poiché $d^{k+1}[\,j\,] \leq d^k[\,j\,]$, possiamo concludere che (2.9) è soddisfatta per tutti gli archi $(\,i\,,\,j\,) \in FS(i)$ per ogni $i \notin Q^k$. Consideriamo invece $i \in Q^k$; i viene estratto da Q in un qualche momento della fase $k+1$, e quindi possiamo affermare che (2.9) sarà valida, al termine di quell'iterazione, per tutti gli archi uscenti da i. Infatti, in quel momento vale $d[\,j\,] \leq d[\,i\,] + c_{ij}$ per ogni $(\,i\,,\,j\,) \in FS(i)$. Inoltre vale anche $d[\,i\,] \leq d^k[\,i\,]$, in quanto l'etichetta di i può essere diminuita, rispetto al valore che aveva al termine della fase k, nelle iterazioni della fase $k+1$ precedenti a quella in cui i viene estratto. Infine, $d[\,j\,]$ può diminuire ulteriormente nelle iterazioni successive della fase $k+1$. Quindi, (2.9) è soddisfatta anche per tutti gli archi $(\,i\,,\,j\,) \in FS(i)$ per ogni $i \in Q^k$.
Abbiamo quindi mostrato che (2.9) vale per tutti gli archi; da questo, applicando l'ipotesi induttiva $d^k \leq \bar{d}^k$ si ottiene

$$d^{k+1}[\,j\,] \leq \min\left\{\, d^k[\,i\,] + c_{ij} \ :\ (\,i\,,\,j\,) \in A \,\right\} \leq \min\left\{\, \bar{d}^k_i + c_{ij} \ :\ (\,i\,,\,j\,) \in A \,\right\} \ .$$

Inoltre, $d^{k+1}[\,j\,] \leq d^k[\,j\,] \leq \bar{d}^k[\,j\,]$; ciò deriva dal fatto che le etichette sono non crescenti, ed ancora dall'ipotesi induttiva. Possiamo finalmente concludere che

$$d^{k+1}[\,j\,] \leq \min\left\{\, \bar{d}^k_j \ ,\ \left\{\, \bar{d}^k_i + c_{ij} \ :\ (i,j) \in A \,\right\} \,\right\} = \bar{d}^{k+1}_j \ .$$

dove per l'ultima uguaglianza abbiamo usato (2.8). ◇

Il teorema precedente ha alcune interessanti conseguenze. Intanto, interrompendo opportunamente l'algoritmo si possono ottenere valutazioni inferiori sulla lunghezza dei *cammini minimi vincolati*, ossia i cammini minimi soggetti all'ulteriore condizione che il numero di archi del cammino non possa superare una certa soglia prefissata k.

Esercizio 2.9. Si discuta sotto quali condizioni le valutazioni inferiori sono esatte (suggerimento: si faccia riferimento al Teorema 2.2 ed al Teorema B.1).

Inoltre, il teorema mostra che, in assenza di cicli negativi, nessun nodo può essere inserito più di $n-1$ volte in Q, e quindi che il numero di volte in cui si esamina un nodo o un arco è limitato superiormente da n. Infatti, per definizione un nodo può uscire da (ed entrare in) Q al massimo una volta per fase, ed in assenza di cicli negativi non possono esserci più di $n-1$ fasi. Da ciò segue che:

- siccome tutte le operazioni sui nodi (estrazione e rimozione da Q) e sugli archi (loro scansione, controllo delle condizioni di Bellman, eventuale aggiornamento di predecessore ed etichetta) sono implementabili in modo da avere complessità costante, la complessità della procedura è dominata dal massimo numero di volte che si esamina lo stesso arco $(n-1)$ per il numero di archi (m), e quindi è $O(mn)$;
- *SPT.L.Queue* può essere utilizzata per controllare se un grafo orientato possiede cicli negativi contando il numero di estrazioni da Q di ciascun nodo: appena un nodo i viene estratto per l'n-esima volta, si può affermare con certezza che quel nodo appartiene ad un ciclo negativo. In effetti, si può mostrare che un ciclo negativo può essere effettivamente determinato percorrendo all'indietro, a partire dal nodo i trovato, il vettore di predecessori $p[\cdot]$ fino al momento in cui non si incontra nuovamente i.

Esercizio 2.10. Scrivere la procedura *SPT.L* in cui Q è una fila ed è presente il controllo sui cicli negativi, assicurandosi che tutte le operazioni siano implementate in modo tale che la complessità sia $O(mn)$.

Liste a doppio ingresso

Nella letteratura scientifica sono state proposte altre realizzazioni dell'insieme Q come lista. Molto utilizzata è la lista a doppio ingresso, o *deque*, in cui i nodi sono inseriti in coda a Q la prima volta, mentre tutte le altre volte vengono inseriti in testa a Q. Si ottiene pertanto una politica ibrida *LIFO–FIFO*; in effetti, la lista Q può essere interpretata come una coppia di liste Q' e Q'' connesse in serie,

vedi Figura 2.11. Q' conterrà solo i nodi reinseriti in Q mentre Q'' conterrà solo i nodi inseriti per la prima volta in Q ed ancora non rimossi. Il nodo testa di Q'' viene rimosso solo se $Q' = \emptyset$, pertanto Q' è una *pila* (stack) e Q''è una *fila* (queue).

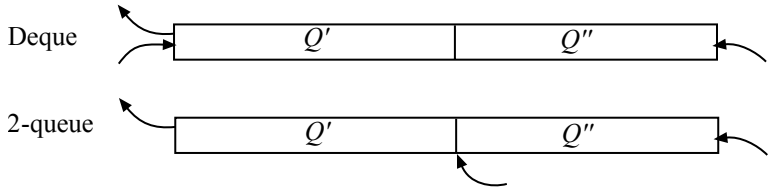


Figura 2.11: Liste a doppio ingresso

Esercizio 2.11. Applicare *SPT.L.deque* al grafo di figura 2.9 con radice $r = 1$.

La motivazione per l'uso di una deque risiede nel fatto che, se un nodo i viene inserito in Q dopo essere stato precedentemente rimosso, la sua etichetta d_i è stata utilizzata per aggiornare etichette di altri nodi, discendenti di i nell'albero corrente. Una nuova inserzione di i in Q avviene poiché d_i è stata diminuita; appare pertanto conveniente correggere quanto prima le etichette dei discendenti di i (sia pure senza ricalcolare immediatamente tutte le etichette del sottoalbero), in modo da evitare il più possibile che vengano compiute iterazioni con valori delle etichette che rappresentano una "cattiva" approssimazione del valore reale del costo del cammino.

La versione di *SPT.L* in cui Q è implementata come *deque* ha però complessità esponenziale $O(n2^n)$; esistono infatti grafi per i quali una tale versione effettua un numero esponenziale di inserimenti e rimozioni di nodi da Q (per ulteriori dettagli si rimanda alla letteratura citata). Comunque, l'analisi della complessità computazionale nel caso peggiore fornisce solo una misura di salvaguardia nella crescita del numero di operazioni: infatti, nei problemi reali in cui le reti sono abbastanza *sparse* ($m \approx n$) questa variante ha un comportamento molto buono, anzi spesso risulta il più efficiente algoritmo per i cammini minimi. Ciò è particolarmente vero per reti stradali, in cui si è osservato sperimentalmente che il numero medio di estrazioni di uno stesso nodo da Q è inferiore a 2. Un'alternativa all'uso della deque consiste nell'implementare anche Q' come fila; per questo basta mantenere un puntatore all'ultimo elemento della porzione Q' di Q ed effettuare gli inserimenti successivi al primo nella coda di Q' (cf. Figura 2.11). I nodi verranno così inseriti, la prima volta, in coda a Q" (che coincide con la coda di Q), le altre volte in coda a Q'. La struttura è conosciuta come *doppia coda* (o 2-queue). L'algoritmo risultante risulta sperimentalmente molto efficiente e, dal punto di vista teorico, ha complessità polinomiale: si può infatti dimostrare che il massimo numero di inserimenti dello stesso nodo in Q' è $O(n^2)$, e pertanto la complessità dell'algoritmo è $O(mn^2)$. Sono stati proposte altre implementazioni di Q basate su idee analoghe, ad esempio introducendo una "soglia" (*threshold*) opportunamente calcolata per decidere, in base al valore dell'etichetta d_i, se il nodo i sia da inserire in Q' oppure in Q''; per ulteriori dettagli si rinvia alla letteratura citata.

2.2.6 Cammini minimi su grafi aciclici

Un grafo orientato è detto *aciclico* se non contiene cicli orientati. È immediato verificare che un grafo orientato è aciclico se è *ben numerato*, ossia ha la proprietà che

$$(i, j) \in A \quad \Longrightarrow \quad i < j \ . \tag{2.10}$$

In effetti si può dimostrare che la condizione è sia necessaria che sufficiente: un grafo è aciclico se e solo se è possibile determinare una *buona numerazione* per i nodi, ossia una che soddisfi (2.10). Il problema di verificare se un dato grafo orientato sia aciclico e, in caso affermativo, di numerare i nodi del grafo in modo da soddisfare la proprietà (2.10), può essere risolto per mezzo di una visita del grafo (si veda il paragrafo B.3) ed ha pertanto complessità $O(m)$.

Esercizio 2.12. Scrivere una procedura che, in $O(m)$, controlli se un grafo orientato sia aciclico e, in caso positivo, ne numeri i nodi in modo che sia soddisfatta la proprietà (2.10) (suggerimento: se un grafo è aciclico, deve esistere almeno un nodo con stella entrante vuota; eliminando tale nodo e gli archi uscenti da esso, il sottografo indotto risulta a sua volta aciclico).

Nel seguito è descritta una procedura per il problema della determinazione dell'albero dei cammini minimi, di radice 1, su un grafo aciclico i cui nodi sono stati numerati in accordo alla (2.10); si lascia per esercizio la dimostrazione della sua correttezza e del fatto che la sua complessità è $O(m)$.

```
procedure ( p , d ) = SPT.Acyclic ( G , c )  {
  foreach( i ∈ N ) do { p[ i ] = 1; d[ i ] = M; }; d[ 1 ] = 0;
  for( i = 1 ; i < n ; i++ )
    foreach( ( i , j ) ∈ FS( i ) ) do
      if( d[ i ] + c_ij < d[ j ] ) then { d[ j ] = d[ i ] + c_ij; p[ j ] = i; }
  }
```

Procedura 2.2: Algoritmo *SPT.Acyclic*

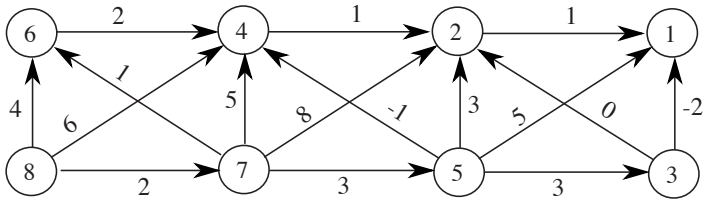
Esercizio 2.13. Applicare *SPT.Acyclic* al grafo di Figura 2.9 da cui siano stati eliminati gli archi $(3, 2)$ e $(6, 3)$, con radice $r = 1$, dopo aver eventualmente rinumerato i suoi nodi.

Esercizio 2.14. Discutere come mai nell'algoritmo *SPT.Acyclic* sia ragionevole assumere che la radice sia sempre il nodo 1.

Proponiamo infine un esercizio riassuntivo che illustra le differenze tra le varianti dell'algoritmo *SPT* su una stessa istanza.

Esempio 2.9. Confronto di diverse varianti di *SPT*

Si vuole individuare un albero dei cammini minimi di radice 8 sul grafo in figura, utilizzando le varianti *SPT.S*, *SPT.L.Queue*, *SPT.L.Stack* e *SPT.L.Deque* dell'algoritmo *SPT*. Si vuole inoltre determinare se il grafo è oppure no aciclico, ed in caso di risposta positiva risolvere il problema anche utilizzando *SPT.Acyclic*. Per tutte le varianti useremo $M = (n-1)\max\{ c_{ij} : (i, j) \in A \} + 1 = 7 \times 8 + 1 = 57$ come valore iniziale delle etichette, e descriveremo il funzionamento degli algoritmi mediante una tabella che riporta i valori del nodo selezionato i, di $p[\cdot]$, $d[\cdot]$ e Q (ove usata, ossia non in *SPT.Acyclic*) ad ogni iterazione, evidenziando per comodità in grassetto i valori dei vettori che cambiano da un'iterazione all'altra. Per le varianti che usano liste si esplorano gli archi della stella uscente del nodo selezionato in ordine crescente del nodo testa.

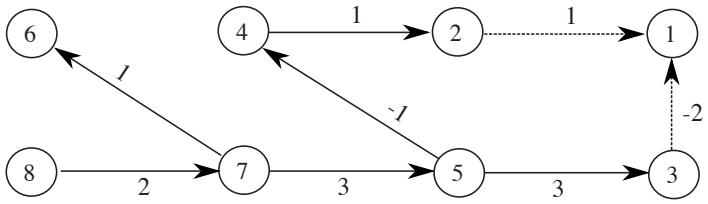


Iniziamo eseguendo *SPT.S*. Si noti che il grafo ha due archi di costo negativo, $(5, 4)$ e $(3, 1)$, quindi il Teorema 2.3 non si applica e non c'è garanzia che ogni nodo esca dalla coda una sola volta. Questo potrebbe in teoria portare la complessità dell'algoritmo ad essere esponenziale, ma ciò accade tipicamente solo su istanze costruite "ad hoc" e non sulle istanze reali, per le quali *SPT.S* in generale ha un buon comportamento computazionale nonostate la presenza di archi di costo negativo. In questo caso riportiamo Q ordinata per valore non decrescente dell'etichetta dei nodi.

		$p[\cdot]$								$d[\cdot]$								
it.	i	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	Q
0		8	8	8	8	8	8	8	8	57	57	57	57	57	57	57	0	$Q = \{8\}$
1	8	8	8	8	8	8	8	8	8	57	57	57	**6**	57	**4**	**2**	0	$Q = \{7, 6, 4\}$
2	7	8	**7**	8	8	**7**	**7**	8	8	57	**10**	57	6	**5**	**3**	2	0	$Q = \{6, 5, 4, 2\}$
3	6	8	7	8	**6**	7	7	8	8	57	10	57	**5**	5	3	2	0	$Q = \{5, 4, 2\}$
4	5	**5**	**5**	**5**	**5**	7	7	8	8	**10**	**8**	**8**	**4**	5	3	2	0	$Q = \{4, 2, 3, 1\}$
5	4	5	**4**	5	5	7	7	8	8	10	**5**	8	4	5	3	2	0	$Q = \{2, 3, 1\}$
6	2	**2**	4	5	5	7	7	8	8	**6**	5	8	4	5	3	2	0	$Q = \{1, 3\}$
7	1	2	4	5	5	7	7	8	8	6	5	8	4	5	3	2	0	$Q = \{3\}$
8	3	2	4	5	5	7	7	8	8	6	5	8	4	5	3	2	0	$Q = \emptyset$

Si noti che all'iterazione 4 ed all'iterazione 5 ci sono due nodi in Q con la stessa etichetta; in questo caso l'ordine di estrazione è arbitrario. Nonostante la presenza di archi a costo negativo, l'algoritmo esegue esattamente 8 iterazioni e ciascun nodo esce da Q esattamente una volta: il Teorema 2.3 fornisce condizioni sufficienti, ma non necessarie, affinché questo capiti. La presenza di archi a costo negativo non avrebbe però permesso di terminare l'algoritmo prima di aver visitato tutti i nodi nel caso si fosse stati interessati al cammino minimo solamente verso

un sottoinsieme di radici, poiché non ci sarebbe stata la garanzia a priori che la proprietà che lo permette fosse verificata (anche se a posteriori si verifica che la proprietà è comunque valida).
L'albero dei cammini minimi individuato è mostrato nella figura seguente, in cui sono tratteggiati gli archi $(2, 1)$ e $(3, 1)$. Questo è perchè esistono due alberi ottimi diversi, ciascuno dei quali contiene uno dei due (ma ovviamente non l'altro). In questa esecuzione è stato determinato quello che contiene $(2, 1)$ ($p[1] = 2$), ma avrebbe potuto essere determinato l'altro se l'algoritmo fosse stato diverso. Questo si vede dal fatto che l'arco $(3, 1)$ rispetta le condizioni di Bellman *all'uguaglianza* ($d_3 + c_{31} = 8 - 2 = d_1 = 6$) pur non facendo parte dell'albero selezionato. In effetti, nell'ultima iterazione l'algoritmo, visitando quell'albero, avrebbe anche potuto cambiare il predecessore di 1 (ma non ha ragione di farlo). È anche facile verificare le due indicate sono le uniche soluzioni ottime del problema. Infatti, tutti gli altri archi che non fanno parte dell'albero selezionato rispettano le condizioni di Bellman come *stretta disuguaglianza* ($d_i + c_{ij} > d_j$), e quindi causerebbero la crescita del valore della soluzione qualora fossero inseriti nell'albero (ammesso che ciò sia possibile senza creare un ciclo).



Passiamo adesso all'esecuzione di *SPT.L.Queue*; in questo caso e nei successivi l'ordine di estrazione da Q (cruciale per l'ordine di visita) dipende invece dall'ordine di inserzione, e quindi anche dall'ordine in cui si visitano le stelle uscenti.

		$p[\cdot]$								$d[\cdot]$								
it.	i	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	Q
0		8	8	8	8	8	8	8	8	57	57	57	57	57	57	57	0	$Q = \{8\}$
1	8	8	8	8	8	8	8	8	8	57	57	57	**6**	57	**4**	**2**	0	$Q = \{4, 6, 7\}$
2	4	8	**4**	8	8	8	8	8	8	57	**7**	57	6	57	4	2	0	$Q = \{6, 7, 2\}$
3	6	8	4	8	8	8	8	8	8	57	7	57	6	57	4	2	0	$Q = \{7, 2\}$
4	7	8	4	8	8	**7**	**7**	8	8	57	7	57	6	**5**	**3**	2	0	$Q = \{2, 5, 6\}$
5	2	**2**	4	8	8	7	7	8	8	**8**	7	57	6	5	3	2	0	$Q = \{5, 6, 1\}$
6	5	2	4	**5**	**5**	7	7	8	8	8	7	**8**	**4**	5	3	2	0	$Q = \{6, 1, 3, 4\}$
7	6	2	4	5	5	7	7	8	8	8	7	8	4	5	3	2	0	$Q = \{1, 3, 4\}$
8	1	2	4	5	5	7	7	8	8	8	7	8	4	5	3	2	0	$Q = \{3, 4\}$
9	3	**3**	4	5	5	7	7	8	8	**6**	7	8	4	5	3	2	0	$Q = \{4, 1\}$
10	4	3	4	5	5	7	7	8	8	6	**5**	8	4	5	3	2	0	$Q = \{1, 2\}$
11	1	3	4	5	5	7	7	8	8	6	5	8	4	5	3	2	0	$Q = \{2\}$
12	2	3	4	5	5	7	7	8	8	6	5	8	4	5	3	2	0	$Q = \emptyset$

In questo caso è stata determinata l'altra soluzione ottima, quella in cui è presente l'arco $(3, 1)$ invece di quello $(2, 1)$: $p[1] = 3$ invece di $p[1] = 2$, ma tutte le etichette sono le stesse. Si noti che l'algoritmo ha compiuto più iterazioni di *SPT.S*: guardando al comportamento al caso pessimo si sarebbe dovuto scegliere *SPT.L.Queue*, che ha complessità polinomiale anche nel caso di presenza di archi negativi, piuttosto che *SPT.S* che ha complessità esponenziale, ma le effettive prestazioni degli algoritmi non sono necessariamente ben caratterizzate dal caso pessimo. Anzi, *SPT.L.Queue*, che ha la migliore complessità al caso pessimo, di solito si rivela uno dei meno efficienti in pratica. Vediamo infatti adesso l'esecuzione di *SPT.L.Stack*, in cui Q è implementato come una *pila* (*stack*), ossia con la strategia LIFO (last in, first out). Ciò significa, nella nostra rappresentazione, che il nodo estratto è l'ultimo a destra invece che il primo a sinistra.

		$p[\cdot]$								$d[\cdot]$								
it.	i	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	Q
0		8	8	8	8	8	8	8	8	57	57	57	57	57	57	57	0	$Q = \{ 8 \}$
1	8	8	8	8	8	8	8	8	8	57	57	57	**6**	57	**4**	**2**	0	$Q = \{ 4, 6, 7 \}$
2	7	8	**7**	8	8	**7**	**7**	8	8	57	**10**	57	6	**5**	**3**	2	0	$Q = \{ 4, 6, 2, 5 \}$
3	5	**5**	**5**	**5**	**5**	7	7	8	8	**10**	**8**	**8**	**4**	5	3	2	0	$Q = \{ 4, 6, 2, 1, 3 \}$
4	3	**3**	5	5	5	7	7	8	8	**6**	8	8	4	5	3	2	0	$Q = \{ 4, 6, 2, 1 \}$
5	1	3	5	5	5	7	7	8	8	6	8	8	4	5	3	2	0	$Q = \{ 4, 6, 2 \}$
6	2	3	5	5	5	7	7	8	8	6	8	8	4	5	3	2	0	$Q = \{ 4, 6 \}$
7	6	3	5	5	5	7	7	8	8	6	8	8	4	5	3	2	0	$Q = \{ 4 \}$
8	4	3	**4**	5	5	7	7	8	8	6	**5**	8	4	5	3	2	0	$Q = \{ 2 \}$
9	2	3	4	5	5	7	7	8	8	6	5	8	4	5	3	2	0	$Q = \emptyset$

Il comportamento di *SPT.L.Stack* è diverso da quello di *SPT.L.Queue*, nonostante essi appaiano quasi identici. Parte della differenza è dovuta al fatto che i nodi appartenenti alla stella uscente vengono visitati "a rovescio" (in *SPT.L.Queue* nell'ordine in cui sono inseriti, ossia nel nostro caso nell'ordine di nodo testa, mentre in *SPT.L.Stack* in ordine inverso per via della strategia LIFO). Oltre a questo, però, l'uso di una pila fa si che appena si cambia l'etichetta di un nodo si dia priorità ad aggiornare quella di quelli da esso raggiungibili, potenzialmente evitando iterazioni in cui vengono usate etichette "sporche" e quindi lavoro inutile. Arriviamo adesso all'esecuzione di *SPT.L.Deqeue*. Si ricordi che in questo caso i nodi vengono inseriti a destra se hanno etichetta (prima del cambiamento M) ed a sinistra altrimenti, e vengono sempre estratti da sinistra.

		$p[\cdot]$								$d[\cdot]$								
it.	i	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	Q
0		8	8	8	8	8	8	8	8	57	57	57	57	57	57	57	0	$Q = \{ 8 \}$
1	8	8	8	8	8	8	8	8	8	57	57	57	**6**	57	**4**	**2**	0	$Q = \{ 4, 6, 7 \}$
2	4	8	**4**	8	8	8	8	8	8	57	**7**	57	6	57	4	2	0	$Q = \{ 6, 7, 2 \}$
3	6	8	4	8	8	8	8	8	8	57	7	57	6	57	4	2	0	$Q = \{ 7, 2 \}$
4	7	8	4	8	8	**7**	**7**	8	8	57	7	57	6	**5**	**3**	2	0	$Q = \{ 6, 2, 5 \}$
5	6	8	4	8	**6**	7	7	8	8	57	7	57	**5**	5	3	2	0	$Q = \{ 4, 2, 5 \}$
6	4	8	4	8	6	7	7	8	8	57	**6**	57	5	5	3	2	0	$Q = \{ 2, 5 \}$
7	2	**2**	4	8	6	7	7	8	8	**7**	6	57	5	5	3	2	0	$Q = \{ 5, 1 \}$
8	5	2	4	**5**	**5**	7	7	8	8	7	6	**8**	**4**	5	3	2	0	$Q = \{ 4, 1, 3 \}$
9	4	2	4	5	5	7	7	8	8	7	**5**	8	4	5	3	2	0	$Q = \{ 2, 1, 3 \}$
10	2	2	4	5	5	7	7	8	8	**6**	5	8	4	5	3	2	0	$Q = \{ 1, 3 \}$
11	1	2	4	5	5	7	7	8	8	6	5	8	4	5	3	2	0	$Q = \{ 3 \}$
12	3	2	4	5	5	7	7	8	8	6	5	8	4	5	3	2	0	$Q = \emptyset$

Esercizio 2.15. Si esegua sull'istanza data la variante *SPT.L.2-queue*.

Siamo infine a considerare l'uso di *SPT.Acyclic*. Per questo dobbiamo determinare se il grafo dato sia o meno aciclico. L'approccio che si può utilizzare per piccole istanze come questo è semplice: inizializzando $i = 1$, si cerca se esiste un nodo privo di archi entranti. Se ciò accade si da al nodo il valore di i, si cancella il nodo e tutti i suoi archi uscenti dal grafo, si incrementa i e si itera. Se il procedimento termina prima di aver cancellato tutto i nodi, e quindi avendo incontrato il caso in cui nessun nodo non ha archi entranti, allora il grafo non è aciclico. Se invece si sono cancellati tutti i nodi, quella così prodotta è una buona numerazione. Applicando il procedimento al grafo dato si determina che

originale	1	2	3	4	5	6	7	8
rinumerato	8	7	6	5	4	3	2	1

è una buona numerazione: infatti per ogni arco (i, j) del grafo rinumerato risulta $i < j$. In questo caso è quindi possibile utilizzare l'algoritmo *SPT.Acyclic* in cui è sufficiente esaminare i nodi in ordine della numerazione data, come mostra la seguente tabella (in cui i nomi dei nodi sono quelli originari)

		$p[\cdot]$								$d[\cdot]$							
it.	i	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
0		8	8	8	8	8	8	8	8	57	57	57	57	57	57	57	0
1	8	8	8	8	8	8	8	8	8	57	57	57	**6**	57	**4**	**2**	0
2	7	8	**7**	8	8	**7**	**7**	8	8	57	**10**	57	**6**	**5**	**3**	2	0
3	6	8	7	8	**6**	7	7	8	8	57	10	57	**5**	5	3	2	0
4	5	**5**	**5**	**5**	**5**	7	7	8	8	**10**	**8**	**8**	**4**	5	3	2	0
5	4	5	**4**	5	5	7	7	8	8	10	**5**	8	4	5	3	2	0
6	3	**3**	4	5	5	7	7	8	8	**6**	5	8	4	5	3	2	0
7	2	3	4	5	5	7	7	8	8	6	5	8	4	5	3	2	0

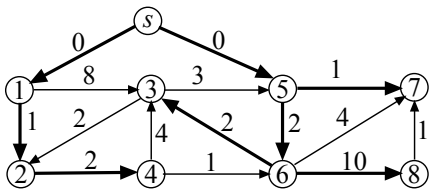
Questo algoritmo è chiaramente il più efficiente, non dovendo gestire nessuna Q e potendo anche fare un'iterazione in meno (visto che in un grafo aciclico il nodo n per definizione non può avere archi uscenti).

2.2.7 Cammini minimi con radici multiple

In alcuni casi è necessario risolvere il seguente problema: dato un grafo $G = (N, A)$ con costi sugli archi, e dato un insieme non vuoto R di nodi "radice", determinare per ogni nodo $i \notin R$ il cammino minimo da uno dei nodi $r \in R$ ad i. In altre parole si vuole determinare, per ogni nodo i, la "radice" dalla quale sia più conveniente raggiungerlo, ossia la radice alla quale corrisponda il cammino meno costoso fino ad i. È facile verificare che questo problema può essere risolto mediante un'applicazione della procedura *SPT* al grafo $G' = (N', A')$ in cui $N' = N \cup \{s\}$, dove s è un nodo fittizio che svolge il ruolo di "super-radice", $A' = A \cup \{(s, r) : r \in R\}$, e in cui i costi degli archi in $A' \setminus A$ (uscenti da s) sono nulli. L'albero dei cammini minimi su G' fornisce una soluzione ottima al problema dei cammini minimi con radici multiple. Si noti che l'albero così ottenuto, ristretto agli archi del grafo G originale, non è in generale più un albero ma una *foresta*, ossia una collezione di sottoalberi disgiunti, al più uno per ciascuna radice. Il sottoalbero relativo ad una certa radice r individua i nodi per i quali r è la radice più conveniente.

Esempio 2.10. Cammini minimi con radici multiple

Si vogliono determinare i cammini minimi rispetto all'insieme di radici $R = \{1, 5\}$ sul grafo di Figura 2.9. Qui accanto è mostrato il corrispondente grafo G' ed il relativo albero dei cammini minimi (archi in grassetto). Quindi, per i nodi 2 e 4 è conveniente selezionare il nodo 1 come radice, mentre per i nodi 3, 6, 7 e 8 è conveniente selezionare il nodo 5 come radice.



Come già visto per la visita (si veda il §B.3.2), è possibile risolvere il problema dell'albero dei cammini minimi di radici multiple senza costruire esplicitamente G'.

Esercizio 2.16. Si discuta come implementare l'algoritmo suggerito lavorando direttamente sul grafo originale G (suggerimento: si modifichi l'inizializzazione in modo opportuno).

Esercizio 2.17. Dato che il grafo G' è sicuramente aciclico se il grafo originale G lo è (s può essere numerato con un indice minore di tutti quelli dei nodi originali, ad esempio 0), si mostri che è possibile risolvere il problema dei cammini minimi con radici multiple su un grafo aciclico in $O(m)$ modificando opportunamente la procedura *SPT.Acyclic*.

2.3 Albero di copertura di costo minimo

Riprendiamo il problema dell'*albero di copertura di costo minimo* (MST), già presentato nell'Esempio 1.6: dato un grafo non orientato $G = (V, E)$ con costi c_{ij} associati ai lati, si vuole della determinare un albero di copertura $T = (V, E_T)$ tale che sia minimo il costo di T, definito come $c(T) = \sum_{\{i,j\} \in E_T} c_{ij}$. Questo problema *non* è un caso particolare del (MCF), in quanto non può essere formulato in forma naturale con le sole variabili di flusso.

Esercizio 2.18. Si dia una formulazione di *PLI* basata su variabili di flusso di (MST) (suggerimento: si parta dalla formulazione di (SPT) per un opportuno grafo *orientato* $G = (N, A)$ e si aggiungano

variabili binarie corrispondenti al fatto che un lato $\{ i , j \}$ è selezionato oppure no per far parte di A_T). Si fornisca poi un esempio che dimostra come, a differenza di (SPT), il rilassamento continuo di tale formulazione (ottenuto eliminando i vincoli di integralità sulle varibili) non sia una formulazione corretta dal problema originario.

Nonostante questo si ritiene utile studiare il problema, principalmente per due ragioni:

1. si tratta di uno dei pochi problemi di ottimizzazione significativi che può essere risolto all'ottimo per mezzo di un algoritmo di tipo *greedy* (cf. §5.1);
2. anche se è un problema diverso da (SPT) (anche perché è definito, a differenza di quello, su un grafo non orientato) alcuni degli algoritmi per la sua soluzione possono essere descritti in una forma sorprendentemente simile ad alcuni di quelli per (SPT).

La discussione del problema viene facilitata assumendo che G sia connesso e che tutti i costi siano positivi. Questo può essere fatto senza alcuna perdita di generalità. Infatti:

- Se G non è connesso non può esistere un albero di copertura, che per definizione è connesso. Ciò che esiste sono *foreste di copertura*, date dall'unione di un albero di copertura per ciascuna delle componenti connesse del grafo. Il problema di determinare la foresta di copertura di costo minimo si risolve banalmente individuando le componenti connesse (attraverso una procedura di visita $O(m)$) e poi facendo girare un algoritmo per (MST) separatamente su ognuna. Alternativamente, in modo analogo a quanto visto per (SPT) si può rendere il grafo connesso selezionando un qualsiasi nodo $i \in V$ e collegandolo con tutti i nodi $j \in V$ per cui non esista già $\{ i , j \} \in A$ attraverso un nuovo lato fittizio $\{ i , j \}$ di costo opportunamente grande $M = c_{max} + 1$. È facile mostrare che la soluzione T^* di (MST) sul grafo così ottenuto è la foresta di copertura di costo minimo completata con lati da i ad un nodo j per ciascuna delle componenti non connesse con i. Infatti, chiaramente nessun lato fittizio $\{ i , j \}$ può far parte della soluzione ottima a meno che j sia non connesso ad i: se esistesse un cammino P_{ij} (non orientato) su G esso costerebbe meno del lato fittizio $\{ i , j \}$, per cui sostituendo P_{ij} ad $\{ i , j \}$ in T^* si otterrebbe un nuovo sottografo connesso di costo minore, contraddicendo l'ottimalità di T^*. Ragionando in modo analogo si dimostra che non possono far parte della soluzione ottima due lati fittizi $\{ i , j \}$ ed $\{ i , h \}$ se j e h appartengono alla stessa componente connessa.
- La soluzione ottima di (MST) non cambia se il costo di tutti i lati viene modificato della stessa costante, ossia diviene $c'_{ij} = c_{ij} + \delta$ per un qualsiasi $\delta \in \mathbb{R}$. Ciò dipende dal fatto che ogni soluzione ammissibile di (MST) (se il grafo è connesso, ma questo punto è già stato trattato) contiene esattamente $n - 1$ lati, e di conseguenza il costo di tutte le soluzioni ammissibili si modifica della stessa costante $(n-1)\delta$; pertanto, qualsiasi soluzione T^* ottima per i costi originali c_{ij} è anche ottima per quelli modificati c'_{ij}, come è immediato verificare. Se quindi $C = \min\{ c_{ij} : \{ i , j \} \in E \} < 0$, è sufficiente modificare i costi aggiungendo a tutti la costante $\delta = -C + 1$ per ottenere una nuova istanza di (MST), equivalente alla precedente, in cui tutti i costi sono positivi.

Presenteremo adesso uno schema algoritmico molto generale, *Greedy-MST*, per la risoluzione di (MST). Tale algoritmo costruisce l'albero incrementalmente, mantenendo ad ogni passo due insiemi di lati: S, l'insieme dei lati già inseriti nell'albero, e R, l'insieme dei lati scartati, cioè l'insieme dei lati che certamente non verranno inseriti nell'albero. S e R, che all'inizio sono vuoti, vengono aggiornati per mezzo delle seguenti operazioni:

- *Inserzione*: seleziona in G un taglio (V' , V'') tale che $S \cap A(V' , V'') = \emptyset$, ed aggiungi ad S un lato $\{ u , v \}$ per cui $c_{uv} = \min\{ c_{ij} : \{ i , j \} \in A(V' , V'') \setminus R \}$.
- *Cancellazione*: seleziona in G un ciclo C tale che $C \cap R = \emptyset$, e aggiungi a R un lato $\{ u , v \} \in C \setminus S$ per cui $c_{uv} = \max\{ c_{ij} : \{ i , j \} \in C \setminus S \}$.

```
procedure S = Greedy-MST( G , c ) {
  S = ∅; R = ∅;
  do applica Inserzione o Cancellazione
  while( S ∪ R ⊊ E and |S| < n − 1; )
  }
```

Procedura 2.3: Algoritmo *greedy* per il problema (MST)

L'algoritmo termina quando nessuna delle due operazioni è più applicabile, cioè quando risulta $S \cup R = E$, oppure quando sono stati inseriti $n-1$ lati in S, cioè quando il grafo parziale definito da S è un albero. Quando l'algoritmo termina, S definisce una soluzione ottima del problema; vale infatti la seguente proprietà:

Lemma 2.2. Se la coppia di insiemi (S, R) è tale per cui esiste in G un albero di copertura di costo minimo $T = (V, E_T)$ con $S \subseteq E_T$ e $R \cap E_T = \emptyset$, allora l'applicazione di una qualsiasi delle operazioni di *inserzione* o di *cancellazione* produce una nuova coppia (S, R) che gode della stessa proprietà.

Dimostrazione. Cominciamo col dimostrare che la tesi vale per l'operazione di *inserzione*. Sia T un albero ottimo tale che E_T contiene S e $R \cap E_T = \emptyset$; siano (V', V'') e $\{u, v\}$, rispettivamente, il taglio ed il lato selezionati dall'operazione di inserzione. Se $\{u, v\} \in E_T$ allora la proprietà è soddisfatta. Altrimenti, essendo T connesso, esiste almeno un lato $\{k, l\}$ dell'unico cammino che connette u e v in T che appartenga ad $A(V', V'')$. Sostituendo $\{u, v\}$ a $\{k, l\}$ in E_T si ottiene un nuovo albero di copertura T'; essendo $c_{uv} \leq c_{kl}$, si ha $c(T') \leq c(T)$. Siccome per ipotesi T è ottimo, anche T' lo è (e quindi, $c_{uv} = c_{kl}$); T' contiene il lato selezionato $\{u, v\}$, e quindi la proprietà è soddisfatta da T' (anche se non da T).
Consideriamo ora l'operazione di *cancellazione*. Sia T un albero ottimo per cui $E_T \cap R = \emptyset$, e siano C e $\{u, v\}$, rispettivamente, il ciclo e l'arco selezionati dall'operazione di cancellazione. Se T non contiene $\{u, v\}$, allora la proprietà è soddisfatta. Altrimenti, cancellando $\{u, v\}$ dall'albero si ottengono due sottoalberi $T' = (V', E'_T)$ e $T'' = (V'', E''_T)$; tra gli archi del ciclo C deve necessariamente esistere un arco $\{k, l\} \notin E_T$ che appartenga a $A(V', V'')$, con $c_{kl} \leq c_{uv}$. Sostituendo $\{u, v\}$ con $\{k, l\}$ in T si ottiene un nuovo albero, anch'esso ottimo, che soddisfa la proprietà (si ha infatti $c_{kl} = c_{uv}$). □

Dal lemma precedente si ricava immediatamente per induzione che l'algoritmo *Greedy-MST* termina fornendo un albero di copertura di costo minimo: basta osservare che all'inizio S e R sono vuoti, e quindi godono banalmente della proprietà richiesta. Si noti che la correttezza dell'algoritmo non dipende dall'ordine con cui vengono realizzate le operazioni di *inserzione* e *cancellazione*. In effetti, esistono diverse possibili implementazioni dell'algoritmo *Greedy-MST* che si distinguono per l'ordine con cui vengono effettuate tali operazioni: nel seguito presenteremo due di tali algoritmi.

2.3.1 Algoritmo di Kruskal

In questo algoritmo, i lati del grafo vengono inizialmente ordinati in ordine di costo non decrescente. Seguendo tale ordinamento, ad ogni iterazione viene selezionato il primo arco $\{u, v\}$ non ancora esaminato: se $\{u, v\}$ non forma alcun ciclo con gli archi in S, allora esso viene inserito in S, cioè si applica l'operazione di *inserzione*, altrimenti l'arco viene inserito in R, cioè si applica l'operazione di *cancellazione*. Le due operazioni sono applicabili legittimamente. Infatti, nel primo caso, la non esistenza di cicli nel grafo parziale (V, S) garantisce l'esistenza di un taglio (V', V'') con $u \in V'$ e $v \in V''$, tale che $\{\{i, j\} : i \in V', j \in V''\} \cap S = \emptyset$; inoltre, poiché i lati non ancora selezionati hanno un costo non minore di c_{uv}, è vero che $\{u, v\}$ è un lato di costo minimo fra quelli del taglio. Nel secondo caso, tutti i lati del ciclo C appartengono ad S tranne $\{u, v\}$: quindi $\{u, v\}$ è il lato di costo massimo tra quelli in $C \setminus S$, essendo l'unico. Si noti comunque che $\{u, v\}$, essendo stato selezionato dopo tutti gli altri lati di C, è anche il lato di costo massimo fra tutti quelli del ciclo. Lo pseudo-codice dell'algoritmo è il seguente, dove la funzione *Sort*(E) restituisce l'insieme E ordinato per costo non decrescente, e la funzione *Component*(E , u , v) risponde *true* se u e v appartengono alla stessa componente connessa del grafo parziale definito da S, cioè se $\{u, v\}$ induce un ciclo su tale grafo, e *false* altrimenti.

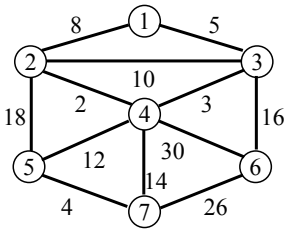
```
procedure S = Kruskal( G , c )  {
  S = ∅; R = ∅; X = Sort( A );
  do {  estrai da X il primo lato { u , v };
        if( Component( E , u , v ) )
          then R = R ∪ { { u , v } };/* cancellazione */
          else S = S ∪ { { u , v } };  /* inserzione */
        } while( |S| < n − 1 );
  }
```

Procedura 2.4: Algoritmo di Kruskal per il problema (MST)

L'operazione più critica dell'algoritmo è il controllo di esistenza di un ciclo comprendente il lato $\{u, v\}$, cioè se due nodi u e v appartengono alla stessa componente connessa o meno. È possibile predisporre delle opportune strutture di dati che permettano di effettuare in modo efficiente questo controllo. Con l'ausilio di tali strutture di dati si può ottenere una complessità computazionale pari a $O(m \log n)$, data dal costo dell'ordinamento degli archi.

Esempio 2.11. Esecuzione dell'algoritmo di Kruskal

Applichiamo ad l'algoritmo di Kruskal al grafo in figura qui accanto; in Figura 2.12 sono riportati i passi effettuati, riportando i lati inseriti in S ed il costo $c(S)$. L'ordinamento iniziale fornisce $X = \{\{2,4\},\{3,4\},\{5,7\},\{1,3\},\{1,2\},\{2,3\},\{4,5\},\{4,7\},\{3,6\},\{2,5\},\{6,7\},\{4,6\}\}$. Si noti che il quinto lato inserito, $\{4, 5\}$, ha costo $c_{45} = 12$; tale inserimento avviene dopo la cancellazione dei lati $\{1, 2\}$ e $\{2, 3\}$ in quanto $\{1, 2\}$ forma ciclo con $\{2, 4\}$, $\{3, 4\}$ e $\{1, 3\}$, mentre $\{2, 3\}$ forma ciclo con $\{2, 4\}$ e $\{3, 4\}$. Analogamente, prima dell'inserzione di $\{3, 6\}$ vi è stata la cancellazione di $\{4, 7\}$ in quanto forma ciclo con $\{5, 7\}$ e $\{4, 5\}$. Con l'inserzione di $\{3, 6\}$ il grafo parziale definito da S diviene connesso ($|S| = 6$), e quindi un albero di copertura di costo minimo; gli ultimi tre archi nell'insieme X non vengono esaminati.



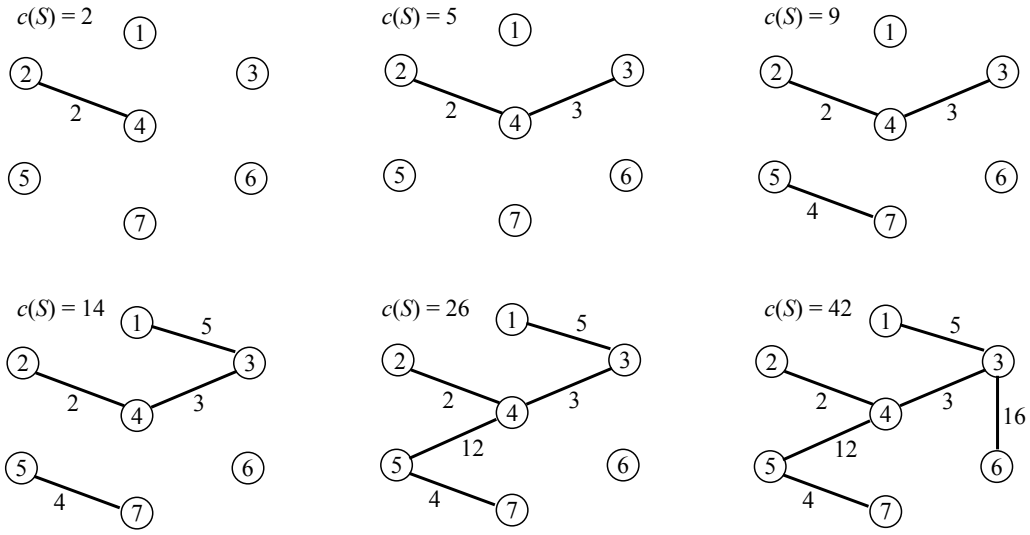


Figura 2.12: Passi effettuati dall'algoritmo di Kruskal

Esercizio 2.19. Applicare *Kruskal* al grafo di figura 2.13; fornire ad ogni iterazione la foresta $T = (V, S)$ ed il lato esaminato indicando se viene eliminato o inserito e perché.

Esercizio 2.20. Se il grafo G non è connesso, non esiste un albero di copertura per G; esiste però una foresta di alberi di copertura, e quindi alberi di copertura di costo minimo, per ciascuna delle componenti connesse. Si discuta come modificare l'algoritmo di Kruskal, senza aumentarne la complessità, in modo tale che determini una foresta di alberi di copertura di costo minimo per tutte le componenti connesse di G.

2.3.2 Algoritmo di Prim

L'*algoritmo di Prim* effettua solamente l'operazione di *inserzione*, mentre l'operazione di *cancellazione* viene effettuata implicitamente. Per effettuare l'operazione di inserzione, viene costruito ed aggiornato

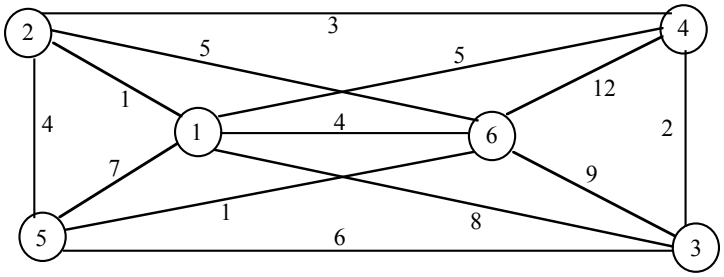


Figura 2.13: Un'istanza di (MST)

ad ogni iterazione un taglio $arcV'V''$ con la caratteristica che (V', S) è un albero di copertura di costo minimo per il grafo indotto da V'. Ad ogni iterazione viene selezionato ed inserito in S un lato a costo minimo fra quelli che hanno un estremo in V' e l'altro in V'', cioè un lato appartenente all'insieme $A(V', V'')$ degli archi del taglio. L'insieme V' viene inizializzato con un nodo arbitrario r: $V' = \{r\}$ e $V'' = V \setminus \{r\}$, pertanto $S = \emptyset$. Come precedentemente discusso introduciamo un lato fittizio $\{r, i\}$, per ogni $i \in V \setminus \{r\}$, di costo opportunamente grande $c_{ri} = M$. Per memorizzare la porzione di albero corrente definita da (V', S), e alla fine l'albero di copertura T, utilizziamo un vettore di predecessori $p[\cdot]$.

Per effettuare l'operazione di inserzione si deve determinare un lato di costo minimo appartenente all'insieme corrente $A(V', V'')$. Al fine di rendere efficiente questa operazione, che in generale avrebbe complessità in tempo $O(m)$, memorizziamo, per ogni vertice $j \in V''$, sia il lato $\{i, j\} \in S(i) \cap A(V', V")$ di costo minimo, utilizzando il vettore di predecessori ($p[j] = i$) che il suo costo, utilizzando un vettore di etichette $d[\cdot]$ ($d[j] = c_{ij}$). In tal modo è possibile determinare, in tempo $O(n)$, il lato di costo minimo in $A(V', V'')$ selezionando un vertice u di etichetta minima tra i vertici in V'', che viene inserito in V'. Lo spostamento di u da V'' a V' equivale all'inserimento di $\{p[u], u\}$ in S. Si noti che, per aggiornare i valori $p[j]$ e $d[j]$ per ogni nodo $j \in V''$, è sufficiente esaminare ciascun lato $\{u, v\} \in S(u)$ tale che $v \in V''$ e verificare se esso non sia il lato del taglio incidente in v di costo minimo. Infatti, basta scegliere il "lato di costo minore" tra $\{u, v\}$ e $\{p[v], v\}$, cioè basta controllare se $c_{uv} < d_v$. In caso affermativo $\{u, v\}$ risulta migliore di $\{p[v], v\}$ e viene sostituito ad esso ponendo $p[v] = u$ e $d_v = c_{uv}$; altrimenti non si effettua nessun cambiamento. Si noti che, nel primo caso, si ha un'operazione di cancellazione implicita del lato $\{p[v], v\}$, mentre nel secondo viene cancellato il lato $\{u, v\}$. Per ragioni implementative, viene utilizzato un insieme Q dei vertici candidati che contenga tutti e soli i vertici di V'' per i quali esista almeno un lato del taglio incidente in essi; in tal modo la ricerca del vertice di etichetta minima viene effettuata in Q. Si noti che i vertici $j \in V'' \setminus Q$ sono tutti e soli i vertici con etichetta arbitrariamente grande ($d[j] = M$).

```
procedure p = Prim( G , c , r )  {
  foreach( i ∈ V ) do { p[i] = r; d[i] = M; }
  d[r] = −M; Q = {r};
  do {  seleziona u in Q tale che d[u] = min{ d[j] : j ∈ Q };
        d[u] = −M; Q = Q \ {u};
        foreach( {u, v} ∈ S(u) ) do
          if( c_uv < d[v] ) then {   d[v] = c_uv; p[v] = u;
                                     if( v ∉ Q ) then Q = Q ∪ {v};
                                     }
        } while( Q ≠ ∅ );
  }
```

Procedura 2.5: Algoritmo di *Prim*

Lo pseudo-codice evidenzia la forte somiglianza con l'algoritmo *SPT.S* descritto nel paragrafo 2.2.3. In esso viene modificata la condizione di scelta dell'arco (lato): al posto della condizione di Bellman si inserisce la condizione di "lato di costo minore". Inoltre, per indicare che un vertice i appartiene all'insieme V', e quindi che non devono essere più effettuate inserzioni di lati incidenti in esso, si pone $d[j] = -M$.

È facile verificare che, nel caso in cui Q sia implementata come una lista non ordinata, la complessità dell'algoritmo di Prim è $O(n^2)$, come quella del suo corrispondente per (SPT). Infatti, anche in questo caso non si avranno più di n estrazioni di vertici da Q, e ogni estrazione ha costo $O(n)$ per la scansione di Q per determinare il nodo di etichetta minima. Le operazioni relative ai lati hanno complessità costante e saranno ripetute al più due volte per ciascun lato: quindi, globalmente costano $O(m) = O(n^2)$ poiché $m < n^2$. Se Q è implementato come uno *Heap binario bilanciato*, invece, la complessità è $O(m \log n)$ (si veda il §2.2.4).

Esempio 2.12. Esecuzione dell'algoritmo di Prim

In Figura 2.14 sono rappresentate le soluzioni al termine delle iterazioni dell'algoritmo di Prim applicato al grafo dell'Esempio 2.11. I lati evidenziati sono quelli inseriti in S, e sono incidenti in vertici i tali che $p[\,i\,]$ è fissato e $d[\,i\,] = -M = -31$, mentre i lati tratteggiati sono quelli $\{\,p[\,v\,]\,,\,v\,\}$ candidati ad essere inseriti in S, per ogni $v \in Q$ (accanto al nodo è riportata la sua etichetta). La linea tratteggiata indica il taglio $(\,V'\,,\,V''\,)$. I vertici evidenziati sono quelli di etichetta minima che verranno estratti da Q all'iterazione successiva. La sequenza dei valori delle etichette dei nodi rimossi da Q è 0, 5, 3, 2, 12, 4, 16. Ad ogni iterazione viene riportato il costo $c(S)$ della soluzione corrente; al termine $c(S) = 42$, cioè la somma delle etichette dei nodi al momento della loro estrazione da Q; infatti, esse rappresentano i costi dei lati che formano l'albero.

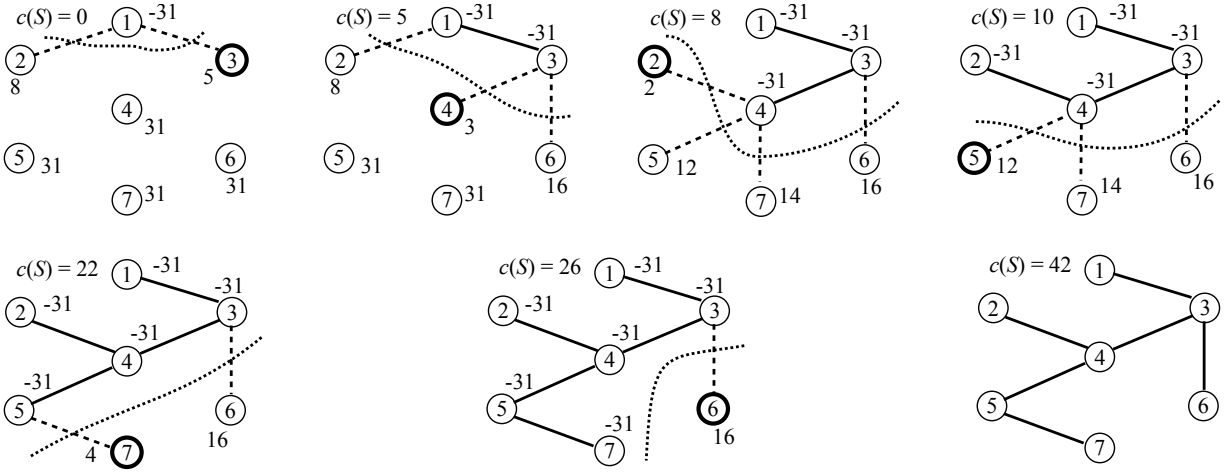


Figura 2.14: Passi effettuati dall'algoritmo di Prim

Esercizio 2.21. Applicare *Prim* al grafo di Figura 2.13; fornire ad ogni iterazione l'albero parziale $T = (\,V'\,,\,E_T\,)$, l'insieme Q ed il vertice u selezionato.

Esercizio 2.22. Si discuta come modificare l'algoritmo di Prim, senza aumentarne la complessità, in modo tale che, anche se applicato ad un grafo G non è connesso, determini alberi di copertura di costo minimo per tutte le componenti connesse di G.

2.3.3 Albero di copertura bottleneck

Un diverso problema di albero ottimo, strettamente correlato con (MST), è quello in cui si richiede di minimizzare non la somma dei costi dei lati appartenenti all'albero di copertura $T = (\,V\,,\,E_T\,)$ ma il suo *valore bottleneck* (collo di bottiglia) $V(T) = \max\{\,c_{ij} \;:\; \{\in, E\}_T\,\}$, ossia il massimo costo degli archi utilizzati. Vale il seguente lemma:

Lemma 2.3. Un albero di copertura di costo minimo è anche un albero bottleneck.

Dimostrazione. Sia $T' = (\,V\,,\,E_{T'}\,)$ un albero di copertura di costo minimo e sia $\{\,u\,,\,v\,\}$ un suo lato di costo massimo. La rimozione di $\{\,u\,,\,v\,\}$ da T' genera un taglio $(\,V'\,,\,V''\,)$ del grafo; per l'ottimalità di T' non esistono lati del taglio di costo inferiore a c_{uv} e quindi il valore $V(T)$ di qualsiasi albero bottleneck T è tale che $V(T) \geq c_{uv}$. Da $V(T') = c_{uv}$ segue che T' è un albero bottleneck. □

Si noti che, in generale, non è vero il viceversa; infatti, dato un albero di copertura di costo minimo (e quindi anche bottleneck) T', di valore bottleneck $V(T')$, se si sostituisce un qualsiasi lato $\{\,i\,,\,j\,\}$ con un lato $\{\,p\,,\,q\,\}$ appartenente al taglio indotto da $\{\,i\,,\,j\,\}$, purché $c_{ij} < c_{pq} \leq V(T')$, si ottiene un

nuovo albero T, che è bottleneck ma non più di costo minimo. Abbiamo pertanto mostrato che i due problemi non sono equivalenti, ma che l'insieme delle soluzioni del problema dell'albero bottleneck contiene l'insieme delle soluzioni del (MST); pertanto per risolvere il primo problema è sufficiente risolvere il secondo.

2.4 Il problema di flusso massimo

Dato il grafo orientato $G = (N, A)$, il vettore $u = [u_{ij}]_{(i,j)\in A}$ di capacità superiori degli archi, e due nodi distinti s e t, detti rispettivamente *origine* (o *sorgente*) e *destinazione* (o *pozzo*), il *problema del flusso massimo* ((MF), da Max Flow problem) consiste nel determinare la massima quantità di flusso che è possibile inviare da s a t attraverso G. Più precisamente, si vuole determinare il massimo valore v per cui ponendo $b_s = -v$, $b_t = v$ e $b_i = 0$ per ogni $i \notin \{s, t\}$ esiste un flusso ammissibile x. Questo significa massimizzare il *valore v del flusso x* che soddisfa

$$\sum_{(j,i)\in BS(i)} x_{ji} - \sum_{(i,j)\in FS(i)} x_{ij} = \begin{cases} -v & i = s \\ v & i = t \\ 0 & \text{altrimenti} \end{cases} \qquad i \in N \ , \tag{2.11}$$

oltre, ovviamente, ai vincoli di capacità (2.3). Il problema di flusso massimo è in realtà un caso particolare del problema di flusso di costo minimo. Infatti, la formulazione (2.11) può essere vista come corrispondente ad un problema (MCF) su un grafo G' ottenuto da G aggiungendo un arco fittizio (t, s), detto *arco di ritorno*, il cui flusso x_{ts} è proprio il valore v: la colonna dei coefficienti relativa alla variabile v, interpretata come una colonna della matrice di incidenza di G', individua proprio l'arco (t, s), come mostrato anche in Figura 2.15.

In G' i nodi sono di trasferimento, compresi s e t, cioè $b = 0$; un tale problema di flusso è detto *di circolazione*. I costi degli archi sono nulli salvo quello dell'arco (t, s) che è posto uguale a -1: di conseguenza, minimizzare $-v$ equivale a massimizzare il valore v del flusso che transita lungo l'arco (t, s), ossia del flusso che in G è inviato da s a t.

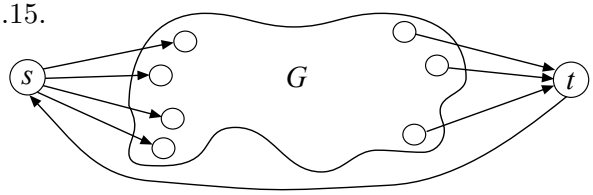


Figura 2.15: Un grafo con l'arco di ritorno (t, s)

Esempio 2.13. Un esempio di flusso ammissibile
Si consideri il grafo in Figura 2.16, in cui la sorgente è $s = 1$ ed il pozzo è $t = 6$. Il vettore x definito sugli archi e riportato in figura è un flusso ammissibile, in quanto sono verificati sia i vincoli di conservazione di flusso che i vincoli di capacità; in particolare, nei nodi 2 e 5 entrano ed escono 7 unità di flusso, mentre nei nodi 3 e 4 ne entrano ed escono 4. Il valore del flusso è $v = 8$, pari alle unità di flusso uscenti da 1 e, equivalentemente, da quelle entranti in 6; se si aggiungesse l'arco di ritorno $(6, 1)$ il suo flusso sarebbe proprio 8. Si noti che non si hanno archi *vuoti*, cioè archi con flusso nullo, mentre gli archi $(1, 2)$, $(3, 5)$ e $(4, 6)$ sono *saturi*, cioè sono archi il cui flusso è uguale alla capacità superiore.

Rispetto al problema di Flusso di Costo Minimo generale, il problema di flusso massimo presenta due importanti caratteristiche: il vettore dei bilanci è nullo, ed il vettore dei costi è nullo tranne che in corrispondenza all'arco fittizio (t, s), in cui è negativo. Come vedremo, queste caratteristiche permettono di sviluppare algoritmi specifici molto efficienti per il problema.

2.4.1 Tagli, cammini aumentanti e condizioni di ottimo

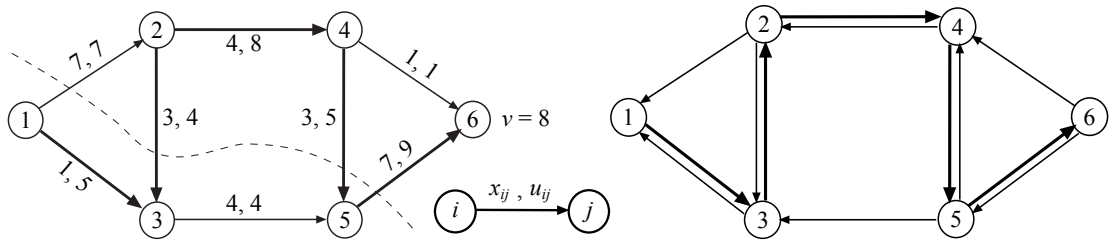


Figura 2.16: un flusso ammissibile x di valore $v = 8$ ed il corrispondente grafo residuo

Come nel caso del problema dei cammini minimi, consideriamo un flusso ammissibile x di valore v, e poniamoci il problema di determinare se x sia oppure no ottimo. Se x non è ottimo, ciò significa che è possibile inviare altro flusso dall'origine alla destinazione; è intuitivo pensare che questo flusso possa essere "instradato" lungo un cammino da s a t. Sia quindi P un cammino, non necessariamente orientato, da s a t: gli archi di P possono essere partizionati nei due insiemi P^+ e P^-, detti *insieme degli archi concordi* ed *insieme degli archi discordi* di P, che contengono rispettivamente gli archi che, andando da s a t, vengono attraversati nel verso del loro orientamento e gli archi attraversati nel verso opposto a quello del loro orientamento. Il cammino P può essere utilizzato per inviare ulteriore flusso da s a t se è possibile modificare il valore del flusso su tutti i suoi archi senza perdere l'ammissibilità ed aumentando il valore v del flusso corrente. È immediato verificare che l'unico modo in cui si può modificare il valore del flusso sugli archi del cammino senza violare i vincoli di conservazione del flusso nei nodi intermedi è aumentare il flusso lungo tutti gli archi concordi, e diminuire il flusso lungo tutti gli archi discordi, di una stessa quantità θ. In altre parole, se x rispetta i vincoli di conservazione del flusso, allora anche $x(\theta) = x \oplus \theta P$ con

$$x_{ij}(\theta) = \begin{cases} x_{ij} + \theta & \text{se } (i, j) \in P^+ \\ x_{ij} - \theta & \text{se } (i, j) \in P^- \\ x_{ij} & \text{altrimenti} \end{cases} \tag{2.12}$$

li rispetta per qualsiasi valore di θ, ed il valore di $x(\theta)$ è $v+\theta$. Tale *operazione di composizione* $\oplus$ tra il flusso x ed il cammino P corrisponde quindi all'invio di θ unità di flusso dall'origine alla destinazione utilizzando il cammino P. Non per tutti i valori di θ, però, l'operazione di composizione produce un flusso ammissibile, in quanto i vincoli (2.3) potrebbero essere violati. La quantità

$$\theta(P, x) = \min\left\{ \min\{ u_{ij} - x_{ij} : (i, j) \in P^+ \}, \min\{ x_{ij} : (i, j) \in P^- \} \right\} \ (\geq 0) \ , \tag{2.13}$$

detta *capacità del cammino P rispetto al flusso x*, rappresenta la massima quantità di flusso che, aggiunta agli archi concordi di P, non produce flussi maggiori delle rispettive capacità, e sottratta agli archi discordi di P, non produce flussi negativi. Si noti che può essere $\theta(P, x) = 0$: ciò accade se e solo se almeno uno degli archi concordi è *saturo* oppure almeno uno degli archi discordi è *vuoto*. Se invece $\theta(P, x) > 0$, P è detto un *cammino aumentante*, cioè è un cammino lungo il quale può essere inviata una quantità positiva di flusso da s verso t.

Esempio 2.14. Cammini aumentanti
Sia dato il grafo in Figura 2.16, e si consideri il cammino (non orientato) $P = \{1, 3, 2, 4, 5, 6\}$, anch'esso mostrato in Figura (archi evidenziati). L'insieme degli archi concordi è $P^+ = \{(1, 3), (2, 4), (4, 5), (5, 6)\}$ mentre l'insieme degli archi discordi è $P^- = \{(2, 3)\}$. La capacità di P rispetto a x è $\theta(P, x) = \min\{\min\{5 - 1, 8 - 4, 5 - 3, 9 - 7\}, \min\{3\}\} = \min\{2, 3\} = 2 > 0$, e pertanto P è un cammino aumentante rispetto a x: infatti, nessun arco concorde è saturo e nessun arco discorde è vuoto. Possiamo quindi utilizziare P inviare altre due unità di flusso da s a t, ossia costruire il nuovo flusso $x' = x \oplus 2P$; applicando la definizione (2.12) si ottiene $x'_{13} = 1 + 2 = 3$, $x'_{23} = 3 - 2 = 1$, $x'_{24} = 4 + 2 = 6$, $x'_{45} = 3 + 2 = 5$ e $x'_{56} = 7 + 2 = 9$, mentre il flusso su tutti gli altri archi è invariato. È immediato verificare che x' è un flusso ammissibile di valore $v' = 8 + 2 = 10$.

Si pone quindi il problema di definire un algoritmo che, dato un grafo G e un flusso ammissibile x, determini (se esiste) un cammino aumentante P rispetto ad x. A questo scopo si introduce il *grafo residuo* $G_x = (N, A_x)$ rispetto al flusso x, dove

$$A_x = A_x^+ \cup A_x^- = \{ (i, j) : (i, j) \in A, \ x_{ij} < u_{ij} \} \cup \{ (i, j) : (j, i) \in A, \ x_{ji} > 0 \} \ .$$

Il grafo residuo, cioè, contiene al più due "rappresentanti" di ciascun arco (i, j) del grafo originale: uno, orientato come l'arco originario, se (i, j) non è saturo e quindi può appartenere all'insieme degli archi *concordi* di un cammino aumentante, mentre l'altro, orientato in modo opposto, se (i, j) non è vuoto, e quindi può appartenere all'insieme degli archi *discordi* di un cammino aumentante. È immediato verificare che G_x permette di ricondurre il concetto di cicli e cammini aumentanti al più usuale concetto di cicli e cammini orientati:

Lemma 2.4. Per ogni cammino aumentante da s a t rispetto ad x in G esiste uno ed un solo cammino orientato da s a t in G_x.

Esempio 2.15. Grafo residuo
In Figura 2.16, a destra, è mostrato il grafo residuo. Il cammino orientato $P = \{1, 3, 2, 4, 5, 6\}$ di G_x corrisponde al cammino aumentante di G mostrato nell'esempio precedente.

Un cammino aumentante, se esiste, può quindi essere determinato mediante una visita del grafo residuo G_x a partire da s. Se la visita raggiunge t, allora si è determinato un cammino aumentante (si noti che la visita può essere interrotta non appena questo accada), ed il flusso x non è ottimo perché il cammino aumentante permette di ottenere un nuovo flusso x' di valore strettamente maggiore. Se invece al termine della visita non si è visitato t, allora x è un flusso massimo. Per dimostrarlo introduciamo alcuni ulteriori concetti.

Indichiamo con (N_s, N_t) un taglio di G che separa s da t, cioè un taglio per cui sia $s \in N_s$ e $t \in N_t$, ed indichiamo con $A^+(N_s, N_t)$ ed $A^-(N_s, N_t)$, rispettivamente, l'insieme degli archi diretti e quello degli archi inversi del taglio (si veda l'Appendice B). Dato un flusso x, per ogni taglio (N_s, N_t) definiamo il *flusso del taglio* $x(N_s, N_t)$ e la *capacità del taglio* $u(N_s, N_t)$ come segue:

$$x(N_s, N_t) = \sum_{(i,j)\in A^+(N_s,N_t)} x_{ij} - \sum_{(i,j)\in A^-(N_s,N_t)} x_{ij} , \tag{2.14}$$

$$u(N_s, N_t) = \sum_{(i,j)\in A^+(N_s,N_t)} u_{ij} . \tag{2.15}$$

Il flusso del taglio è la quantità di flusso che attraversa il taglio (N_s, N_t) da s verso t. Il seguente teorema fornisce la relazione esistente tra il valore del flusso x, ed i flussi e le capacità dei tagli di G.

Teorema 2.5. Per ogni flusso ammissibile x di valore v e per ogni taglio (N_s, N_t) vale

$$v = x(N_s, N_t) \le u(N_s, N_t) .$$

Dimostrazione La disuguaglianza deriva da (2.14), (2.15) e dal fatto che $0 \le x_{ij} \le u_{ij}$ per ogni $(i, j) \in A$; infatti

$$\sum_{(i,j)\in A^+(N_s,N_t)} x_{ij} \le \sum_{(i,j)\in A^+(N_s,N_t)} u_{ij} \qquad \text{e} \qquad -\sum_{(i,j)\in A^-(N_s,N_t)} x_{ij} \le 0 .$$

L'uguaglianza si ottiene sommando tra loro, membro a membro, i vincoli di (2.11) corrispondenti ai nodi in N_t; infatti, nella sommatoria i flussi degli archi $(i, j) \in A^+(N_s, N_t)$ appaiono con coefficiente $+1$ in quanto entranti nel nodo $j \in N_t$, quelli degli archi $(i, j) \in A^-(N_s, N_t)$ appaiono con coefficiente -1 in quanto uscenti dal nodo $j \in N_t$, mentre i flussi degli archi (i, j) i cui estremi appartengono entrambi a N_t appaiono una volta con coefficiente $+1$ ed una con coefficiente -1 e quindi non contribuiscono alla somma. ◇

Esempio 2.16. Flusso e capacità di un taglio
Consideriamo il taglio $(N_s, N_t) = (\{1, 3, 5\}, \{2, 4, 6\})$ mostrato in figura; l'insieme degli archi diretti del taglio è $A^+(N_s, N_t) = \{(1, 2), (5, 6)\}$, mentre quello degli archi inversi è $A^-((N_s, N_t) = \{(2, 3), (4, 5)\}$. Il flusso del taglio è $x(N_s, N_t) = x_{12} + x_{56} - x_{23} - x_{45} = 7 + 7 - 3 - 3 = 8 = v$, mentre la capacità del taglio è $u(N_s, N_t) = u_{12} + u_{56} = 7 + 9 = 16$.

Esercizio 2.23. Cercare, se esiste, un taglio (N_s, N_t) nel grafo in Figura 2.16 avente una capacità inferiore a 16.

Esercizio 2.24. Si consideri il taglio $(N_s, N_t) = (\{1, 2, 5\}, \{3, 4, 6\})$ per il grafo in Figura 2.16: si forniscano gli insiemi degli archi diretti e inversi del taglio, e si calcolino il flusso e la capacità del taglio.

Esercizio 2.25. Ripetere l'esercizio precedente per il taglio $(N_s, N_t) = (\{1, 4, 5\}, \{2, 3, 6\})$.

Il Teorema 2.5 mostra che, comunque si prenda un taglio che separa t da s, il valore del flusso massimo non può eccedere la capacità di tale taglio. Di conseguenza, possiamo dimostrare che x è un flusso massimo se determiniamo un taglio $(N_s, N_t))$ la cui capacità sia uguale a v. Un taglio di questo tipo è in effetti determinato dalla visita del grafo residuo G_x nel caso in cui t non viene raggiunto. Sia infatti N_s l'insieme dei nodi visitati a partire da s, e $N_t = N \setminus N_s$: tutti gli archi di $A^+(N_s, N_t)$ sono saturi, altrimenti l'algoritmo avrebbe potuto visitare un ulteriore nodo, e analogamente, tutti gli archi di $A^-(N_s, N_t)$ sono vuoti. Dal Teorema 2.5 si ha allora $v = x(N_s, N_t) = u(N_s, N_t)$, ossia il flusso e la capacità del taglio coincidono: di conseguenza, x è un flusso massimo. Un'ulteriore importante conseguenza di questa relazione è che (N_s, N_t) è un *taglio di capacità minima* tra tutti i tagli del grafo che separano s da t. Abbiamo quindi dimostrato il seguente teorema:

Teorema 2.6. (Flusso Massimo-Taglio Minimo) Il massimo valore dei flussi ammissibili su G è uguale alla minima delle capacità dei tagli di G che separano s da t.

2.4.2 Algoritmo per cammini aumentanti

Le proprietà enunciate nel precedente paragrafo permettono di progettare un algoritmo per la determinazione di un flusso massimo: partendo da un flusso nullo, si determina ad ogni passo un cammino aumentante, incrementando il valore del flusso, e ci si ferma quando non esiste più alcun cammino aumentante, restituendo quindi anche un taglio di capacità minima.

```
procedure ( x , p ) = Cammini-Aumentanti( G , u , s , t )  {
  for( x = 0 ; ; ) {
    ( p , θ ) = Trova-Cammino( G , s , t , x , u );
    if( p[t] == 0 ) then break;
    x = Aumenta-Flusso( x , p , s , t , θ );
    }
  }
```

Procedura 2.6: Algoritmo basato su cammini aumentanti

La procedura *Trova-Cammino* cerca di determinare un cammino aumentante da s a t, dato il flusso x, visitando il grafo residuo G_x e fornendo il corrispondente vettore $p[\cdot]$ dei predecessori dei nodi nell'albero della visita e la capacità $\theta = \theta(P, x)$ del cammino aumentante P individuato tra s e t. Se t non viene raggiunto θ non è definito (ad esempio può avere valore 0), ma comunque l'algoritmo termina: in questo caso il vettore $p[\cdot]$ fornisce un taglio (N_s, N_t) di capacità minima (N_t sono tutti i nodi con predecessore 0, N_s gli altri). Altrimenti si aggiorna il flusso x mediante la procedura *Aumenta-Flusso* che invia lungo il cammino P la quantità di flusso θ, ossia implementa l'operazione di composizione $x = x \oplus \theta P$.

Per implementare la procedura *Trova-Cammino* è possibile evitare di costruire una rappresentazione di G_x modificando opportunamente la procedura *Visita* in modo che possa lavorare direttamente sulle strutture dati che descrivono il grafo originario G (si veda il §B.3.2 per altri esempi). Inoltre, è facile implementare *Trova-Cammino* in modo tale che, contemporaneamente al cammino, determini anche la sua capacità, memorizzando per ciascun nodo j, raggiunto nella visita, la capacità $d(j)$ dell'unico cammino da s a j nell'albero determinato fino a quel momento. Infatti, si supponga di visitare il nodo j provenendo dal nodo i: se si è usato l'arco (i, j) allora si ha che $d(j) = \min\{ d(i), u_{ij} - x_{ij} \}$, mentre se si è usato l'arco (j, i) allora si ha che $d(j) = \min\{ d(i), x_{ji} \}$; per inizializzare la procedura si pone $d(s) = +\infty$. La procedura *Aumenta-Flusso* può essere implementata percorrendo il cammino in senso inverso da t a s, per mezzo della funzione predecessore $p[\cdot]$, e modificando il flusso in accordo alla (2.12).

Esercizio 2.26. La funzione predecessore della procedura di visita su G_x non distingue l'orientamento degli archi del grafo originale: in altre parole, $p[j] = i$ non distingue se $(i, j) \in A_x^+$ oppure $(i, j) \in A_x^-$, ossia se si è usato l'arco (i, j) oppure l'arco (j, i) del grafo originario per raggiungere j. Si proponga una soluzione a questo problema. Si discuta inoltre come trattare il caso in cui siano presenti nel grafo archi "paralleli", ossia più copie dell'arco (i, j), con capacità diversa.

La terminazione dell'algoritmo è facile da mostrare sotto un'opportuna ipotesi relativa alle capacità:

Teorema 2.7. Se le capacità degli archi sono numeri interi, allora esiste almeno un flusso massimo intero, e l'algoritmo *Cammini-Aumentanti* ne determina uno in un numero finito di iterazioni.

Dimostrazione È facile verificare che, nel caso in cui le capacità siano intere, tutti i flussi x determinati dall'algoritmo sono interi: infatti lo è il flusso alla prima iterazione, e se all'inizio di un'iterazione x è intero, allora θ è un valore intero, e quindi anche $x(\theta)$ sarà un flusso a valori interi. Di conseguenza, ad ogni iterazione (a parte l'ultima) il valore del flusso viene aumentato di almeno un'unità; poiché il valore del flusso massimo è finito, necessariamente l'algoritmo deve terminare in un numero finito di iterazioni determinando la soluzione ottima del problema. ◇

Il Teorema 2.7 fornisce immediatamente una valutazione di complessità $O(mnU)$ per l'algoritmo, con $U = \max\{ u_{ij} : (i, j) \in A \}$. Infatti nU è maggiore della capacità del taglio $(\{ s \}, N \setminus \{ s \})$, e

pertanto anche della capacità minima dei tagli, e di conseguenza anche del massimo valore del flusso. Pertanto, il numero di iterazioni è al più nU, e poiché ogni iterazione costa $O(m)$, segue che la complessità dell'algoritmo è $O(mnU)$. Osserviamo che questa è una complessità *pseudopolinomiale*, essendo U uno dei dati numerici presenti nell'input del problema; ciò significa che anche istanze su grafi "di piccole dimensioni" possono richiedere molte iterazioni se le capacità sono "grandi".

Esercizio 2.27. Si costruisca un'istanza di (MF) con al più 4 nodi e 5 archi per la quale l'algoritmo *Cammini-Aumentanti* richieda effettivamente $\Theta(U)$ iterazioni.

Non solo, qualora le capacità non siano numeri interi l'algoritmo *Cammini-Aumentanti* non è né corretto né completo. Infatti è possibile costruire istanze, in cui alcune capacità sono numeri irrazionali, per cui la successione dei flussi x costruiti dall'algoritmo è infinita, e ancor peggio il valore del flusso converge ad un valore strettamente inferiore a quello del valore del flusso massimo (per i dettagli si rimanda alla letteratura citata). Ciò non è di grande rilevanza pratica (i numeri rappresentati su un computer digitale sono tipicamente interi o razionali), comunque entrambi i problemi possono essere risolti scegliendo in maniera opportuna il modo in cui viene determinato il cammino aumentante, ossia come è implementato l'insieme Q nella procedura *Trova-Cammino*. Infatti, mentre i risultati precedenti non dipendono in alcun modo da questa scelta, si possono dimostrare proprietà specifiche qualora, ad esempio, Q sia una *fila*, ossia si realizzi una *visita a ventaglio* del grafo residuo. Ciò permette di visitare ogni nodo mediante il cammino (aumentante) *più corto*, formato cioè dal minimo numero di archi (cf. il Teorema B.1). Ovvero si privilegiano inizialmente cammini aumentanti "corti", aumentando nelle iterazioni successive la loro lunghezza, ed arrivando eventualmente solo nelle ultime iterazioni ad utilizzare cammini aumentanti che passano attraverso tutti (o quasi) i nodi. L'algoritmo risultante prende il nome di *algoritmo di Edmonds & Karp*.

Esempio 2.17. Esecuzione dell'algoritmo di Edmonds & Karp

Un esempio di esecuzione dell'algoritmo di Edmonds & Karp è mostrato in Figura 2.17. Per ogni iterazione sono mostrati lo stato iniziale del flusso ed il cammino aumentante selezionato (archi in grassetto); nell'ultima iterazione è mostrato il taglio di capacità minima determinato dall'algoritmo.

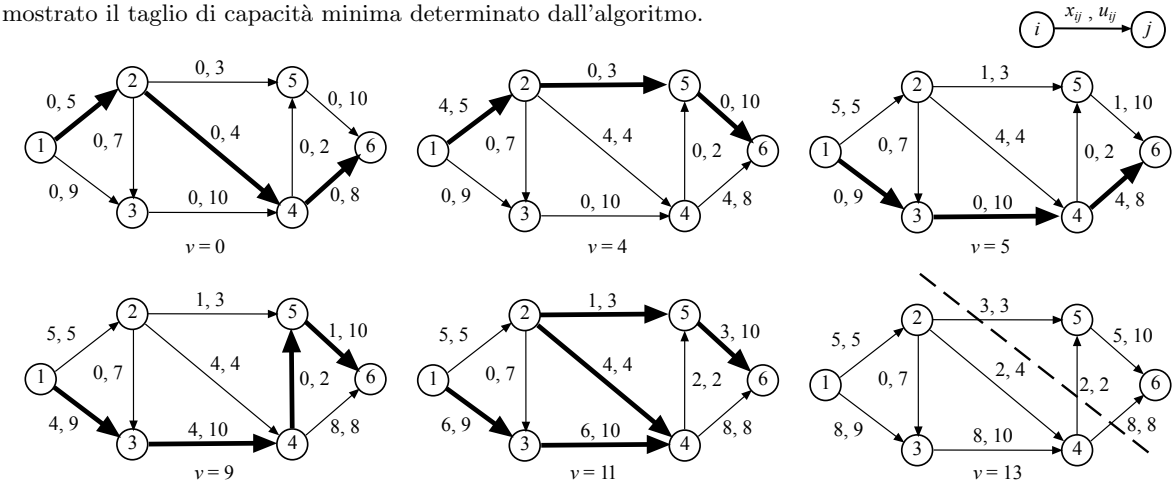


Figura 2.17: Esecuzione dell'algoritmo di Edmonds & Karp

Per l'algoritmo di Edmonds & Karp è possibile dimostrare una valutazione della complessità migliore di quella generale. Per dimostrarlo si consideri il generico flusso x^k determinato alla k-esima iterazione, il corrispondente grafo residuo G^k ed il cammino orientato p^k su G^k utilizzato per ottenere x^{k+1}. Indichiamo con $|p|$ il numero di archi di un generico cammino p, e con $\delta^k(i, j)$ la lunghezza del cammino minimo, in termini di numero di archi, da i a j su G^k (ponendo $\delta^k(i, j) = \infty$ se j non è raggiungibile da i). Vogliamo dimostrare che i cammini aumentanti "corti", per via della visita a ventaglio di G^k, sono utilizzati prima di quelli "lunghi".

Lemma 2.5. Per ogni i e k vale

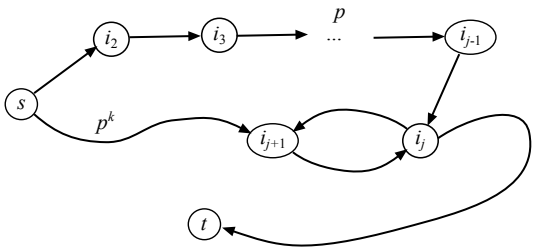
$$\delta^k(s, i) \leq \delta^{k+1}(s, i) \qquad \text{e} \qquad \delta^k(i, t) \leq \delta^{k+1}(i, t) \ , \tag{2.16}$$

e quindi, in particolare, $\delta^k(s, t) = |p^k|$ è non decrescente in k.

Dimostrazione Dimostriamo la disuguaglianza a sinistra in (2.16), perché l'altra segue in modo del tutto analogo. Per questo si fissino arbitrariamente i e k: se $\delta^{k+1}(s, i) = \infty$ non c'è niente da dimostrare, per cui si assuma $\delta^{k+1}(i, j) < \infty$. Sia adesso $p = \{i_1, i_2, \ldots, i_h, i_{h+1}\}$ uno qualsiasi dei cammini di lunghezza minima da s ad i in G^{k+1} (quindi $s = i_1$, $i = i_{h+1}$, $|p| = h = \delta^{k+1}(s, i)$): vogliamo mostrare che vale

$$\delta^k(s, i_{j+1}) \leq \delta^k(s, i_j) + 1 \qquad j = 1, \ldots, h \ . \tag{2.17}$$

Per questo si consideri un fissato j, ed il corrispondente arco $(i_j, i_{j+1}) \in p$: se $(i_j, i_{j+1}) \in G^k$, allora (2.17) è verificata perché l'arco permette di costruire un cammino da s ad i_{j+1} a partire da un qualsiasi cammino da s ad i_j.



Se invece $(i_j, i_{j+1}) \notin G^k$, poiché $(i_j, i_{j+1}) \in G^{k+1}$ allora possiamo sicuramente concludere che $(i_{j+1}, i_j) \in p^k$: l'arco non esisteva nel grafo residuo all'iterazione k ma esiste alla successiva, quindi o era saturo ed è stato parzialmente vuotato o viceversa, in ogni caso è stato usato "in direzione inversa" durante l'iterazione, come mostrato in figura qui accanto. Ma p^k è uno dei cammini di lunghezza minima su G^k, il che significa che $\delta^k(s, i_j) = \delta^k(s, i_{j+1}) + 1$, e quindi $\delta^k(s, i_{j+1}) = \delta^k(s, i_j) - 1 < \delta^k(s, i_j) + 1$, confermando anche in questo caso (2.17). A questo punto, sommando (2.17) per tutti i valori di $j = 1, \ldots, h$, usando $s = i_1$, $i = i_{h+1}$ e $\delta^k(s, s) = 0$ si ottiene $\delta^k(s, i) \leq h = \delta^{k+1}(s, i)$, ossia (2.16). ◇

Da questa proprietà deriva il seguente teorema:

Teorema 2.8. L'algoritmo di Edmonds & Karp ha complessità $O(m^2 n)$.

Dimostrazione Definiamo *bottleneck* un arco che viene saturato o vuotato in una data iterazione dell'algoritmo. Si consideri un arco (i, j) che è bottleneck per due iterazioni $h < l$; ovviamente $(i, j) \in p^h$ e $(i, j) \in p^l$, ma è anche ovvio che deve risultare $(j, i) \in p^k$ per qualche $h < k < l$ (se l'arco è bottleneck all'iterazione h non appare più in G^k per $k > h$ finché non viene utilizzato "al contrario" in un cammino aumentante). Possiamo allora dimostrare che $|p^k| \geq |p^h| + 2$. Infatti, poiché $(i, j) \in p^h$ abbiamo $|p^h| = \delta^h(s, i) + 1 + \delta^h(j, t)$; in più, $\delta^h(s, j) = \delta^h(s, i) + 1$ (il cammino minimo tra s e j passa per i), ed analogamente $\delta^h(i, t) = \delta^h(j, t) + 1$. Inoltre da $(j, i) \in p^k$ abbiamo

$$|p^k| = \delta^k(s, j) + 1 + \delta^k(i, t) \geq \delta^h(s, j) + 1 + \delta^h(i, t) = (\delta^h(s, i) + 1) + 1 + (\delta^h(j, t) + 1) = |p^h| + 2$$

dove per la prima disuguaglianza abbiamo usato (2.16). In altri termini, ogni arco non può essere bottleneck più di $n/2$ volte, e quindi non possono essere fatte più di $O(mn)$ iterazioni. Siccome ogni iterazione costa $O(m)$, il costo totale di $O(m^2 n)$. ◇

Si noti che questo risultato vale anche nel caso in cui le capacità non siano intere.

Esercizio 2.28. Partendo dal flusso $x = 0$, determinare il flusso massimo da 1 a 6 sul grafo in Figura 2.16, dove la capacità dell'arco $(4, 6)$ è $u_{46} = 5$, utilizzando l'algoritmo di Edmonds & Karp. Fornire per ogni iterazione l'albero della visita, il cammino aumentante, la sua capacità, il flusso e il suo valore. Fornire al termine il taglio di capacità minima.

2.4.3 Flusso massimo con più sorgenti / pozzi

Una generalizzazione del problema di flusso massimo è quella in cui si ha un insieme S di nodi sorgente ed un insieme T di nodi pozzo (entrambi non vuoti), e si vuole individuare il massimo flusso che può essere spedito dai nodi sorgente ai nodi pozzo. Questo problema può essere facilmente risolto applicando un qualunque algoritmo per il flusso massimo ad un grafo ampliato $G' = (N', A')$ con $N' = N \cup \{s, t\}$ e $A' = A \cup \{(s, j) : j \in S\} \cup \{(i, t) : i \in T\}$. I nodi s e t sono una "super-sorgente" ed un "super-pozzo", collegati rispettivamente a tutti i nodi in S e T con archi a capacità infinita. È possibile modificare l'algoritmo *Cammini-Aumentanti* in modo tale che risolva direttamente il problema più generale senza la necessità di costruire esplicitamente il grafo ampliato G'. Per questo è sufficiente che la procedura *Trova-Cammino* implementi una visita a partire dall'insieme di nodi S invece che dal singolo nodo s, ossia inizializzando $Q = S$, (si veda il §B.3.2); la visita è interrotta non appena si raggiunga un qualsiasi nodo in T, nel qual caso si è determinato un cammino aumentante da una delle sorgenti ad uno dei pozzi, oppure quando Q è vuoto, e quindi si è determinato un taglio saturo che separa *tutte* le sorgenti da *tutti* i pozzi.

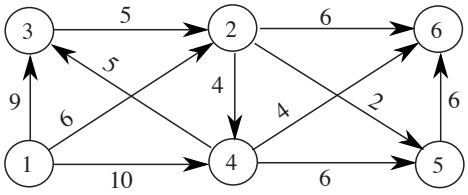
Un'ulteriore generalizzazione del problema è quella in cui ciascuna sorgente $i \in S$ e ciascun pozzo $i \in T$ ha una *capacità finita* u_i, ossia una massima quantità di flusso che può immettere nella/prelevare dalla rete. Anche questo problema può essere risolto applicando un algoritmo per il flusso massimo alla rete

ampliata G', con l'unica differenza che la capacità degli archi (s, i) e (i, t) viene posta a u_i e non a $+\infty$. Anche in questo caso è possibile modificare l'algoritmo per cammini aumentanti in modo tale che risolva la versione più generale del problema senza costruire esplicitamente la rete G'.

Esercizio 2.29. Si fornisca una descrizione formale, in pseudo-codice, dell'algoritmo per risolvere il problema del flusso massimo da un insieme di sorgenti S ad un insieme di destinazioni T, nei casi con e senza capacità associate alle sorgenti / destinazioni. Nel secondo caso, si presti attenzione alla necessità di mantenere traccia del valore x_i (il flusso sugli archi (s, i) e (i, t)) del flusso già immesso dalla sorgente / prelevato dal pozzo i nel flusso corrente x, mantenere gli insiemi S_x e T_x delle sorgenti / pozzi che possono ancora immettere / prelevare flusso (tali che gli archi (s, i) / (i, t) non sono saturi) dai quali iniziare / terminare la visita, ed utilizzare opportunamente le x_i durante il calcolo della capacità dei cammini.

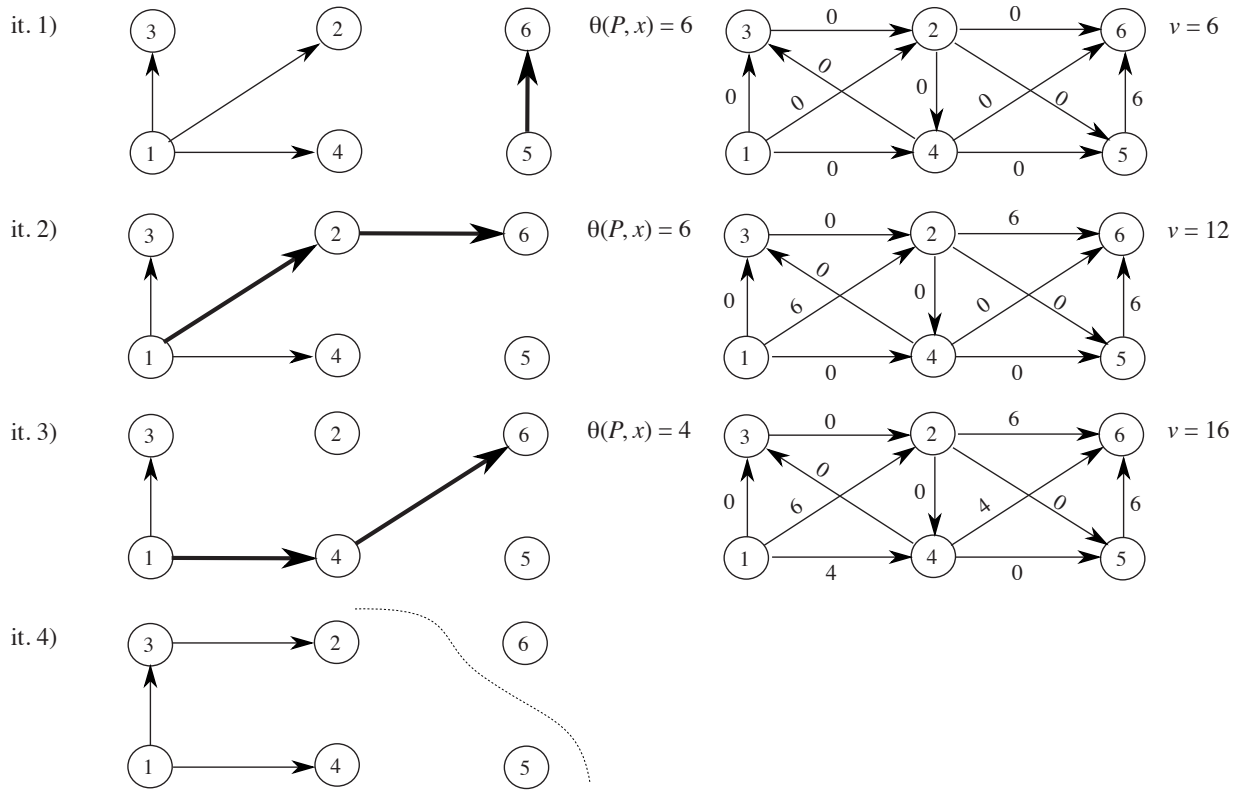
Esempio 2.18. Edmonds & Karp con sorgenti multiple

Si individui un flusso massimo dai due nodi sorgente 1 e 5 al nodo pozzo 6 sulla rete in figura, utilizzando l'algoritmo di Edmonds & Karp. Ad ogni iterazione si fornisca l'albero della visita, il cammino aumentante individuato con la relativa capacità, ed il flusso ottenuto con il relativo valore. Al termine, si indichi il taglio (N_s, N_t) restituito dall'algoritmo e la sua capacità, giustificando la risposta.



Risolvere un problema di flusso massimo con un insieme di sorgenti è equivalente a risolvere il problema di flusso massimo su un grafo aumentato, in cui è stata aggiunta una super-sorgente s collegata a tutte le sorgenti da archi a capacità infinita. Nello svolgimento eviteremo di esplicitare la super-sorgente s, piuttosto iniziando la visita del grafo residuo con $Q = \{1, 5\}$. Si ricordi però che le visite topologiche dipendono anche dall'ordinamento degli archi della stella uscente (del grafo residuo, e quindi anche di quella entrante del grafo originario), che in queste dispense consideriamo ordinati in modo crescente dei rispettivi nodi testa. Per rispettare tale regola anche per gli archi (non fisicamente presenti) nella stella uscente della super-sorgente inseriamo i nodi sorgente in Q in ordine crescente di nome, in modo da esaminarli (dato che Q è una fila) nello stesso ordine.

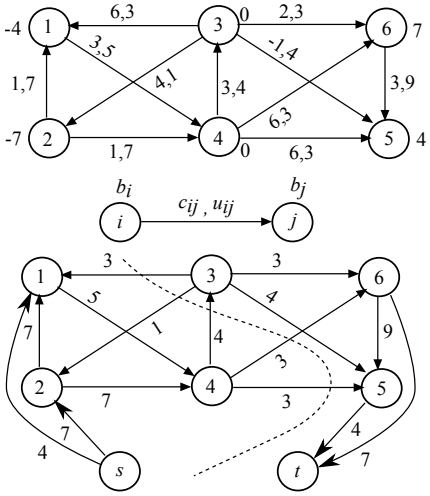
Le iterazioni sono rappresentate di seguito, dall'alto in basso. Per ogni iterazione, a sinistra è mostrato l'albero della visita ed il cammino aumentante P individuato (archi evidenziati); a destra viene invece indicato il flusso ottenuto in seguito all'invio, lungo P, di una quantità di flusso pari alla capacità $\theta(P, x)$, con il relativo valore v. Al termine è riportato il taglio $(N_s, N_t) = (\{1, 2, 3, 4, 5\}, \{6\})$ determinato dall'algoritmo. I nodi in N_s sono quelli esplorati durante l'ultima visita del grafo residuo (ovvero, la visita in cui si dimostra la non esistenza di un cammino aumentante): si noti che per definizione entrambe le sorgenti fanno parte di N_s, poiché gli archi uscenti dalla super-sorgente hanno capacità infinita e quindi non sono mai saturi. Il relativo albero della visita è illustrato nell'ultima figura in basso a sinistra. Il taglio è di capacità minima: infatti $u(N_s, N_t) = u_{26} + u_{46} + u_{56} = 6 + 4 + 6 = 16 = v$.



Il problema appena introdotto è particolarmente interessante anche in quanto esso coincide con il problema di determinare se esiste oppure no una soluzione ammissibile per (MCF).

Esercizio 2.30. Si dimostri l'affermazione precedente (suggerimento: si ricordi la Figura 2.4); si determini poi un flusso ammissibile per la rete in Figura 2.1 utilizzando l'algoritmo dell'esercizio precedente.

Esempio 2.19. Ammissibilità di (MCF) attraverso (MF)



Si vuole determinare se l'istanza (MCF) in figura ha oppure no una soluzione ammissibile. Per farlo è sufficiente risolvere un'istanza di (MF) costruita introducendo una "super-sorgente" s, collegata a tutte le origini, ossia i nodi con deficit negativo $O = \{ i \in N : b_i < 0 \}$, con archi la cui capacità è l'opposto del deficit del nodo, ed un "super-pozzo" t, a cui sono collegate tutte le destinazioni, ossia i nodi con deficit positivo $D = \{ i \in N : b_i > 0 \}$, con archi la cui capacità è il deficit del nodo.
Per l'istanza in oggetto la costruzione è mostrata in figura qui accanto. Tale istanza ha la caratteristica che tutti i flussi ammissibili per (MCF), se ne esistono, corrispondono a flussi il cui valore è $\bar{v} = \sum_{i \in D} b_i$, ossia che saturano tutti gli archi entranti in t (e, poiché $\sum_{i \in N} b_i = 0$, anche quelli uscenti da s). Pertanto è sufficiente esaminare i tagli di capacità minima: se tra di essi c'è il taglio $(\{ s \}, N \cup \{ t \})$, e quindi anche il taglio $(N \cup \{ s \}, \{ t \})$ che ha la stessa capacità $\bar{v}$, allora l'istanza è ammissibile. In questo caso, avendo a disposizione un qualsiasi flusso massimo è possibile determinare un flusso ammissibile per (MCF) semplicemente scartando gli archi "artificiali" (uscenti da s ed entranti in t) e mantenendo sugli altri il valore del flusso dato. Se invece esiste un taglio che separa s da t la cui capacità è inferiore a $\bar{v}$ allora non può esistere nessun flusso ammissibile; ciò discende direttamente dal Teorema 2.5. Si può quindi certamente rispondere alla domanda in positivo o in negativo esaminando il taglio di capacità minima, ma per dimostrare che non esiste alcun flusso ammissibile è sufficiente mostrare un qualsiasi taglio di capacità inferiore a $\bar{v}$.

Nel caso in questione, $\bar{v} = b_5 + b_6 = 4 + 7 = 11$. Il taglio $(N_s , N_t) = (\{ s , 1 , 2 , 4 \} , \{ 3 , 5 , 6 , t \})$ mostrato in figura ha capacità $u(N_s , N_t) = u_{43} + u_{45} + u_{46} = 4 + 3 + 3 = 10 < 11 = \bar{v}$; pertanto, l'istanza di (MCF) non ha alcuna soluzione ammissibile.

2.4.4 Algoritmo basato su preflussi

Gli algoritmi per (MF) visti finora, basati su cammini aumentanti, sono ragionevolmente efficienti in pratica ma non i migliori possibili; in effetti, per un grafo completo la complessità di Edmons & Karp è $O(n^5)$, che pur essendo polinomiale è comunque molto elevata. Sono stati recentemente proposti algoritmi in grado di risolvere (MF) con complessità quasi lineare, ossia vicina a $O(m)$, ma sono basati su tecniche algoritmiche molto sofisticate e la loro efficienza in pratica non è ancora chiaramente dimostrata. Sono però disponibili algoritmi di tipo diverso che hanno una complessità al caso pessimo molto inferiore a $O(n^5)$, sono semplici da implementare—anche in parallelo—ed hanno ottime prestazioni in pratica.

L'analisi di complessità suggerisce che uno dei limiti fondamentali degli algoritmi basati su cammini aumentanti sia il fatto che, ad ogni iterazione, nel caso peggiore può essere necessario esplorare tutto il grafo, senza alcuna garanzia di determinare un cammino aumentante lungo il quale sia possibile inviare una consistente quantità di flusso. Un approccio alternativo è basato sul concetto di *preflusso*, ossia di un vettore x che rispetta (2.3) e la "versione rilassata di (2.2)"

$$e_i = \sum_{(j,i)\in BS(i)} x_{ji} - \sum_{(i,j)\in FS(i)} x_{ij} \geq 0 \qquad i \in N \setminus \{ s, t \}$$

che permette che la quantità di flusso che arriva ad un nodo sia maggiore (ma non minore) di quella che ne esce. Un nodo i viene detto *attivo* se il suo *eccesso* e_i è positivo, altrimenti ($e_i = 0$) viene detto *bilanciato*. Per semplificare la presentazione dell'algoritmo e delle sue proprietà faremo costantemente riferimento al *grafo aumentato* $G' = (N, A' = A^+ \cup A^-)$ che contiene la coppia di archi $(i, j) \in A^+$ e $(j, i) \in A^-$ (detti "gemelli") per ciascun arco $(i, j) \in A$. Su G' si definisce l'immagine x' di un qualsiasi flusso x ponendo $x'_{ij} = x_{ij}$ per $(i, j) \in A^+$, $x'_{ij} = u_{ji} - x_{ji}$ per $(i, j) \in A^-$; si pone inoltre $u'_{ij} = u_{ij}$ per $(i, j) \in A^+$, $u'_{ij} = u_{ji}$ per $(i, j) \in A^-$. È quindi ovvio che il grafo residuo $G_x = (N, A_x)$ rispetto al preflusso x, definito come abbiamo già visto per il caso dei flussi, coincide col sottografo di G' che contiene i soli archi non saturi, ossia $x'_{ij} < u'_{ij}$. L'aggiornamento del flusso su un arco comporta quello sul suo "gemello": se $x'_{ij} = x'_{ij} + \theta$ per $(i, j) \in A^+$ ($x_{ij} = x_{ij} + \theta$ per $(i, j) \in A$) allora $x'_{ji} = x_{ji} - \theta$ per $(j, i) \in A^-$, e viceversa se $x'_{ij} = x'_{ij} + \theta$ per $(i, j) \in A^-$ ($x_{ji} = x_{ji} - \theta$ per $(j, i) \in A$) allora $x'_{ji} = x_{ji} - \theta$ per $(j, i) \in A^+$. In questo modo l'aggiornamento degli eccessi avviene in modo uniforme nei due casi: $x'_{ij} = x'_{ij} + \theta$ per $(i, j) \in A'$ implica $e_i = e_i - \theta$ ed $e_j = e_j + \theta$.

L'idea che sta alla base dell'algoritmo basato su preflussi è di cercare di spingere flusso verso la destinazione, usando ogni volta solo informazione locale, ossia relativa al nodo in esame e a quelli ad esso adiacenti. A questo scopo si definisce, per ogni nodo i, una etichetta d_i con la proprietà che $d_t = 0$ e $d_i - d_j \leq 1$ se $(i, j) \in A_x$. Un siffatto insieme di etichette viene detto *etichettatura valida*, ed è facile verificare che d_i è una *valutazione per difetto* della lunghezza (numero di archi) dei cammini aumentanti da i a t, ossia per qualsiasi etichettatura valida non esistono cammini dal nodo i al pozzo t su G_x formati da meno di d_i archi. Data un'etichettatura valida, un arco $(i, j) \in A_x$ è detto *ammissibile per* i se $d_i = d_j + 1$. Se i è un nodo in attivo per x ed esiste un arco ammissibile (i, j), allora è possibile inviare l'eccesso, o una parte di esso, da i al nodo j che risulta, per l'etichettatura valida, "più vicino" a t attraverso l'*operazione di push*

$$Push(\, i, j, x, d \,)\{\, \theta = \min\{ e_i, u'_{ij} - x'_{ij} \};\; x'_{ij} = x'_{ij} + \theta; e_i = e_i - \theta;\; e_j = e_j + \theta;\; \}\ .$$

Grazie all'uso del grafo aumentato possiamo evitare di distinguere tra le operazioni di *push in avanti*, in cui $(i, j) \in A^+_x$, e quelle di *push all'indietro*, in cui $(i, j) \in A^-_x$, evitando anche di specificare l'aggiornamento del flusso sull'arco "gemello". Se invece il nodo attivo i non ha archi incidenti che siano ammissibili, significa che per ogni arco $(i, j) \in A'$ si ha $x'_{ij} = u'_{ij}$ (l'arco è saturo, ossia $(i, j) \notin A_x$) oppure $d_i < d_j + 1$. In altri termini, possiamo affermare che non esistono cammini aumentanti da i a t formati da d_i archi; quindi l'etichetta di i può essere incrementata attraverso l'*operazione di relabel*

$$Relabel(\, d, i \,)\{\, d_i = 1 + \min\{ d_j \,:\, (i,j) \in A_x \},\ \}\ .$$

L'operazione di relabel rende ammissibile almeno un arco incidente in i (quello per cui si è ottenuto il valore minimo). Si può facilmente verificare che, se da un certo nodo i non si possono effettuare

operazioni di push, allora l'applicazione di un'operazione di relabel del nodo i a partire da un'etichettatura valida produce una nuova etichettatura anch'essa valida. Si noti che la massima lunghezza di un cammino aumentante è $n-1$. Pertanto, per ogni nodo i con $d_i \geq n$ si può affermare che non esistono cammini aumentanti da esso a t. Presentiamo adesso l'algoritmo basato su preflussi, che utilizza le operazioni di push e relabel per risolvere il problema del flusso massimo. L'idea dell'algoritmo consiste nell'introdurre nel grafo una quantità di flusso maggiore o uguale al valore del flusso massimo saturando tutti gli archi uscenti da s, e poi utilizzare un'etichettatura valida d per "assistere" il flusso nel suo tragitto fino a t.

```
( x , d ) = procedure Preflow-Push( G , u , s , t )  {
  x = 0; foreach( ( s , j ) ∈ FS( s ) ) do x_sj = u_sj;
  d = Etichettatura-Valida( G , t ); d_s = n;
  while( ∃ i ∈ N \ { s , t } con e_i > 0 ) do
    if( ∃( i , j ) ∈ A_x con d_i = d_j + 1 )
      then Push( i , j , x , d );
      else Relabel( d , i );
  }
```

Procedura 2.7: Algoritmo basato su preflussi

L'inizializzazione introduce nel grafo una quantità di flusso pari alla capacità del taglio $(\{s\}, N \setminus \{s\})$, e quindi maggiore o uguale al valore del flusso massimo, ed un'etichettatura valida d, ad esempio ponendo d_i uguale alla lunghezza del cammino minimo, in termini di numero di archi, da i a t. Ciò può essere facilmente ottenuto mediante una visita a ventaglio (in cui Q è implementato come una fila) "all'indietro" di G_x a partire dal nodo t, ossia percorrendo gli archi all'inverso del loro orientamento (cf. §B.3.2 ed il Teorema B.1). Si noti che la visita può escludere tutti gli archi uscenti da s, e quindi visitare solamente archi vuoti ($x_{ij} = 0$) nel verso opposto a quello del loro orientamento; in termini di G' si possono quindi visitare esattamente gli archi di $A' \setminus A_x$. Inizia a questo punto il "main cycle" dell'algoritmo: se è possibile effettuare un'operazione di push, l'algoritmo sposta flusso da un nodo attivo i ad un nodo j "più vicino" a t rispetto a i (secondo l'etichettatura valida d), mentre se non sono possibili operazioni di push, allora l'etichetta di un nodo attivo viene incrementata mediante un'operazione di relabel. L'algoritmo termina quando non ci sono più nodi attivi: tutto il flusso possibile è arrivato a t, il rimanente (la differenza tra la capacità del taglio $(\{s\}, N \setminus \{s\})$ e quella del taglio di capacità minima) è stato rispedito indietro a s. Al termine l'algoritmo, oltre ad un flusso massimo, individua anche un taglio (N_s, N_t) di capacità minima, "codificato" attraverso il vettore di etichette d: in particolare, fanno parte di N_s tutti quei nodi che, al termine dell'algoritmo, hanno un'etichetta di valore maggiore od uguale a n.

Un modo figurato per visualizzare il significato delle etichette è quello di considerarle come l'altezza dei nodi rispetto ad un piano comune: il flusso va dai nodi "più in alto" a quelli "più in basso", ma scendendo di un solo livello. Se un nodo ha un eccesso di flusso che non riesce a smaltire, viene portato al livello superiore del più in basso dei suoi vicini (raggiungibili attraverso archi non saturi o non vuoti) in modo da consentirgli di diminuire il suo sbilanciamento. Per visualizzare il comportamento dell'algoritmo riportiamo adesso un esempio di una sua esecuzione.

Esempio 2.20. Esecuzione di *Preflow-Push*

Nella Tabella 2.1 è riportata una possibile descrizione di tutte le operazione compiute dall'algoritmo *Preflow-Push* per risolvere l'istanza di Figura 2.17, a partire dall'etichettatura valida $d = [6, 2, 2, 1, 1, 0]$. In questo caso non specifichiamo esattamente come venga selezionata l'operazione di push o relabel effettuata tra tutte quelle possibili (quando ve n'è più di una). Una descrizione grafica delle operazioni è anche mostrata in Figura 2.18 (da sinistra a destra, dall'alto in basso); per brevità, nella figura non sono mostrate tutte le iterazioni, ma sono evidenziati gli archi coinvolti in operazioni di push.

nodo	oper.	arco	θ	correzioni			
2	push	(2 , 4)	3	$e_x(2) = 2$	$x_{24} = 3$	$e_x(4) = 3$	
2	push	(2 , 5)	2	$e_x(2) = 0$	$x_{25} = 2$	$e_x(5) = 2$	
3	push	(3 , 5)	9	$e_x(3) = 0$	$x_{35} = 9$	$e_x(5) = 11$	
4	push	(4 , 6)	3	$e_x(4) = 0$	$x_{46} = 3$	$v_t = 3$	
5	push	(5 , 6)	8	$e_x(5) = 3$	$x_{56} = 8$	$v_t = 11$	
5	relabel				$d_5 =$	$d_4 + 1$	$= 2$
5	push	(5 , 4)	2	$e_x(5) = 1$	$x_{54} = 2$	$e_x(4) = 2$	
4	push	(4 , 6)	2	$e_x(4) = 0$	$x_{46} = 5$	$v_t = 13$	
5	relabel				$d_5 =$	$d_2 + 1$	$= 3$
5	push	(2 , 5)	1	$e_x(5) = 0$	$x_{25} = 1$	$e_x(2) = 1$	
2	relabel				$d_2 =$	$d_3 + 1$	$= 3$
2	push	(2 , 3)	1	$e_x(2) = 0$	$x_{23} = 1$	$e_x(3) = 1$	
3	relabel				$d_3 =$	$d_2 + 1$	$= 4$
3	push	(2 , 3)	1	$e_x(3) = 0$	$x_{23} = 0$	$e_x(2) = 1$	
2	relabel				$d_2 =$	$d_5 + 1$	$= 4$
2	push	(2 , 5)	1	$e_x(2) = 0$	$x_{25} = 2$	$e_x(5) = 1$	
5	relabel				$d_5 =$	$d_2 + 1$	$= 5$
5	push	(2 , 5)	1	$e_x(5) = 0$	$x_{25} = 1$	$e_x(2) = 1$	
2	relabel				$d_2 =$	$d_3 + 1$	$= 5$
2	push	(2 , 3)	1	$e_x(2) = 0$	$x_{23} = 1$	$e_x(3) = 1$	
3	relabel				$d_3 =$	$d_2 + 1$	$= 6$
3	push	(2 , 3)	1	$e_x(3) = 0$	$x_{23} = 0$	$e_x(2) = 1$	
2	relabel				$d_2 =$	$d_5 + 1$	$= 6$
2	push	(2 , 5)	1	$e_x(2) = 0$	$x_{25} = 2$	$e_x(5) = 1$	
5	relabel				$d_5 =$	$d_2 + 1$	$= 7$
5	push	(3 , 5)	1	$e_x(5) = 0$	$x_{35} = 8$	$e_x(3) = 1$	
3	relabel				$d_3 =$	$d_2 + 1$	$= 7$
3	push	(3 , 1)	1	$e_x(3) = 0$	$x_{13} = 8$	$v_s = 13$	STOP

Tabella 2.1: Esecuzione dell'algoritmo *Preflow-Push*

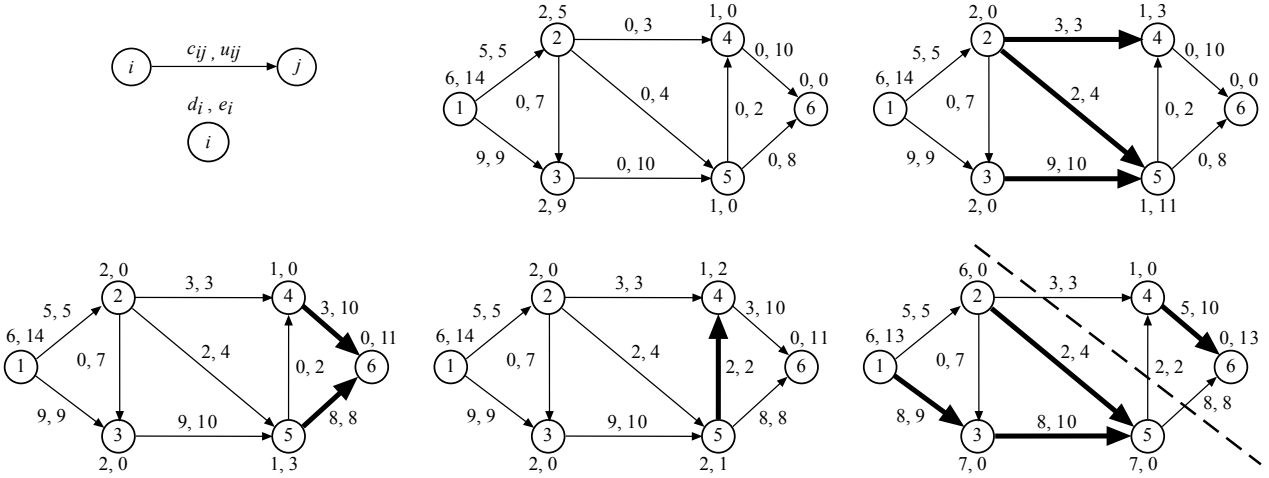


Figura 2.18: Esecuzione dell'algoritmo *Preflow-Push*

L'esempio precedente mostra come l'algoritmo compia un numero di iterazioni tipicamente molto maggiore di quelle di Edmons & Karp, ma ciascuna di esse ha complessità al più $O(|S(i)|)$, ed è quindi tipicamente molto più veloce di una visita che ha complessità $O(m)$, soprattutto per grafi di grandi dimensioni. Una peculiarità dell'algoritmo è che si inizia con un preflusso che ha sicuramente un valore maggiore od uguale al valore del flusso massimo. Una volta che tutto il flusso possibile è giunto a t, quello eventualmente in eccesso viene riportato a s: per questo, le etichette dei nodi appartenenti a N_s nel taglio minimo devono crescere ad un valore maggiore o uguale a n. L'algoritmo esegue allora una sequenza di operazioni di push e relabel mediante le quali i nodi in N_s si inviano ripetutamente le unità di flusso in eccesso al solo scopo di far aumentare le loro etichette fino ad un valore che consenta di re-instradare il flusso in eccesso fino ad s. Nell'esempio questa fase dell'algoritmo inizia alla nona iterazione, e coinvolge i nodi 2, 3 e 5 e gli archi (2 , 3), (2 , 5) e (3 , 5).

Si può dimostrare che, scegliendo opportunamente il preflusso iniziale, è sempre possibile effettuare o

un'operazione di push oppure un'operazione di relabel. Da questo può essere dedotta la completezza dell'algoritmo: la sequenza di operazioni di push tra due operazioni di relabel consecutive sposta eccesso da nodi più lontani verso nodi più vicini a t (rispetto alla etichettatura valida d che non cambia nel corso di tali operazioni). Pertanto, non è possibile avere due volte lo stesso stato di etichettatura e di eccessi. Inoltre, nessun nodo avrà mai etichetta maggiore di $2n-2$: infatti, quando l'etichetta di un nodo in eccesso i viene aggiornata ad un valore $d_i \geq n$, essa eccede di un'unità l'etichetta di un nodo j che si trova lungo un *cammino di ritorno* verso s, formato da al più $n-2$ archi. Pertanto non si eseguiranno mai più di $2n-3$ operazioni di relabel per uno stesso nodo, e quindi il numero globale di operazioni di relabel è $O(n^2)$. È possibile dimostrare che, se i nodi in eccesso vengono selezionati secondo criteri specifici, il numero globale di push che possono essere effettuate è $O(n^3)$, e che questo domina la complessità dell'algoritmo. Per i dettagli si rimanda alla letteratura citata.

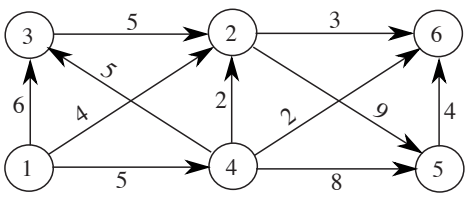
Altrettanto importante è il fatto che l'algoritmo, se opportunamente implementato, risulta essere molto efficiente in pratica. Ciò richiede alcune accortezze, in particolare per evitare, almeno parzialmente, la fase finale di re-instradamento del flusso in eccesso. Alcune di tali accortezze si basano sull'evento che vi sia un *salto di etichettatura*, cioè che esista un valore intero $k < n$ per cui nessun nodo ha etichetta uguale a k. In tal caso, a tutti i nodi i con etichetta $k < d_i < n$ può essere assegnata un'etichetta $d_i = n+1$.

Esempio 2.21. Salto di etichettatura
Questo capita nell'esempio precedente, subito dopo l'operazione di relabel del nodo 3 in cui $d_3 = 4$. Infatti, in quel momento, il vettore delle etichette è $d = [\,6\,,\,3\,,\,4\,,\,1\,,\,3\,,\,0\,]$. Verificandosi un salto di etichettatura per $k = 2$, si può porre $d_2 = d_3 = d_5 = 7$, risparmiando 14 operazioni e potendo re-instradare immediatamente l'unità di flusso in eccesso verso il nodo sorgente.

L'algoritmo si presta anche ad altre modifiche. Ad esempio, si può dimostrare che se si è interessati solamente al taglio di capacità minima (e non anche al flusso massimo), è possibile terminare l'algoritmo nel momento in cui tutti i nodi attivi hanno etichetta $d_i \geq n$: l'insieme di tutti i nodi in quel momento non raggiungibili da s attraverso cammini aumentanti fornisce infatti l'insieme N_t di un taglio di capacità minima. Si noti che, nell'esempio precedente, questo avviene nel momento del salto di etichettatura, che permetterebbe quindi di terminare l'algoritmo. Infine, è possibile proporre implementazioni parallele dell'algoritmo in ambiente asincrono, che richiedono solamente comunicazione locale tra processori che controllano nodi adiacenti; per questi dettagli si rimanda ancora una volta alla letteratura citata.

Esercizio 2.31. Si discuta come modificare la procedura *Preflow-Push* per risolvere le due varianti del problema di flusso massimo con più sorgenti / pozzi, presentate nel paragrafo 2.4.3, senza costruire esplicitamente il grafo aumentato G' (si noti che, nella prima variante, gli archi uscenti dalla "super-radice" s hanno capacità infinita).

Esempio 2.22. Preflow-Push con salto di etichettatura
Si vuole determinare il flusso massimo tra il nodo 1 ed il nodo 6, relativamente all'istanza in figura, utilizzando l'algoritmo Preflow-Push. Si userà la seguente strategia per la selezione dell'operazione da compiere: si seleziona il nodo attivo con etichetta più bassa, ed a parità di etichetta quello con indice più alto; per l'operazione di push si seleziona l'arco (attivo) che porta al nodo con indice più alto. Si vuole anche indicare quale sia il taglio di capacità minima individuato dall'algoritmo appena esso sia individuato.



Inizialmente si determina un'etichettatura valida attraverso una visita a ventaglio "all'indietro" del grafo a partire dal pozzo 6; questa produce $d_6 = 0$, $d_2 = d_4 = d_5 = 1$, $d_3 = 2$, mentre si pone $d_1 = n = 6$. Si saturano inoltre tutti gli archi uscenti dalla sorgente, ossia $x_{12} = 4$, $x_{13} = 6$, e $x_{14} = 5$, mentre tutti gli altri flussi sono nulli; di conseguenza $e_2 = 4$, $e_3 = 6$, $e_4 = 5$, $e_5 = 0$. Si indica con "push(j, θ)" l'operazione di push di θ unità di flusso sul nodo j, mentre il nodo di origine i è indicato separatamente. Non si distingue quindi tra push "all'avanti" e "all'indietro", ossia se sia utilizzato l'arco (i, j) o quello (j, i), poiché la cosa che si evince chiaramente dal contesto.

it.	i	op.	cambiamenti
1	4	push(6,2)	$x_{46} = 2$, $e_4 = 3$
2	4	relabel	$d_4 = 2$
3	2	push(6 , 3)	$x_{26} = 3$, $e_2 = 1$
4	2	relabel	$d_2 = 2$
5	4	push(5 , 3)	$x_{45} = 3$, $e_4 = 0$, $e_5 = 3$
6	5	push(6 , 3)	$x_{56} = 3$, $e_5 = 0$
7	3	relabel	$d_3 = 3$
8	2	push(5 , 1)	$x_{25} = 1$, $e_2 = 0$, $e_5 = 1$
9	5	push(6 , 1)	$x_{56} = 4$, $e_5 = 0$
10	3	push(2 , 5)	$x_{32} = 5$, $e_3 = 1$, $e_2 = 5$
11	2	push(5 , 5)	$x_{25} = 6$, $e_2 = 0$, $e_5 = 5$
12	5	relabel	$d_5 = 3$

All'iterazione 12 si determina un salto di etichettatura: esistono nodi di etichetta 2 (2 e 4), ma nessuno di etichetta 1. L'algoritmo quindi porta istantaneamente le etichette di tutti i nodi che hanno etichetta maggiore di 1 a 7; pertanto, $d_2 = d_3 = d_4 = d_5 = 7$. A questo punto, si è già determinato il taglio $(N_s , N_t) = (\{ 1 , 2 , 3 , 4 , 5 \} , \{ 6 \})$, dove N_s è l'insieme dai nodi con etichetta $\geq 6 = n$, di capacità minima $u(N_s , N_t) = u_{26} + u_{46} + u_{56} = 3 + 2 + 4 = 9$, pari infatti al flusso che è già arrivato a 6. Per determinare un flusso ammissibile occorre far terminare l'algoritmo:

it.	i	op.	cambiamenti
13	5	relabel	$d_5 = 8$
14	5	push(4 , 3)	$x_{45} = 0$, $e_5 = 2$, $e_4 = 3$
15	4	push(1 , 3)	$x_{14} = 2$, $e_4 = 0$
16	5	push(2 , 2)	$x_{25} = 4$, $e_5 = 0$, $e_2 = 2$
17	3	push(1 , 1)	$x_{13} = 5$, $e_3 = 0$
18	2	push(1 , 2)	$x_{12} = 2$, $e_4 = 0$

A questo punto si è ottenuto un flusso ammissibile di valore $v = 9$, pari alla capacità del taglio, e l'algoritmo termina.

2.5 Il problema del Flusso di Costo Minimo

Siamo finalmente adesso in condizione di studiare algoritmi risolutivi per il problema (MCF) come introdotto nel §2.1. È ovvio che (MCF) è più generale sia di (SPT) che di (MF), in quanto, come abbiamo visto, questi ultimi possono essere formulati come particolari problemi di flusso di costo minimo. È utile rimarcare come (SPT) e (MF) in qualche modo catturano due parti diverse della struttura di (MCF), che solo nel problema generale sono contemporaneamente presenti: in (SPT) gli archi hanno associati costi ma non capacità, mentre in (MF) gli archi hanno associate capacità ma non costi. Questo in effetti porta ad algoritmi che "guardano aspetti diversi di un flusso": ottimalità rispetto ai costi per (SPT), ammissibilità rispetto alle capacità per (MF). In effetti, vedremo come alcuni algoritmi per (MCF) fanno uso, al loro interno, di algoritmi per (SPT) o (MF) esattamente per affrontare l'uno o l'altro dei due aspetti. Si potrebbe pensare (SPT) ed (MF), per come inizialmente presentati, differiscano in modo sostanziale da (MCF) anche per via del fatto che la struttura delle domande / offerte dei nodi è molto particolare; i §2.2.7 e §2.4.3 però mostrano come questo non sia veramente il caso.

2.5.1 Cammini, cicli aumentanti e condizioni di ottimo

Il primo passo per lo sviluppo di algoritmi per un qualsiasi problema di ottimizzazione, e quindi anche per (MCF), consiste generalmente nello studio delle *condizioni di ottimalità per il problema*, ossia di proprietà che consentano di verificare se una data soluzione—in questo caso, un flusso x—sia ottima. Per poter presentare una più vasta gamma di algoritmi, però, deriveremo condizioni di ottimo non per i soli flussi, ma per il sovrainsieme degli *pseudoflussi*, ossia dei vettori $x \in \mathbb{R}^m$ che rispettano i soli vincoli di capacità sugli archi (2.3). Definiamo *sbilanciamento* di un nodo i rispetto ad x la quantità

$$e_x(i) = \sum_{(j , i) \in BS(i)} x_{ji} - \sum_{(i , j) \in FS(i)} x_{ij} - b_i$$

e indichiamo con $O_x = \{ i \in N : e_x(i) > 0 \}$ e $D_x = \{ i \in N : e_x(i) < 0 \}$, rispettivamente, l'insieme dei nodi con *eccedenza di flusso* e con *difetto di flusso*. Si noti come gli sbilanciamenti abbiano "il segno rovesciato" rispetto ai deficit dei nodi: i nodi con sbilanciamento positivo, o eccedenza di flusso, sono quelli che, considerato l'effetto del flusso già presente sugli archi devono ancora inserire flusso

nella rete, mentre le sorgenti come definite in (2.1) hanno deficit negativo. Il viceversa vale per nodi con sbilanciamento negativo e pozzi. Ciò è ovvio pensando al fatto che se $x_{ij} = 0$ per tutti gli archi $(i, j) \in A$, allora $e_x(i) = -b_i$ per ogni nodo $i \in N$.

È ovvio che gli pseudoflussi contengono i flussi ammissibili come caso particolare. Infatti, le tre condizioni equivalenti (come è facile dimostrare) $D_x = \emptyset$, $O_x = \emptyset$ ed $e_x = 0$ implicano che il vettore x rispetti anche i vincoli (2.2), e che pertanto sia un flusso ammissibile. Un nodo $i \in N$ con $e_x(i) = 0$ (e che quindi non appartiene né a O_x né a D_x) si dice *bilanciato* rispetto allo pseudoflusso x; pertanto, un flusso ammissibile è uno pseudoflusso in cui tutti i nodi sono *bilanciati*. Un'altra condizione equivalente usa lo *sbilanciamento complessivo* di x

$$g(x) = \sum_{i \in O_x} e_x(i) \;\; (= -\sum_{j \in D_x} e_x(j)) \;\geq 0 \quad :$$

x è un flusso ammissibile se e solo se $g(x) = 0$.

Dato un cammino P, non necessariamente orientato, tra una qualunque coppia di nodi s e t del grafo, definiamo come verso di P quello che va da s a t. Gli archi del cammino risultano quindi partizionati nei due insiemi P^+ e P^-, rispettivamente degli archi concordi e discordi con il verso del cammino. Un cammino si dirà *aumentante* se la sua capacità $\theta(P, x)$, definita in (2.13), è positiva. Dato uno pseudoflusso x, è possibile inviare una quantità di flusso $0 < \theta \leq \theta(P, x)$ lungo P mediante l'*operazione di composizione* definita in (2.12), ottenendo un nuovo pseudoflusso $x(\theta) = x \oplus \theta P$ tale che

$$e_{x(\theta)}(i) = \begin{cases} e_x(s) - \theta & \text{se } i = s \\ e_x(t) + \theta & \text{se } i = t \\ e_x(i) & \text{altrimenti} \end{cases} .$$

In altre parole, inviare flusso lungo un cammino aumentante modifica solamente lo sbilanciamento dei nodi estremi del cammino, mentre lo sbilanciamento di tutti gli altri nodi rimane invariato. Un caso particolare di cammino aumentante è quello in cui $s = t$, ossia il cammino è un *ciclo* su cui è arbitrariamente fissato un verso di percorrenza. Chiaramente, inviare flusso lungo un ciclo (aumentante) non modifica lo sbilanciamento di alcun nodo; di conseguenza, se in particolare x è un flusso *ammissibile*, allora ogni flusso $x(\theta) = x \oplus \theta C$ per $0 \leq \theta \leq \theta(C, x)$ è ancora ammissibile.

Il *costo* di un cammino (o ciclo) P, che indicheremo con $c(P)$, è il costo di invio di un'unità di flusso lungo P secondo il verso fissato, ossia

$$c(P) = \sum_{(i,j) \in P^+} c_{ij} - \sum_{(i,j) \in P^-} c_{ij} \;\; . \tag{2.18}$$

Questa definizione è legata a quella dell'operazione di composizione: su ogni arco concorde (i, j) il flusso aumenta di θ, e quindi il costo del flusso su quell'arco cambia da $c_{ij}x_{ij}$ a $c_{ij}(x_{ij}+\theta) = c_{ij}x_{ij}+c_{ij}\theta$. Quindi, ciascun arco concorde contribuisce con un termine c_{ij} (moltiplicato per la quantità di flusso inviata) al cambiamento del costo complessivo. Viceversa, per gli archi discordi il costo cambia in $c_{ij}(x_{ij} - \theta) = c_{ij}x_{ij} - c_{ij}\theta$, ossia l'arco contribuisce con un termine $-c_{ij}$ (ancora moltiplicato per la quantità di flusso inviata) al cambiamento del costo complessivo. Tutto ciò mostra che il cambiamento complessivo del costo corrispondente ad inviare θ unità di flusso lungo il cammino è lineare in θ e $c(P)$, ossia

$$cx(\theta) = c(x \oplus \theta P) = cx + \theta c(P) \; . \tag{2.19}$$

Pertanto, cammini di costo negativo fanno diminuire (migliorare) il costo del flusso e viceversa, ma in questo occorre ovviamente tenere presente l'effetto sull'ammissibilità del flusso. Rimandando al seguito la discussione sulla soddisfazione o meno dei vincoli di conservazione di flusso (2.2) (che non sono richiesti dagli pseudoflussi), è chiaro che questa operazione può essere fatta solamente se non viola i vincoli di capacità (2.3), ossia se P è aumentante (e $\theta \leq \theta(P, x)$). Pertanto, determinare cammini aumentanti di costo negativo è un'operazione che appare (ed è) cruciale per lo sviluppo di algoritmi, e prima ancora per le condizioni di ottimo del problema.

Per determinare cammini (o cicli) aumentanti si può usare il *grafo residuo pesato* $G_x = (N, A_x)$ rispetto allo pseudoflusso x, definito come per il problema (MF) tranne per il fatto che agli archi sono anche associati dei costi: per ogni arco $(i, j) \in A$ si pone (i, j) in A_x, con costo $c'_{ij} = c_{ij}$, se e solo

se $x_{ij} < u_{ij}$, e si pone (j, i) in A_x, con costo $c'_{ji} = -c_{ij}$, se e solo se $x_{ij} > 0$. È immediato verificare che vale la seguente generalizzazione del Lemma 2.4:

Lemma 2.6. Comunque si fissino s e t, per ogni cammino aumentante da s a t rispetto ad x in G esiste uno ed un solo cammino orientato da s a t in G_x, ed i due cammini hanno lo stesso costo.

Possiamo ora dimostrare che cammini e cicli aumentanti sono gli "strumenti base per costruire pseudoflussi, e quindi flussi".

Teorema 2.9. *[Decomposizione degli pseudoflussi]* Comunque scelti due pseudoflussi x' ed x'', esistono $k \leq n + m$ cammini o cicli aumentanti (semplici) rispetto a x', $P_1, \ldots, P_k$, di cui al più m sono cicli, tali che $x^1 = x'$, $x^{i+1} = x^i \oplus \theta_i P_i$, per $i = 1, \ldots, k$, $x^{k+1} = x''$, dove $0 < \theta_i \leq \theta(P_i, x')$. In particolare, tutti i cammini aumentanti hanno come estremi nodi in cui lo sbilanciamento di x' è diverso dallo sbilanciamento di x'', per cui se $e_{x'} = e_{x''}$ allora tutti i P_i sono cicli.

Dimostrazione La dimostrazione è costruttiva: manteniamo uno pseudoflusso x, inizialmente pari ad x', ed in un numero finito di passi lo rendiamo uguale ad x'' utilizzando cammini e cicli aumentanti rispetto a x'. Per questo definiamo il grafo $\bar{G}_x = (N, \bar{A}^+_x \cup \bar{A}^-_x)$, dove $\bar{A}^+_x = \{(i, j) : x''_{ij} > x_{ij}\}$ e $\bar{A}^-_x = \{(j, i) : x''_{ij} < x_{ij}\}$. $\bar{G}_x$ "descrive" la differenza tra x ed x''; è immediato verificare che $\bar{A}^+_x = \bar{A}^-_x = \emptyset$ se e solo se $x'' = x$. Ad ogni arco $(i, j) \in \bar{A}^+_x$ associamo la capacità $u^x_{ij} = x''_{ij} - x_{ij} > 0$, e, analogamente, ad ogni arco $(j, i) \in \bar{A}^-_x$ associamo la capacità $u^x_{ji} = x_{ij} - x''_{ij} > 0$; per ogni cammino (ciclo) orientato P in $\bar{G}_x$ definiamo come sua capacità $\theta^x(P) = \min\{u^x_{ij} : (i, j) \in P\}$. Definiamo inoltre gli insiemi $\bar{O}_x = \{i \in N : e_x(i) > e_{x''}(i)\}$ e $\bar{D}_x = \{i \in N : e_x(i) < e_{x''}(i)\}$, rispettivamente, dei nodi che hanno sbilanciamento rispetto a x maggiore dello sbilanciamento rispetto a x'' e di quelli in cui avviene l'opposto. È facile verificare che $\bar{O}_x = \bar{D}_x = \emptyset$ se e solo se x ed x'' hanno lo stesso vettore di sbilanciamento; inoltre tutti i nodi in $\bar{O}_x$ hanno almeno un arco uscente in $\bar{G}_x$, tutti i nodi in $\bar{D}_x$ hanno almeno un arco entrante in $\bar{G}_x$, mentre tutti i nodi in $N \setminus (\bar{O}_x \cup \bar{D}_x)$ o non hanno né archi entranti né archi uscenti oppure hanno sia almeno un arco entrante che almeno un arco uscente.

Utilizzando $\bar{G}_x$ è possibile costruire iterativamente i cicli e cammini richiesti. Se $\bar{A}^+_x = \bar{A}^-_x = \emptyset$, ossia $x = x''$, il procedimento termina, altrimenti consideriamo gli insiemi $\bar{O}_x$ e $\bar{D}_x$: se $\bar{O}_x \neq \emptyset$ si seleziona un nodo $s \in \bar{O}_x$, altrimenti, per costruire un ciclo, si seleziona un qualsiasi nodo s che abbia almeno un arco uscente (e quindi almeno uno entrante). Si visita quindi $\bar{G}_x$ a partire da s, che ha sicuramente almeno un arco uscente; siccome ogni nodo tranne al più quelli in $\bar{D}_x$ ha almeno un arco uscente, in un numero finito di passi la visita:

- o raggiunge un nodo $t \in \bar{D}_x$;
- oppure raggiunge un nodo precedentemente visitato.

Nel primo caso si determina un cammino (semplice) P in $\bar{G}_x$ da un nodo $s \in \bar{O}_x$ a un nodo $t \in \bar{D}_x$; su questo cammino viene inviata una quantità di flusso pari a $\theta = \min\{\theta^x(P), e_x(s) - e_{x''}(s), e_{x''}(t) - e_x(t)\}$. Altrimenti si determina un ciclo (semplice) C in $\bar{G}_x$ e su C viene inviata una quantità di flusso pari a $\theta^x(C)$. In questo modo si ottiene un nuovo pseudoflusso x "più simile" ad x'' del precedente, in quanto u^x_{ij} diminuisce della quantità $\theta > 0$ per ogni $(i, j) \in P$ (C). In particolare, se si determina un ciclo si avrà $u^x_{ij} = 0$ per almeno un $(i, j) \in C$, e quindi tale arco non comparirà più in $\bar{G}_x$; se invece si determina un cammino allora si avrà che o $u^x_{ij} = 0$ per almeno un $(i, j) \in P$, oppure lo sbilanciamento rispetto ad x di almeno uno tra s e t diventa pari allo sbilanciamento rispetto ad x'', e quindi almeno un nodo tra s e t non comparirà più in $\bar{O}_x$ o in $\bar{D}_x$. Si noti che sugli archi di $\bar{A}^+_{x'}$ il flusso può solo aumentare, mentre sugli archi di $\bar{A}^-_{x'}$ il flusso può solo diminuire; siccome il flusso su (i, j) non viene più modificato non appena $x_{ij} = x''_{ij}$, nessun "nuovo" arco può essere creato in $\bar{G}_x$. Pertanto, ad ogni passo $\bar{G}_x$ è un sottografo del grafo residuo iniziale $G_{x'}$, e quindi un qualunque cammino (o ciclo) che viene utilizzato è aumentante rispetto allo pseudoflusso iniziale x'; è anche facile verificare che la quantità θ inviata lungo ogni cammino (o ciclo) è minore od uguale della capacità di quel cammino rispetto ad x'. Siccome ad ogni passo o si cancella almeno un arco da $\bar{G}_x$ o si cancella almeno un nodo da $\bar{O}_x \cup \bar{D}_x$, in al più $n + m$ passi tutti gli archi di $\bar{G}_x$ vengono cancellati e l'algoritmo termina. Ovviamente se $e_{x'} = e_{x''}$ in partenza si ha che $\bar{O}_x = \bar{D}_x$ durante tutto il processo, quindi non possono essere generati altro che cicli, che quindi sono al più m. ◇

Un elemento importante della tesi del Teorema 2.9, che è utile sottolineare, è che *tutti i cammini / cicli* P_i sono aumentanti rispetto al flusso iniziale x'; questo sarà cruciale per gli usi che faremo del Teorema. Un primo interessante corollario è il seguente:

Corollario 2.1. Qualsiasi pseudoflusso può essere costruito inviando opportune quantità di flusso lungo $k \leq n + m$ cammini o cicli (semplici), di cui al più m sono cicli. In particolare, tutti i cammini hanno come estremi nodi il cui deficit è diverso da zero. Di conseguenza, qualsiasi flusso ammissibile per un problema di (MCF) in cui $b = 0$ (detto *problema di circolazione*) può essere costruito inviando opportune quantità di flusso lungo $k \leq m$ cicli.

Dimostrazione Immediata dal Teorema 2.9 prendendo $x' = 0$ ed $x'' = x$. ◇

Il Corollario 2.1—ma, in effetti, il Teorema 2.9—mostra che è possibile rappresentare qualsiasi (pseudo)flusso come una composizione di un piccolo numero di sue "costituenti elementari", ossia flussi lungo cammini / cicli. Vedremo come tale risultato potrà essere esteso a classi di problemi più generali (cf. Teorema 3.4).

Tornando al principale argomento di questa sezione, il Teorema 2.9 consente di caratterizzare gli pseudoflussi, e quindi i flussi, "ottimi". Definiamo infatti *minimale* uno pseudoflusso x che abbia costo minimo tra tutti gli pseudoflussi aventi lo stesso vettore di sbilanciamento e_x; si noti che ogni soluzione ottima di (MCF) è un flusso ammissibile minimale, avendo costo minimo tra tutti gli (pseudo)flussi con $e_x = 0$.

Corollario 2.2. Uno pseudoflusso (flusso ammissibile) x è minimale (ottimo) se e solo se non esistono cicli aumentanti rispetto ad x il cui costo sia negativo.

Dimostrazione Se esiste un ciclo aumentante C rispetto ad x il cui costo $c(C)$ è negativo, allora x non è minimale: per ogni $0 < \theta \leq \theta(C, x)$, lo pseudoflusso $x(\theta) = x \oplus \theta C$ ha lo stesso vettore di sbilanciamento di x, ma $cx(\theta) < cx$ (si veda (2.19)).

Viceversa, sia x uno pseudoflusso tale che non esistono cicli aumentanti di costo negativo rispetto ad x, e supponiamo che x non sia minimale, ossia che esista uno pseudoflusso x' con lo stesso vettore di sbilanciamento tale che $cx' < cx$: per il Teorema 2.9 si ha $x' = x \oplus \theta_1 C_1 \oplus \ldots \oplus \theta_k C_k$, dove C_i sono cicli aumentanti rispetto a x; siccome tutti i θ_i sono numeri positivi, $cx' < cx$ e (2.19) implicano che $c(C_i) < 0$ per un qualche i, il che contraddice l'ipotesi. ◇

Ancora, vedremo come tale risultato potrà essere esteso a classi di problemi più generali, cf. Corollario 3.1. Nei prossimi paragrafi vedremo come la teoria appena sviluppata può essere utilizzata per progettare algoritmi risolutivi per (MCF).

2.5.2 Algoritmo basato su cancellazione di cicli

Il Corollario 2.2 suggerisce immediatamente un approccio risolutivo per (MCF): si determina un flusso ammissibile, e poi si utilizzano cicli aumentanti di costo negativo per ottenere flussi ammissibili di costo inferiore. L'algoritmo termina quando non esistono più cicli aumentanti di costo negativo: il flusso ammissibile così determinato è sicuramente di costo minimo. Quando si satura un ciclo aumentante C, inviando lungo i suoi archi un flusso pari a $\theta(C, x)$, si dice che si "cancella" il ciclo C in quanto esso non risulta più aumentante per il flusso $x(\theta)$; non si può però escludere che C possa tornare ad essere aumentante per flussi generati successivamente. Ciò porta alla definizione del seguente algoritmo.

```
procedure ( x , caso ) = Cancella-Cicli( G , c , b , u ) {
  ( x , caso ) = Flusso-Ammissibile( G , b , u );
  if( caso == "vuoto" ) then return;
  for( ; ; ) {
    ( C , θ ) = Trova-Ciclo( G , c , u , x );
    if( θ == 0 ) then break;
    x = Cambia-Flusso( x , C , θ );
    }
  }
```

Procedura 2.8: *Algoritmo basato sulla cancellazione di cicli*

La procedura *Flusso-Ammissibile* determina, se esiste, un flusso ammissibile: in tal caso restituisce *caso* = "ottimo" ed il flusso x, altrimenti restituisce *caso* = "vuoto". Una possibile implementazione di questa procedura è stata discussa nel §2.4.3. La procedura *Trova-Ciclo* determina, dato x, se esiste un ciclo aumentante rispetto ad x di costo negativo: in questo caso restituisce il ciclo individuato C, con il suo verso e la sua capacità $\theta = \theta(C, x)$, altrimenti restituisce $\theta = 0$. Dal Lemma 2.6 segue che il problema di determinare un ciclo aumentante di costo negativo in G rispetto ad x è equivalente al problema di determinare un ciclo orientato di costo negativo in G_x; tale problema può essere risolto in diversi modi, ad esempio utilizzando la procedura *SPT.L.queue*. In particolare, si può utilizzare la procedura per il problema dell'albero dei cammini minimi con radici multiple (si veda il §2.2.7) con insieme di radici $R = N$; ciò corrisponde ad aggiungere al grafo residuo una radice fittizia r e un

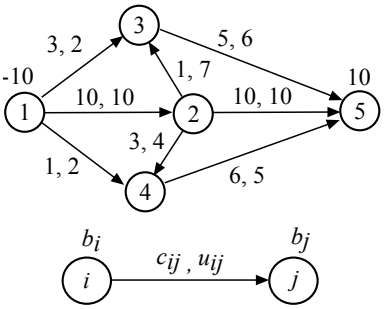
arco (r, j) di costo $c_{rj} = 0$ per ogni nodo $j \in N$. Una volta determinato il ciclo C ed il valore θ, la procedura *Cambia-Flusso* costruisce il nuovo flusso $x(\theta) = x \oplus \theta C$.

Esercizio 2.32. Si fornisca una descrizione formale, in pseudo-codice, delle procedure *Trova-Ciclo* e *Cambia-Flusso*.

Si noti come l'algoritmo basato sulla cancellazione di cicli sia un esempio di algoritmo di ricerca locale, in cui l'intorno di x è dato da tutti i flussi ottenibili da x inviando flusso lungo un ciclo aumentante semplice.

Esempio 2.23. Esecuzione dell'algoritmo *Cancella-Cicli*

Sia data l'istanza di (MCF) raffigurata qui accanto; un possibile esempio di funzionamento dell'algoritmo basato sulla cancellazione di cicli è mostrato in Figura 2.19. Una soluzione ammissibile può essere facilmente ottenuta inviando 10 unità di flusso da 1 a 5 lungo gli archi $(1, 2)$ e $(2, 5)$. I successivi cinque grafi (da sinistra a destra, dall'alto in basso) mostrano i passi svolti dall'algoritmo a partire da tale soluzione ammissibile, senza specificare come esattamente avviene la selezione tra i diversi cicli aumentanti di costo negativo (quando ve n'è più di uno). Per ogni iterazione vengono mostrati il valore del flusso su tutti gli archi in cui non è nullo, il ciclo selezionato (archi evidenziati) ed il suo verso (freccia tratteggiata), il valore della funzione obiettivo, $c(C)$ e $\theta(C, x)$. Il grafo in basso a destra fornisce invece la *dimostrazione* che la soluzione determinata è effettivamente ottima: in figura viene mostrato il grafo residuo G_x corrispondente a tale soluzione, con i relativi costi degli archi, ed il corrispondente albero dei cammini minimi con insieme di radici $R = N$, con le relative etichette (è facile verificare che le condizioni di Bellman sono rispettate). Dato che G_x ammette un albero dei cammini minimi, non esistono cicli orientati di costo negativo in G_x, e quindi non esistono cicli aumentanti rispetto ad x di costo negativo, il che garantisce che x sia un flusso ottimo.



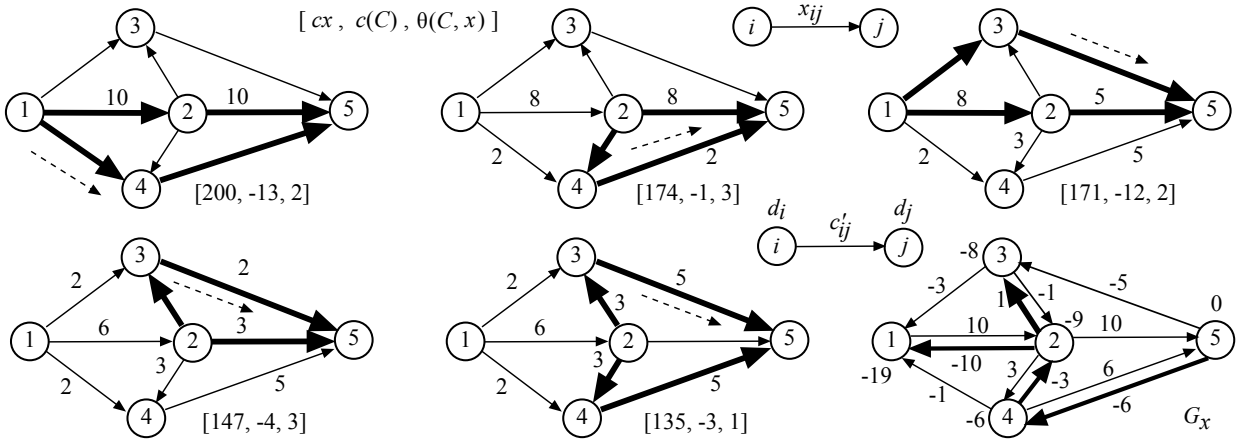


Figura 2.19: Esecuzione dell'algoritmo *Cancella-Cicli*

La correttezza dell'algoritmo *Cancella-Cicli* discende direttamente dal Corollario 2.2; analizziamo adesso la sua complessità. Innanzitutto notiamo che, se le capacità degli archi ed i bilanci dei nodi sono numeri interi, allora ad ogni passo dell'algoritmo il flusso x è intero. Infatti possiamo assumere che il flusso restituito da *Flusso-Ammissibile* sia intero (si veda il Teorema 2.7): se all'inizio di un'iterazione x è intero allora lo è anche $\theta = \theta(C, x)$, e quindi lo è anche il flusso al termine dell'iterazione. Sia adesso $\bar{u} = \max\{u_{ij} : (i, j) \in A\}$ la massima capacità degli archi, che assumiamo finita, e sia $\bar{c} = \max\{|c_{ij}| : (i, j) \in A\}$ il massimo valore assoluto dei costi degli archi: è facile verificare che il costo di qualunque soluzione ammissibile è compreso tra $m\bar{u}\bar{c}$ e $-m\bar{u}\bar{c}$. Se tutti i costi sono interi, il costo di qualsiasi ciclo aumentante utilizzato dall'algoritmo sarà sicuramente inferiore od uguale a -1. Siccome $\theta \geq 1$, ad ogni iterazione il valore della funzione obiettivo diminuisce di almeno un'unità, e quindi non possono essere eseguite più di $O(m\bar{u}\bar{c})$ iterazioni. Quindi, se i vettori b, c ed u hanno tutte componenti finite e intere, allora l'algoritmo termina. La complessità dipende allora dal modo in cui è realizzata la procedura *Trova-Ciclo*: poiché la procedura *SPT.L.queue* permette di determinare un ciclo di costo negativo in $O(mn)$, l'algoritmo *Cancella-Cicli* può sicuramente essere implementato in modo da avere complessità pseudopolinomiale $O(nm^2\bar{u}\bar{c})$.

L'analisi precedente indica come ci siano molte strade possibili per migliorare la complessità asintotica (e quindi, sperabilmente, anche l'efficienza in pratica) dell'algoritmo. Innanzitutto, l'algoritmo dovrebbe "decrementare di molto" il valore della funzione obiettivo ad ogni iterazione, in modo da diminuire il numero di iterazioni che compie. Per questo si potrebbe pensare di determinare il ciclo che minimizza il valore $c(C) \cdot \theta(C, x) < 0$ (di quanto la funzione obiettivo diminuisce), ma questo è difficile: infatti, anche solo determinare un ciclo che minimizzi il valore $c(C)$ è un problema $\mathcal{NP}$-arduo (si veda il §2.2.1). Esistono diversi modi possibili per affrontare il problema. Un'idea interessante sono le *tecniche di scalatura*, nelle quali i costi e le capacità vengono mappati su piccoli intervalli di numeri interi. Si immagini ad esempio di mappare tutte le capacità su $\{0, 1\}$; ciò significa che qualsiasi arco con capacità $< \bar{u}/2$ viene considerato avere capacità 0 (e quindi temporaneamente ignorato). Se sul grafo così modificato si determina un ciclo aumentante di costo negativo, quindi, la sua capacità reale è almeno pari a $\bar{u}/2$. Quando questo non è più possibile si raffina l'insieme delle capacità, ad esempio ponendolo pari a $\{0, 1, 2, 3\}$ (ossia considerando $\bar{u}/4$ come capacità minima di un ciclo); operando in modo opportuno, e similmente sui costi, si riescono ad ottenere valutazioni di complessità simili a quella precedentemente esposta, in cui i termini pseudopolinomiali $\bar{c}$ e $\bar{u}$ si riducono però a $\log \bar{c}$ e $\log \bar{u}$ (risultando quindi polinomiali).

Esercizio 2.33. Si sviluppino in dettaglio versioni dell'algoritmo basato sulla cancellazione di cicli con tecniche di scalatura sulle capacità e / o sui costi che abbiano complessità polinomiale.

Esistono anche idee diverse. Ad esempio si dimostra che il problema di determinare un ciclo di *lunghezza media* (lunghezza diviso per il numero di archi) minima è polinomiale, e che algoritmi di cancellazione di cicli che usano questi cicli possono essere implementati in modo da avere complessità *pienamente polinomiale* (ovvero in polinomiale in n ed m, e indipendente da $\bar{c}$ e $\bar{u}$). Per ulteriori dettagli si rimanda alla letteratura citata.

Un'altra possibilità di miglioramento consiste nel diminuire la complessità di determinazione di un ciclo di costo negativo. Una tecnica per determinare efficacemente cicli negativi è quella basata sulle *basi di cicli*. Abbiamo già notato che un qualsiasi ciclo può essere ottenuto mediante composizione di cicli semplici; diciamo che un insieme di cicli semplici (senza aver fissato un verso di percorrenza) è una base di cicli se ogni ciclo può essere ottenuto mediante composizione di cicli dell'insieme, scegliendo opportunamente il loro verso di percorrenza. Una base di cicli è facilmente ottenibile mediante un albero di copertura $T = (N, A_T)$ del grafo G: ogni arco $(i, j) \in A \setminus A_T$ induce su T un ciclo $C_T(i, j)$ (si veda l'Appendice B), e l'insieme

$$B(T) = \{ C_T(i, j) : (i, j) \in A \setminus A_T \}$$

è la base di cicli indotta da T. Una base di cicli ha pertanto cardinalità $|B(T)| = m - n + 1$. La proprietà fondamentale delle basi di cicli è descritta dal seguente teorema, di cui omettiamo la dimostrazione.

Teorema 2.10. Data una base di cicli $B(T)$, se esiste un ciclo C di costo negativo, allora esiste un ciclo $C' \in B(T)$ di costo negativo.

Questa proprietà permette di costruire algoritmi che utilizzano le basi di cicli per determinare l'esistenza di un ciclo ammissibile di costo negativo. Ad ogni iterazione si ha un albero di copertura T ed un flusso ammissibile x con la proprietà che tutti gli archi $(i, j) \notin T$ sono o saturi ($x_{ij} = u_{ij}$) oppure vuoti ($x_{ij} = 0$): di conseguenza, ogni ciclo $C_T(i, j) \in B(T)$ può essere aumentante solo in uno dei due versi (concorde con (i, j) se l'arco è vuoto, discorde con (i, j) se l'arco è saturo). Con questi elementi, si può dimostrare che se nessuno dei cicli di $B(T)$, scelti con l'unico verso che può renderli aumentanti, ha costo negativo, allora non esistono cicli aumentanti di costo negativo e l'algoritmo termina dichiarando che x è ottimo. Altrimenti, viene scelto un ciclo di costo negativo $C_T(i, j) \in B(T)$ (col verso opportuno) e viene aggiornato il flusso lungo di esso. L'aggiornamento rende vuoto o saturo uno degli archi di $C_T(i, j)$ in $B(T)$: tale arco viene quindi eliminato da T e sostituito con (i, j), in modo da considerare nell'iterazione successiva una diversa base di cicli. Si noti che può accadere $\theta(C_T(i, j), x) = 0$, ossia che uno degli archi discordi del ciclo è già vuoto oppure uno degli archi concordi è già saturo: in questo caso il flusso non cambia e cambia solamente la base $B(T)$. Ad ogni

iterazione, quindi, ci si sposta da una base $B(T)$ ad una base $B(T')$ "adiacente", ossia tale che T' è ottenuto da T mediante lo scambio di due archi. È possibile dimostrare che, dopo un numero finito di iterazioni, si ottiene un albero di copertura T tale che tutti i cicli $C_T(i,j) \in B(T)$ (col verso opportuno) hanno costo non negativo, e quindi il flusso x è ottimo. Gli algoritmi basati sulle basi di cicli sono conosciuti come "Algoritmi del Simplesso su Reti", perché si possono interpretare come versioni specializzate (altamente efficienti) degli algoritmi del simplesso per la *PL* che vedremo nel §3.3. Queste relazioni sono brevemente discusse nel §3.3.3.

2.5.3 Algoritmo basato su cammini minimi successivi

Un approccio sostanzialmente diverso è quello dell'algoritmo dei *cammini minimi successivi*, che mantiene ad ogni passo uno pseudoflusso *minimale* x e determina un *cammino aumentante di costo minimo* da un nodo $s \in O_x$ a un nodo $t \in D_x$ per diminuire, al minor costo possibile, lo sbilanciamento di x. L'uso di cammini aumentanti di costo minimo permette di conservare la minimalità degli pseudoflussi:

Teorema 2.11. Sia x uno pseudoflusso minimale, e sia P un cammino aumentante rispetto a x avente costo minimo tra tutti i cammini che uniscono un dato nodo $s \in O_x$ ad un dato nodo $t \in D_x$: allora, comunque si scelga $\theta \leq \theta(P, x)$, $x(\theta) = x \oplus \theta P$ è uno pseudoflusso minimale.

Dimostrazione Fissato $\theta \leq \theta(P, x)$, sia x' un qualsiasi pseudoflusso con vettore di sbilanciamento $e_{x(\theta)}$. Il Teorema 2.9 mostra che esistono k cammini aumentanti $P_1, \ldots, P_k$ da s a t (in quanto s e t sono gli unici nodi in cui e_x ed $e_{x(\theta)}$ differiscono) e $h \leq m$ cicli aumentanti $C_1, \ldots, C_h$ rispetto ad x tali che

$$x' = x \oplus \theta_1 P_1 \oplus \ldots \oplus \theta_k P_k \oplus \theta_{k+1} C_1 \oplus \ldots \oplus \theta_{k+h} C_h \ .$$

In più, deve sicuramente essere $\theta_1 + \ldots + \theta_k = \theta$. Siccome x è minimale, ciascuno degli h cicli aumentanti deve avere costo non negativo; inoltre, siccome P ha costo minimo tra tutti i cammini aumentanti da s a t, si ha $c(P) \leq c(P_i)$, $i = 1, \ldots, k$. Di conseguenza

$$cx' = cx + \theta_1 c(P_1) + \ldots + \theta_k c(P_k) + \theta_{k+1} c(C_1) + \ldots + \theta_{k+h} c(C_h) \geq cx + \theta c(P) = cx(\theta) \ ,$$

e quindi $x(\theta)$ è minimale. ◇

Con un'opportuna scelta di θ, l'operazione di composizione tra lo pseudoflusso x ed il cammino P permette di diminuire lo sbilanciamento complessivo: infatti, è immediato verificare che per

$$\theta = \min\{\, \theta(P, x) \,,\, e_x(s) \,,\, -e_x(t) \,\} > 0 \tag{2.20}$$

(si ricordi che $e_x(s) > 0$ e $e_x(t) < 0$), $x(\theta)$ è uno pseudoflusso (minimale) con sbilanciamento complessivo $g(x(\theta)) = g(x) - \theta < g(x)$. Questa scelta di θ corrisponde alla maggior diminuzione possibile dello sbilanciamento complessivo corrispondente al cammino P ed allo pseudoflusso x. Ciò conduce direttamente alla definizione del seguente algoritmo:

```
procedure (x, caso) = Cammini-Minimi-Successivi(G, c, b, u) {
  x = Inizializza(c, u); caso = "ottimo";
  while(g(x) ≠ 0) do
    (p, d) = Trova-Cammino-Minimo(G_x, O_x, D_x);
    t = argmin {d[i] : i ∈ D_x};
    if(d[t] == ∞) then { caso = "vuoto"; break; }
    x = Aumenta-Flusso(x, p, t);
    }
  }
```

Procedura 2.9: *Algoritmo basato su cammini minimi successivi*

La funzione *Inizializza* costruisce e ritorna uno pseudoflusso x minimale: un semplice modo per implementare tale procedura è quello di porre, per ogni $(i, j) \in A$, $x_{ij} = 0$ se $c_{ij} \geq 0$, e $x_{ij} = u_{ij}$ altrimenti. In tal modo i costi degli archi in G_x sono tutti non negativi, e quindi non esistono cicli orientati in G_x (cicli aumentanti rispetto ad x in G) di costo negativo, per cui x è minimale. In effetti è anche facile verificare che lo pseudoflusso così costruito ha costo minimo tra tutti i possibili pseudoflussi per il dato vettore di capacità u. Si noti che questa fase di inizializzazione richiede che non esistano archi con costo negativo e capacità infinita.

La funzione *Trova-Cammino-Minimo* determina un albero dei cammini minimi con insieme di nodi radice O_x ed insieme di destinazioni D_x (si veda il §2.2.7) su G_x; in altri termini, se $|\,O_x\,| > 1$ si aggiunge a G_x un nodo "radice" r collegato a tutti i nodi in O_x con archi a costo nullo, e poi si risolve un problema di albero dei cammini minimi di radice r sul grafo così ottenuto (altrimenti basta usare come radice l'unico nodo in O_x). La procedura termina sicuramente: infatti x è minimale, e quindi non esistono cicli negativi in G_x. Si seleziona quindi il nodo $t \in D_x$ con etichetta $d[\,t\,]$ minima, corrispondente al più corto tra tutti i cammini minimi: se $d[\,t\,] = \infty$ allora non esiste alcun cammino aumentante da qualsiasi nodo in O_x a qualsiasi nodo in D_x e *Cammini-Minimi-Successivi* termina restituendo *caso* = "vuoto", in quanto non esiste nessuna soluzione ammissibile per il problema di flusso di costo minimo.

Esercizio 2.34. Si dimostri l'affermazione precedente (suggerimento: si consideri il §2.4.3).

Se invece $d[\,t\,] < \infty$ allora esiste un cammino aumentante P in $p[\cdot]$ che unisce un nodo $s \in O_x$ a t: la procedura *Aumenta-Flusso* determina la quantità di flusso θ, definita in (2.20), che deve essere inviata lungo P ed implementa l'operazione di composizione $x = x \oplus \theta P$, in modo simile alla *Aumenta-Flusso* utilizzata per l'algoritmo *Cammini-Aumentanti*. Si noti che se $\theta = e_x(\,s\,)$ allora il nodo s risulterà bilanciato rispetto al nuovo flusso, ed analogamente per il nodo t se $\theta = -e_x(\,t\,)$; altrimenti, θ è determinato dalla capacità del cammino, il che significa che almeno un arco di P diviene saturo oppure vuoto.

Siccome l'algoritmo usa sempre cammini aumentanti di costo minimo, per il Teorema 2.11 ad ogni passo lo pseudoflusso x è minimale: quindi, se l'algoritmo termina con $g(\,x\,) = 0$, allora x è un flusso ottimo. La terminazione dell'algoritmo può essere facilmente provata nel caso in cui b e u siano interi. Infatti, in questo caso lo pseudoflusso iniziale è anch'esso intero, e quindi lo è la quantità θ a quell'iterazione, e quindi lo è anche lo pseudoflusso x ottenuto al termine dell'iterazione. Di conseguenza, ad ogni iterazione x è intero, $\theta \geq 1$ e $g(x)$ diminuisce di almeno un'unità, e quindi l'algoritmo termina in un numero finito di iterazioni. Da questa analisi segue:

Teorema 2.12. Se le capacità degli archi ed i bilanci dei nodi sono interi, allora per qualsiasi vettore di costi associati agli archi esiste una soluzione ottima intera per il problema (MCF).

Questa *proprietà di integralità* è molto importante per le applicazioni: si pensi ad esempio al caso in cui il flusso su un arco rappresenta il numero di camion, o carrozze ferroviarie, o containers, o ancora pacchetti in una rete di comunicazione. In effetti, una delle motivazioni principali per cui si studia (MCF) risiede nel fatto che essa costituisce la più grande ed utile classe di problemi che possiedono questa proprietà, come discusso nel Capitolo 4 ed in quelli successivi.

Esempio 2.24. Esecuzione dell'algortimo *Cammini-Minimi-Successivi*

Consideriamo di nuovo l'istanza del problema (MCF) dell'Esempio 2.23; il funzionamento dell'algoritmo basato su cammini minimi successivi è mostrato in Figura 2.20. Siccome tutti i costi sono non negativi, *Inizializza* costruisce uno pseudoflusso iniziale identicamente nullo. Le iterazioni procedono da sinistra a destra: in alto viene mostrato il grafo residuo G_x, e sotto lo pseudoflusso ottenuto al termine dell'iterazione. In G_x non sono riportati (per chiarezza di visualizzazione) i costi degli archi, ma è evidenziato l'albero dei cammini minimi con i valori delle corrispondenti etichette; è inoltre mostrato il valore θ del flusso inviato lungo il relativo cammino aumentante da 1 a 5. I valori del flusso e degli sbilanciamenti sono mostrati solamente per quegli archi/nodi in cui sono diversi da zero. Nella quarta iterazione tutti i nodi hanno sbilanciamento nullo, e la soluzione è ottima.

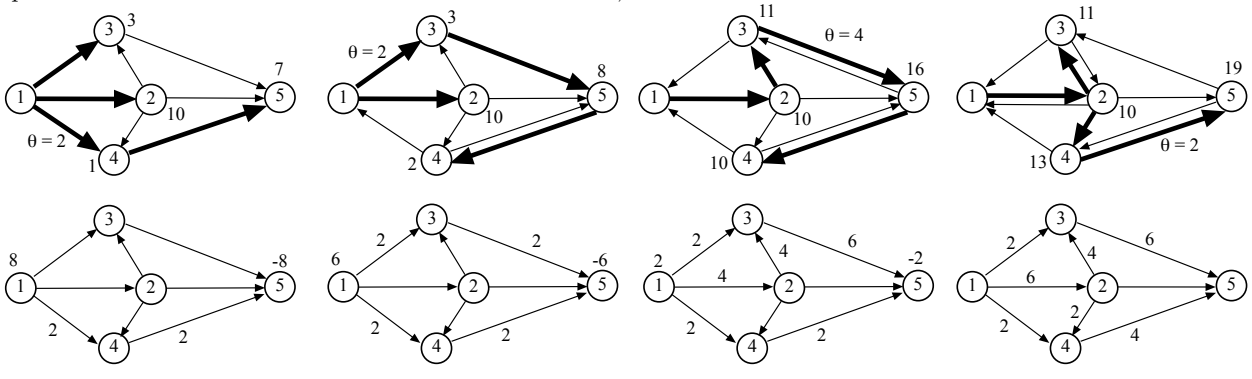
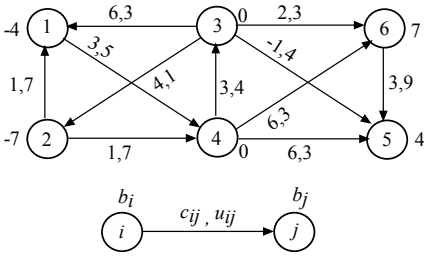


Figura 2.20: Esecuzione dell'algoritmo basato su cammini minimi successivi

Il prossimo esempio di esecuzione dell'algoritmo *Cammini-Minimi-Successivi* mostra invece cosa accade quando l'istanza di (MCF) non ha soluzione ammissibile.

Esempio 2.25. *Cammini-Minimi-Successivi* nel caso vuoto

Si vuole risolvere l'istanza di (MCF) in figura utilizzando l'algoritmo dei cammini minimi successivi. Ad ogni iterazione si seleziona come destinazione il nodo con etichetta minore, ed a parità di etichetta quello con indice minore.



L'esecuzione dell'algoritmo è mostrata in Figura 2.21. L'algoritmo determina inizialmente uno pseudoflusso minimale saturando tutti gli archi di costo negativo (il solo $(3, 5)$) e vuotando tutti gli altri. L'algoritmo esegue quindi quattro iterazioni, illustrate dalle figure (dall'alto in basso). In ogni figura, a destra è mostrato lo pseudoflusso (minimale) all'inizio dell'iterazione con i relativi sbilanciamenti, mentre a sinistra è mostrato l'albero dei cammini minimi con insieme di radici O_x (archi evidenziati, sul grafo originale), con le relative etichette. È inoltre indicata la quantità di flusso θ inviata sul cammino selezionato, ossia quello che collega il nodo di etichetta minima tra quelli con sbilanciamento negativo alla corrispondente "radice" dell'albero dei cammini minimi, con sbilanciamento positivo.

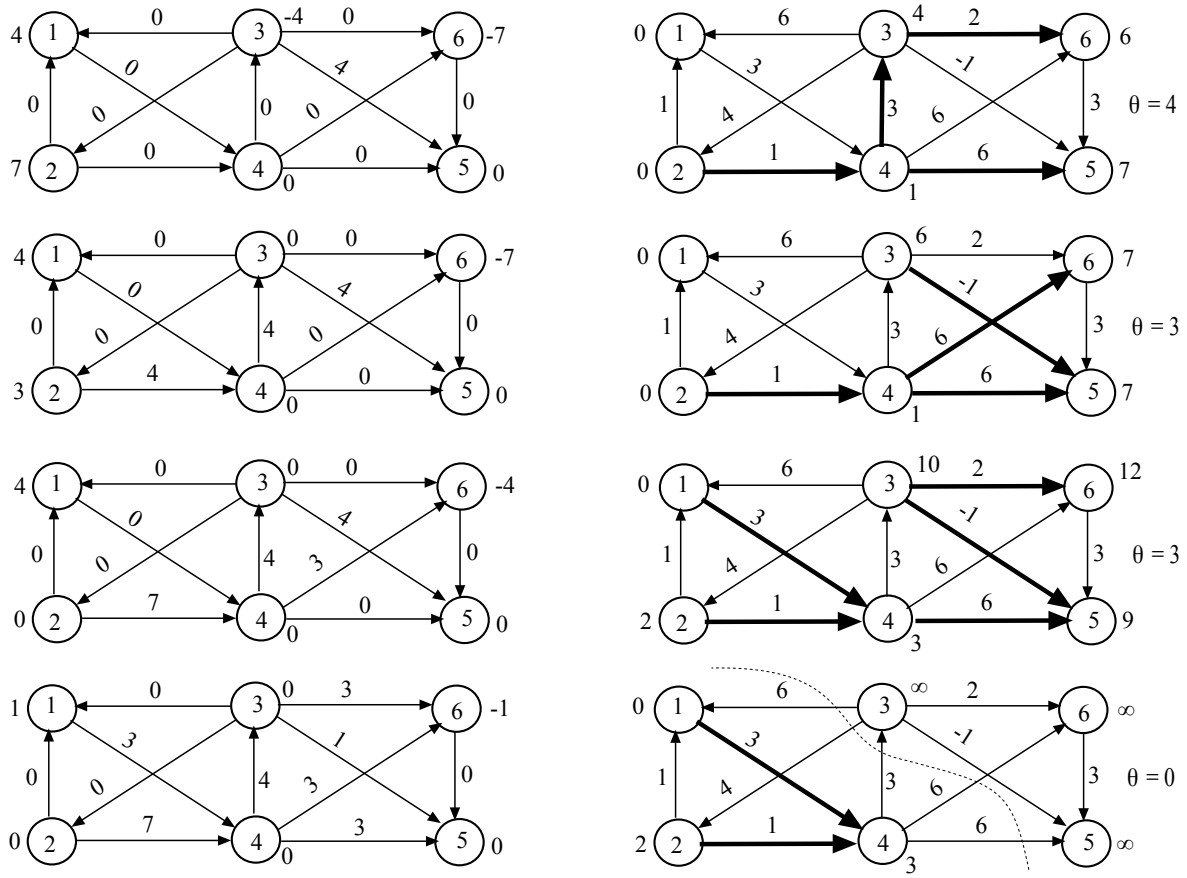


Figura 2.21: Esecuzione dell'algoritmo basato su cammini minimi successivi

Nell'ultima iterazione non si determina alcun cammino aumentante tra i nodi che hanno ancora sbilanciamento positivo (1) e quelli con sbilanciamento negativo (6); pertanto, non esiste nessuna soluzione ammissibile. Ciò è facile da verificare considerando il taglio saturo $(N', N'') = (\{1, 2, 4\}, \{3, 5, 6\})$ evidenziato in basso a destra, che ha capacità $u(N', N'') = u_{43} + u_{45} + u_{46} = 4 + 3 + 3 = 10$, ma separa due rive che hanno deficit complessivo pari ad 11. Infatti, ad esempio, $\sum_{i \in N'} b_i = b_1 + b_2 + b_4 = -4 - 7 + 0 = -11$. Pertanto è ovvio che non sia possibile determinare alcun flusso ammissibile.

È però possibile mostrare che, grazie alla specifica regola di selezione del particolare cammino ad ogni iterazione, lo pseudoflusso ottenuto è quello di costo minimo tra tutti quelli che massimizzano la quantità di flusso inviata dalle sorgenti alle destinazioni, ossia tra quelli che violano il meno possibile i vincoli di conservazione del flusso (in altri termini, è un flusso massimo di costo minimo); si veda il §2.6.2.

Per analizzare la complessità dell'algoritmo, si consideri che lo sbilanciamento complessivo dello pseudoflusso x costruito da *Inizializza* è limitato superiormente da $\overline{g} = \sum_{(i,j)\in A:\, c_{ij}<0} u_{ij} + \sum_{i\in N:\, b_i>0} b_i$; siccome $g(x)$ diminuisce di almeno un'unità ad ogni iterazione, il numero di iterazioni non potrà eccedere $\overline{g}$. È facile vedere che tutte le operazioni effettuate durante una singola iterazione hanno complessità $O(n)$, esclusa l'invocazione della procedura *Trova-Cammino-Minimo*: se utilizziamo l'algoritmo *SPT.L.queue*, che ha complessità $O(mn)$, la procedura *Cammini-Minimi-Successivi* risulta avere complessità pseudopolinomiale $O(\overline{g}mn)$.

Analogamente al caso dell'algoritmo *Cancella-Cicli*, ci sono due modi per migliorare la complessità asintotica (e quindi, sperabilmente, anche l'efficienza in pratica) dell'algoritmo: utilizzare cammini che portano "molto flusso", in modo da diminuire il numero di iterazioni necessarie ad ottenere un flusso ammissibile, oppure utilizzare algoritmi più efficienti di *SPT.L.queue* per calcolare il cammino minimo. Per il primo approccio si possono utilizzare, ad esempio, le tecniche di scalatura già accennate in precedenza. Per il secondo è necessario (almeno in teoria) utilizzare algoritmi tipo *SPT.S* su grafi con costi non negativi. Questo è in effetti possibile modificando opportunamente i costi mediante un vettore $\pi \in \mathbb{R}^n$ di *potenziali* dei nodi, mediante i quali si può definire il *costo ridotto* $c^\pi_{ij} = c_{ij} + \pi_i - \pi_j$ di ogni arco (i, j) rispetto a π.

Teorema 2.13. Uno pseudoflusso x è minimale se e solo se esiste un vettore di potenziali π tale che

$$\begin{array}{lcl} x_{ij} > 0 & \Longrightarrow & c^\pi_{ij} \le 0 \\ x_{ij} < u_{ij} & \Longrightarrow & c^\pi_{ij} \ge 0 \end{array} \quad . \tag{2.21}$$

Dimostrazione È facile verificare che, comunque si scelga un vettore di potenziali π ed un ciclo C (con un verso di percorrenza), il costo del ciclo $c(C)$ è uguale al suo *costo ridotto* $c^\pi(C)$ (definito come in (2.18) usando i costi ridotti al posto dei costi): infatti, per ciascun nodo i del ciclo il corrispondente potenziale π_i appare due volte nella sommatoria, una per ciascuno dei due archi incidenti, e sempre con coefficienti opposti. Si consideri adesso il grafo residuo G_x con i costi definiti da c^π. Se (2.21) è vera, allora i costi (ridotti) su tutti gli archi di G_x sono non negativi. Quindi il costo ridotto di qualsiasi ciclo in G_x è non negativo, ovvero lo è il costo di qualsiasi ciclo aumentante, e quindi x è minimale. Viceversa, se x è minimale allora non esistono cicli aumentanti di costo negativo rispetto a x; esiste quindi un albero dei cammini minimi su G_x (con i costi originali, e, ad esempio, insieme di radici $R = N$), con il corrispondente vettore di etichette d che rispetta le condizioni di Bellman (2.7): ciò significa che

$$c_{ij} + d_i - d_j \ge 0 \qquad (i, j) \in A_x$$

ed è facile verificare che questo implica (2.21) prendendo $\pi = d$. ◇

Il teorema precedente suggerisce una variante dell'algoritmo che usa un vettore di potenziali π, inizializzato come $\pi = 0$, e risolve il problema (SPT) in G_x utilizzando i costi ridotti c^π associati agli archi invece dei costi originari. Si noti che alla prima iterazione i due insiemi di costi sono uguali, ed abbiamo già notato come in quel caso il grafo residuo abbia solamente archi di costo non negativo, per cui è possibile utilizzare algoritmi *SPT.S* per determinare un albero dei cammini minimo, con complessità inferiore a $O(mn)$. Oltre al cammino P utilizzato per inviare il flusso, l'algoritmo *SPT* restituisce anche un vettore di etichette d: è facile dimostrare che se x ed π rispettano (2.21) (all'inizio dell'iterazione), allora anche $x \oplus \theta P$ e $\pi + d$ le rispettano (alla fine dell'iterazione).

Esercizio 2.35. Dimostrare l'affermazione precedente.

Pertanto, anche per questa variante dell'algoritmo si ottiene, tramite il Teorema 2.13, che tutti gli pseudoflussi generati sono minimali, e quindi che l'algoritmo è corretto. Il vantaggio è che ad ogni passo si calcola un albero dei cammini minimi su un grafo con archi di costo non negativo, e questo può essere ottenuto in $O(n^2)$ oppure $O(m \log n)$.

Esercizio 2.36. Un ulteriore vantaggio di *SPT.S* è che sarebbe possibile, in linea di principio, interrompere la computazione non appena un nodo j con $e_j < 0$ viene estratto da Q, evitando così di esplorare inutilmente parti del grafo. In questo caso, però, le etichette dei nodi che non sono ancora stati estratti da Q al momento della terminazione anticipata possono non soddisfare le condizioni di Bellman, e quindi $\pi + d$ potrebbe non soddisfare le condizioni (2.21). Si discuta come si possa gestire adeguatamente questa occorrenza.

Oltre a suggerire implementazioni potenzialmente più efficienti dell'algoritmo, che possono essere utili in casi specifici (si veda ad esempio il §(2.6.2), queste considerazioni sono interessanti perché introducono ad un'ampia classe di algoritmi per (MCF), detti *primali-duali*, che sono tra i più efficienti in pratica. Questi sono basati sull'osservazione che, in uno pseudoflusso minimale, il costo ridotto di tutti gli archi (i, j) che non sono né saturi né vuoti ($0 < x_{ij} < u_{ij}$) deve essere zero; viceversa, dato un vettore di potenziali π, il valore di x_{ij} su qualsiasi arco (i, j) tale che $c^{\pi}_{ij} = 0$ può essere fissato arbitrariamente, ottenendo sempre uno pseudoflusso minimale. Infatti, si può facilmente osservare che, nella variante sopra discussa, i costi ridotti calcolati rispetto a $\pi + d$ sono nulli per tutti gli archi dell'albero dei cammini minimi individuato.
Gli algoritmi di questa classe procedono quindi mantenendo una coppia (x, π) che rispetta (2.21). Nella *fase primale* si mantiene π fisso, e si cerca di modificare x in modo tale da ottenere un flusso ammissibile; per questo si usano tecniche per il problema di flusso massimo, ad esempio analoghe a quelle studiate nel §2.4.4. Se si ottiene un flusso ammissibile l'algoritmo termina; altrimenti la fase primale determina informazione (ad esempio un opportuno taglio) che permette di modificare π nella *fase duale*, creando nuovi archi a costo ridotto nullo ove sia possibile inviare liberamente il flusso. Il nome di questi algoritmi deriva dal fatto che i potenziali π sono variabili del *duale* di (MCF), concetto che verrà discusso in dettaglio nel Capitolo 3.

Nella letteratura scientifica sono stati proposti moltissimi algoritmi per il flusso di costo minimo: alcuni di questi algoritmi possono essere considerati versioni (molto) raffinate di quelli illustrati in questo corso, mentre altri algoritmi sono basati su idee completamente diverse. Per maggiori dettagli su questi temi si rinvia alla letteratura indicata ed a corsi successivi dell'area di Ricerca Operativa.

2.6 Problemi di accoppiamento

Presentiamo adesso un'ulteriore sottoclasse importante dei problemi di (MCF), che abbiamo già introdotto nel §1.2.4. In questo caso, la particolarità è, come vedremo, la topologia del grafo. In effetti la definizione naturale del problema è su un grafo *non orientato* e *bipartito* $G = (O \cup D, E)$, dove $O = \{1, \ldots, n\}$ è l'insieme dei *nodi origine*, $D = \{n+1, \ldots, n+d\}$ è l'insieme dei nodi destinazione, e $E \subseteq O \times D$, con $|A| = m$, è l'insieme dei lati, ai quali possono essere associati costi c_{ij}. Non è restrittivo supporre $n \leq d$ (altrimenti basta scambiare O con D).

Un *accoppiamento* (*matching*) M è un sottoinsieme di lati che non hanno nodi in comune. I lati in M sono detti *interni*, mentre i lati in $A \setminus M$ sono detti *esterni*. Dato un accoppiamento M, un nodo i è *esposto* rispetto a M se nessun lato di M incide in i, altrimenti i è detto *accoppiato*; indicheremo con O_M e D_M gli insiemi dei nodi rispettivamente in O e D che sono esposti. Nel caso in cui $|O| = |D|$, cioè $d = n$, M è un *accoppiamento perfetto* (o *assegnamento*) se nessun nodo è esposto, ovvero se $|M| = n$. Un esempio è fornito in Figura 2.22. La *cardinalità* di un accoppiamento M è $|M|$, mentre il costo $C(M)$ di un accoppiamento M è la somma dei costi dei lati di M (si assume $C(\emptyset) = 0$). Dato un accoppiamento $M \neq \emptyset$, il lato $\{i, j\} \in M$ di costo massimo è detto *lato bottleneck* (collo di bottiglia) di M; il valore $V(M) = \max\{c_{ij} : \{i, j\} \in M\}$ è detto il *valore bottleneck* di M.

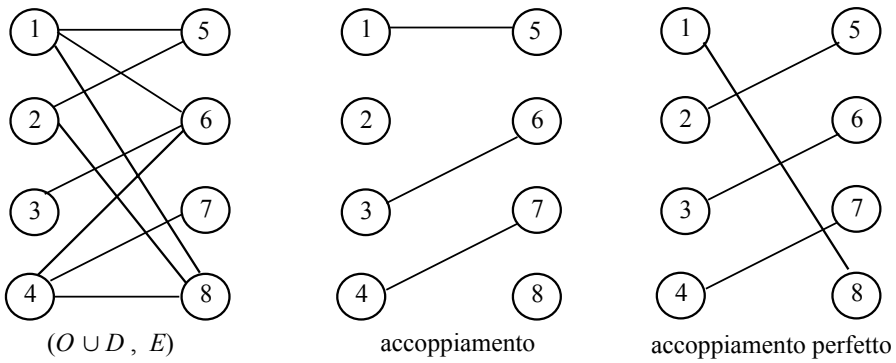


Figura 2.22: Esempi di accoppiamenti

Nel seguito studieremo i seguenti problemi:

1. *Accoppiamento di Massima Cardinalità* in G.
2. *Assegnamento di Costo Minimo*: si vuole determinare, tra tutti gli accoppiamenti perfetti in G, uno che abbia di costo minimo.
3. *Assegnamento di Massima Cardinalità Bottleneck*: si vuole determinare, tra tutti gli accoppiamenti di massima cardinalità in G, uno che abbia valore bottleneck minimo, cioè tale che il massimo costo degli archi sia minimo.

2.6.1 Accoppiamento di massima cardinalità

Il problema di *Accoppiamento di Massima Cardinalità* (MCM, da *Maximum Cardinality Matching*) in un grafo $G = (O \cup D, E)$ può essere trasformato in un problema equivalente di (MF) con più sorgenti e pozzi sul grafo orientato $\bar{G} = (O \cup D, A)$ dove A contiene un arco (i, j) per ogni lato $\{i, j\}$ di E, orientato dal nodo in O a quello in D, con capacità superiore $u_{ij} = 1$. O è l'insieme delle sorgenti, D è l'insieme dei pozzi, ed ogni sorgente/pozzo può immettere nella/prelevare dalla rete un'unità di flusso; equivalentemente si possono aggiungere a $\bar{G}$ una "super sorgente" s ed un "super pozzo" t, collegati rispettivamente a tutti i nodi di O e D da archi di capacità unitaria, e considerare il problema di (MF) da s a t. È facile verificare che i lati in G corrispondenti all'insieme degli archi saturi in qualunque flusso ammissibile (intero) x in $\bar{G}$ (escludendo quindi quelli incidenti in s e t, se costruiti) forma un accoppiamento M in G la cui cardinalità è pari al valore v del flusso. Viceversa, da un qualunque accoppiamento M si costruisce un flusso ammissibile. Nell'esempio in Figura 2.23, relativo al grafo G di Figura 2.22, è mostrato in (a) un accoppiamento M con $|M| = 3$, ed in (b) il corrispondente flusso ammissibile x su $\bar{G}$ con valore del flusso $v = 3$ (sono indicati solo i flussi diversi da zero).

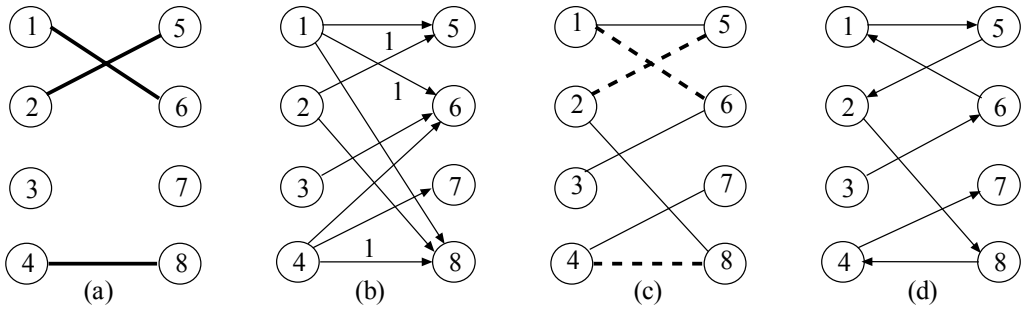


Figura 2.23: Flussi, accoppiamenti e cammini (alternanti) aumentanti

È quindi possibile risolvere il problema (MCM) in G applicando un qualsiasi algoritmo per il problema (MF) con più sorgenti e pozzi in $\bar{G}$. Data la particolare struttura del problema, però, alcune operazioni degli algoritmi possono essere implementate in maniera più efficiente, o hanno un particolare significato che è possibile sfruttare ai fini algoritmici. Si consideri ad esempio il concetto di cammino aumentante sul grafo $\bar{G}$ rispetto ad un qualche flusso x che rappresenta un accoppiamento M, ossia tale che $x_{ij} = 1$ per $(i, j) \in A$ se e solo se $\{i, j\} \in M$. Un arco $(i, j) \in A$ è saturo se e solo se il corrispondente lato $\{i, j\} \in E$ è interno ad M. Siccome il grafo è bipartito, i nodi di qualsiasi cammino su $\bar{G}$ devono appartenere alternativamente ad O ed a D. Ma tutti i lati $\{i, j\} \notin M$ su G corrispondono ad archi vuoti su $\bar{G}$: quindi, tali archi possono essere utilizzati in un cammino aumentante solamente in modo concorde col loro verso, ossia da un nodo di O ad un nodo di D. Viceversa, tutti i lati $\{i, j\} \in M$ su G corrispondono ad archi saturi su $\bar{G}$: quindi, tali archi possono essere utilizzati in un cammino aumentante solamente in modo discorde al loro verso, ossia da un nodo di D ad un nodo di O. Da tutto questo segue che un cammino aumentante su $\bar{G}$ rispetto ad un flusso x corrisponde ad un *cammino alternante* su G rispetto all'accoppiamento M, ossia un cammino formato alternativamente da archi esterni ed archi interni rispetto a M.

Non tutti i cammini alternanti su G rappresentano però cammini aumentanti su $\bar{G}$; affinché questo accada, occorre anche che il cammino parta da un'origine esposta e termini in una destinazione esposta, ed in questo caso il cammino alternante è detto anche *aumentante*. Infatti, le origini / destinazioni

esposte sono quelle per cui non transita ancora nessun flusso: siccome ogni origine / destinazione ha "capacità" unitaria, le origini esposte sono i nodi nell'insieme S_x delle sorgenti "attive" e le destinazioni esposte sono i nodi nell'insieme T_x dei pozzi "attivi" (si veda il §2.4.3). Per questo, qualsiasi cammino aumentante su $\bar{G}$ deve avere come primo nodo un'origine esposta e come ultimo nodo una destinazione esposta. Esiste quindi una corrispondenza biunivoca tra cammini aumentanti su $\bar{G}$ e *cammini alternanti aumentanti* su G. Un cammino alternante P su G è aumentante se, detti $P^+ = P \setminus M$ e $P^- = M \cap P$ l'insieme degli archi esterni e quello degli archi interni di P, si ha $|\,P^+\,| - |\,P^-\,| = 1$, ossia gli archi esterni sono esattamente uno in più di quelli interni.

Esempio 2.26. Cammini alternanti aumentanti
Le affermazioni precedenti possono essere facilmente verificate nell'esempio in Figura 2.23; in (c) e (d) sono mostrati rispettivamente un cammino alternante aumentante su G rispetto all'accoppiamento M ed un cammino aumentante su $\bar{G}_x$ rispetto al flusso x.

Si noti che, dato un cammino (alternante) aumentante, la capacità del cammino è sempre 1. Questo corrisponde al fatto che, in questo caso, l'operazione di composizione $x' = x \oplus P'$, dove P' è un cammino aumentante su $\bar{G}$, corrisponde a

$$M' = M \oplus P = M \setminus P^- \cup P^+$$

dove P è un cammino alternante aumentante su G: in altre parole, l'operazione di composizione corrisponde a togliere da M i lati interni di P ed aggiungere quelli esterni di P. Siccome $|P^+| = |P^-| + 1$, si ha che $|\,M \oplus P\,| = |\,M\,| + 1$; infatti, il nuovo flusso x' ha valore $v' = v + 1$.

Esempio 2.27. Operazione di composizione
Proseguendo l'esempio di Figura 2.23, ed applicando l'operazione di composizione aull'accoppiamento M mostrato in (a) ed al cammino P mostrato in (c), si ottiene il nuovo accoppiamento $M' = \{\,(1,5)\,,\,(2,8)\,,\,(3,6)\,,\,(4,7)\,\}$. È immediato verificate che il nuovo accoppiamento corrisponde al flusso che si ottiene dal flusso mostrato in (b) inviando un'unità di flusso lungo il cammino aumentante mostrato in (d).

Con queste notazioni, possiamo costruire una versione specializzata dell'Algoritmo 2.4.2 per risolvere il problema (MCM).

```
procedure M = Accoppiamento-MaxCard( O , D , E )  {
  M = ∅;
  while( O_M ≠ ∅ ) do {
    p = Cammino-Aumentante( O , D , E , M );
    if( p[i] == 0 per ogni i ∈ D_M ) then break;
    seleziona t ∈ D_M tale che p[t] ≠ 0;
    M = Cambia-Accoppiamento( M , p , t );
    }
  }
```

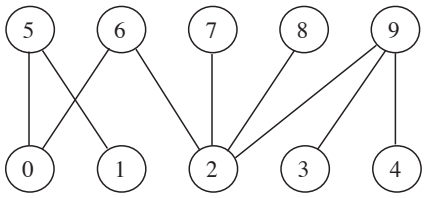
Procedura 2.10: Algoritmo *Accoppiamento-MaxCard*

L'inizializzazione, $M = \emptyset$, corrisponde a scegliere $x = 0$ come flusso iniziale. La procedura *Cammino-Aumentante* determina, se esiste, un cammino alternante aumentante: per questo è sufficiente visitare il grafo bipartito G partendo dai nodi di O_M e visitando alternativamente archi esterni e interni, il che corrisponde alla procedura *Visita* con semplici modifiche. Si noti che, rispetto al caso del (MF), il controllo di ammissibilità di un arco è più semplice, e non è necessario determinare la capacità del cammino. Alla fine visita si controlla se si è raggiunto almeno un nodo in D_M: se questo non è il caso allora non esistono cammini aumentanti e l'accoppiamento è di massima cardinalità. Ciò corrisponde al fatto che non esistono cammini aumentanti su $\bar{G}_x$, e quindi il flusso x ha valore massimo. Se invece viene trovata una destinazione esposta raggiungibile da un'origine esposta viene invocata la procedura *Cambia-Accoppiamento* che realizza l'operazione di composizione $M \oplus P$ per il cammino alternante aumentante P che le unisce; tale operazione è analoga alla *Aumenta-Flusso* dell'algoritmo *Cammini-Aumentanti*, ma più semplice.

Esercizio 2.37. Si fornisca una descrizione formale, in pseudo-codice, delle procedure *Cammino-Aumentante* e *Cambia-Accoppiamento*.

La complessità di *Accoppiamento-MaxCard* è $O(mn)$, in qualunque modo venga implementata la visita: infatti, la complessità della generica procedura *Cammini-Aumentanti* è $O(mnU)$, ma in questo caso $U = 1$. In effetti, è immediato verificare che la procedura termina dopo al più n iterazioni, ognuna delle quali richiede una visita del grafo e quindi costa $O(m)$.

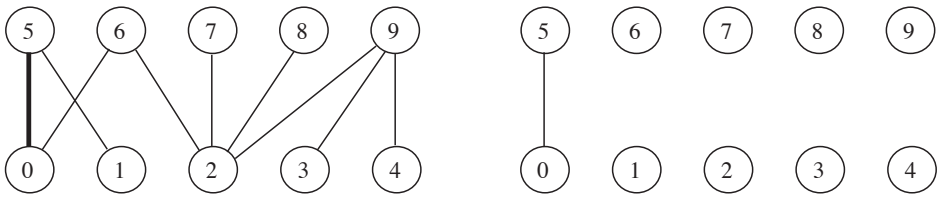
Esempio 2.28. Esecuzione di *Accoppiamento-MaxCard*



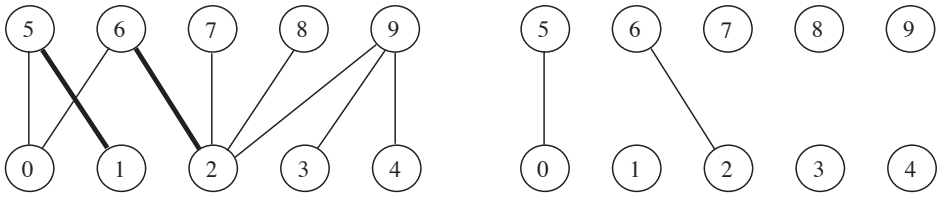
Si vuole risolvere il problema (MCM) per il grafo in figura mediante l'algoritmo dei cammini alternanti aumentanti che effettua una visita in ampiezza (ossia l'algoritmo di Edmons&Karp). Durante la visita si esaminerà la stella uscente del vertice selezionato in ordine crescente dell'altro estremo del lato; poiché si assume che le origini siano i nodi con gli indici inferiori, ciò implica un analogo ordinamento per l'ordine di visita delle origini esposte.

L'algoritmo parte dall'accoppiamento vuoto (flusso tutto nullo) e compie le cinque iterazioni mostrate in seguito; a sinistra è mostrato l'albero della visita (archi più spessi sul grafo originale, visto che l'orientamento degli archi in questo caso è ovvio), mentre a destra è mostrato l'accoppiamento ottenuto dopo l'applicazione dell'operazione di composizione col cammino scelto.

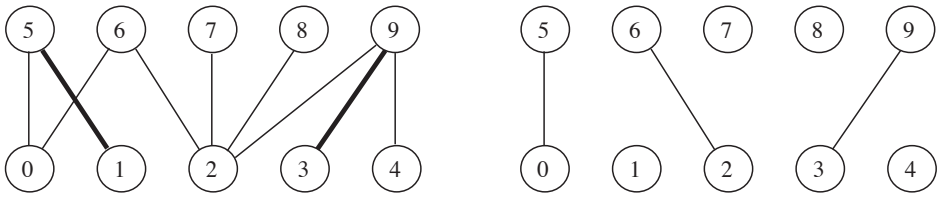
it. 1)



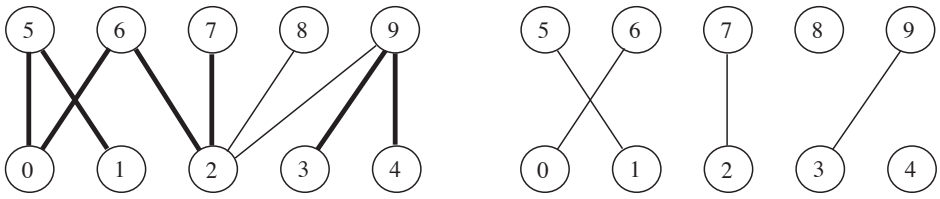
it. 2)



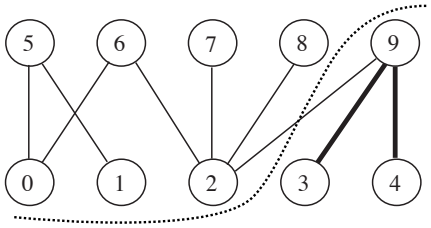
it. 3)



it. 4)



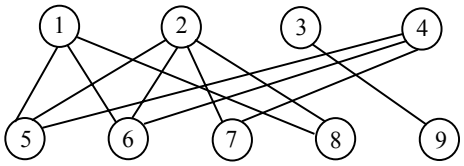
it. 5)



Alla quinta iterazione, l'albero della visita determina il taglio $(N_s, N_t) = (\{3, 4, 9\}, \{0, 1, 2, 5, 6, 7, 8\})$ evidenziato in figura. Nel grafo del problema di flusso, in cui sono aggiunte la sorgente s e la destinazione t, tale taglio ha una capacità pari a 4: infatti, gli archi diretti nel taglio (tutti di capacità pari ad 1, ovviamente) in tale

grafo sono $(s, 0)$, $(s, 1)$, $(s, 2)$, $(9, t)$. Pertanto ciascun flusso ammissibile può avere valore al massimo pari a 4, e pertanto l'assegnamento (flusso) determinato, di capacità (valore) 4, è ottimo.

Esercizio 2.38. Si applichi *Accoppiamento-MaxCard* al grafo in figura qui accanto, fornendo ad ogni iterazione l'accoppiamento, l'albero della visita ed il cammino aumentante.



2.6.2 Assegnamento di Costo Minimo

Analogamente al problema (MCM), il problema dell'*Assegnamento di Costo Minimo* (MCA, da *Min-Cost Assignment*) è equivalente al problema (MCF) sul grafo orientato $\bar{G}$ in cui le capacità degli archi sono unitarie, i costi degli archi sono quelli del problema di accoppiamento, ogni nodo in O produce un'unità di flusso ed ogni nodo in D consuma un'unità di flusso. La trasformazione è illustrata in Figura 2.24 (b) per l'istanza in (a) (per chiarezza di visualizzazione non sono indicate le capacità degli archi, tutte pari a 1).

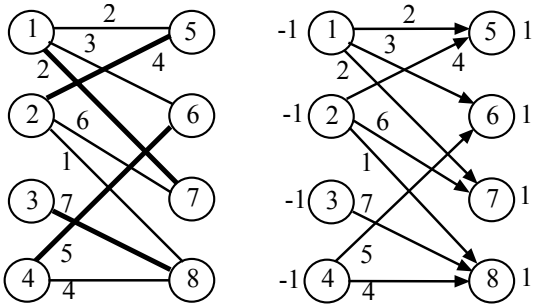


Figura 2.24: Trasformazione in un (MCF)

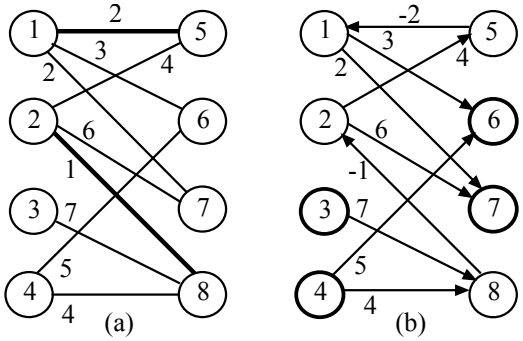
Di conseguenza è possibile specializzare gli algoritmi per il (MCF) al caso particolare del (MCA). Nell'algoritmo basato sui cammini minimi successivi, ad esempio, si determina ad ogni passo un cammino aumentante di costo minimo che connette un nodo con eccesso di flusso ad uno con difetto di flusso: è immediato verificare che, nel caso del (MCA), ciò corrisponde a determinare il *cammino alternante aumentante di costo minimo* rispetto all'accoppiamento corrente M. Per fare questo si può utilizzare il grafo ausiliario $G_M = (N, A_M)$ con

$$A_M = A_M^+ \cup A_M^- = \{ (i, j) : \{i, j\} \in E \setminus M \} \cup \{ (j, i) : \{i, j\} \in M \} ,$$

dove $(i, j) \in A_M^+$ ha costo c_{ij} mentre $(j, i) \in A_M^-$ ha costo $-c_{ij}$. È facile verificare che G_M è il grafo residuo $\bar{G}_x$ per lo pseudoflusso x corrispondente all'accoppiamento M. Un qualunque cammino orientato P_{st} da un nodo $s \in O_M$ ad un nodo $t \in D_M$ corrisponde ad un cammino alternante aumentante P, e viceversa; inoltre, per costruzione il costo di P_{st} in G_M è uguale al costo di P, definito come in (2.18). Inviare un'unità di flusso lungo un cammino aumentante corrisponde ad applicare l'operazione di composizione $M' = M \oplus P$ del paragrafo precedente; è facile verificare che risulta $C(M') = C(M) + C(P)$.

Esempio 2.29. Cammini alternanti aumentanti

Consideriamo il grafo G e l'accoppiamento M in figura (a) qui accanto, di costo $C(M) = 3$. Il grafo ausiliario G_M è descritto in figura (b). Al cammino orientato $P_{37} = \{(3, 8), (8, 2), (2, 7)\}$ nel grafo ausiliario in figura (b), avente costo 12, corrisponde nel grafo originario il cammino aumentante $P = \{\{3, 8\}, \{2, 8\}, \{2, 7\}\}$, avente anch'esso costo 12.



Quanto detto finora porta alla definizione del seguente algoritmo per la soluzione del problema.

```
procedure M = Assegnamento-MinCost( O , D , E , c ) {
 M = ∅;
 while( O_M ≠ ∅ ) do {
   ( p , d ) = Cammino-AA-Minimo( G_M );
   t = argmin { d[ i ] : i ∈ D_M };
   if( d[ t ] == ∞ ) then break;
   M = Cambia-Accoppiamento( M , p , t );
   }
 }
```

Procedura 2.11: Algoritmo *Assegnamento-MinCost*

L'algoritmo parte dall'accoppiamento iniziale vuoto, corrispondente al flusso iniziale $x = 0$. Si noti che non è necessario eseguire la procedura *Inizializza* del §2.5.3, anche in presenza di costi negativi, in quanto x è sicuramente minimale: il grafo residuo, essendo bipartito, non contiene nessun ciclo orientato. La procedura *Cammino-AA-Minimo* determina una albero dei cammini minimi con insieme di nodi radice O_M ed insieme di nodi destinazione O_M su G_M, col relativo vettore di etichette d. Si seleziona poi il nodo $t \in D_M$ con etichetta $d[\,t\,]$ minore): se $d[\,t\,] = \infty$ allora nessuna origine esposta è connessa con una destinazione esposta e l'algoritmo termina avendo determinato comunque un accoppiamento di massima cardinalità (ma non un assegnamento). Altrimenti si è determinato un cammino alternante aumentante P di costo minimo tra un nodo $s \in O_M$ ed un nodo $t \in D_M$: la procedura *Cambia-Accoppiamento* esegue l'operazione di composizione tra l'accoppiamento M corrente ed il cammino P. Quando l'algoritmo termina, se $|\,M\,| = |\,O\,| = |\,D\,|$ allora si è determinato un assegnamento di costo minimo, altrimenti non esistono accoppiamenti perfetti in G.

Esercizio 2.39. Dimostrare l'affermazione precedente.

Esempio 2.30. Due iterazioni di *Assegnamento-MinCost*

In Figura 2.25 sono mostrate in dettaglio due iterazioni dell'algoritmo. In (a) l'accoppiamento M corrente, in (b) il corrispondente grafo ausiliario G_M ed in (c) l'albero dei cammini minimi con insieme di radici $R = O_M = \{\,3\,,\,4\,\}$, con le relative etichette ottime ai nodi. Essendo $D_M = \{\,6\,,\,7\,\}$, si pone $t = 6$ poiché $d_6 = 5 < 7 = d_7$, selezionando così il cammino alternante aumentante $P = \{\,\{\,4\,,\,6\,\}\,\}$. In Figura 2.25(d) è mostrato il nuovo accoppiamento $M' = M \oplus P = \{\,\{\,1\,,\,5\,\}\,,\,\{\,2\,,\,8\,\}\,,\,\{\,4\,,\,6\,\}\,\}$, di cardinalità 3 e costo $C(M') = C(M) + C(P) = 3 + 5 = 8$, mentre in 2.25(e) ed (f) sono mostrati rispettivamente il corrispondente grafo ausiliario $G_{M'}$ e l'albero dei cammini minimi. Il cammino alternante aumentante di costo minimo è $P' = \{\,\{3,8\}\,,\,\{2,8\}\,,\,\{2,5\}\,,\,\{1,5\}\,,\,\{1,7\}\,\}$, con $C(P') = d(7) = 10$. L'assegnamento ottimo, $M'' = M' \oplus P' = \{\,\{1,5\}\,,\,\{2,8\}\,,\,\{4,6\}\,\} \setminus \{\,\{2,8\}\,,\,\{1,5\}\,\} \cup \{\,\{3,8\}\,,\,\{2,5\}\,,\,\{1,7\}\,\} = \{\,\{1,7\}\,,\,\{2,5\}\,,\,\{3,8\}\,,\,\{4,6\}\,\}$, di costo $C(M'') = C(M') + c(P') = 8 + 10 = 18$, è mostrato in Figura 2.24(a).

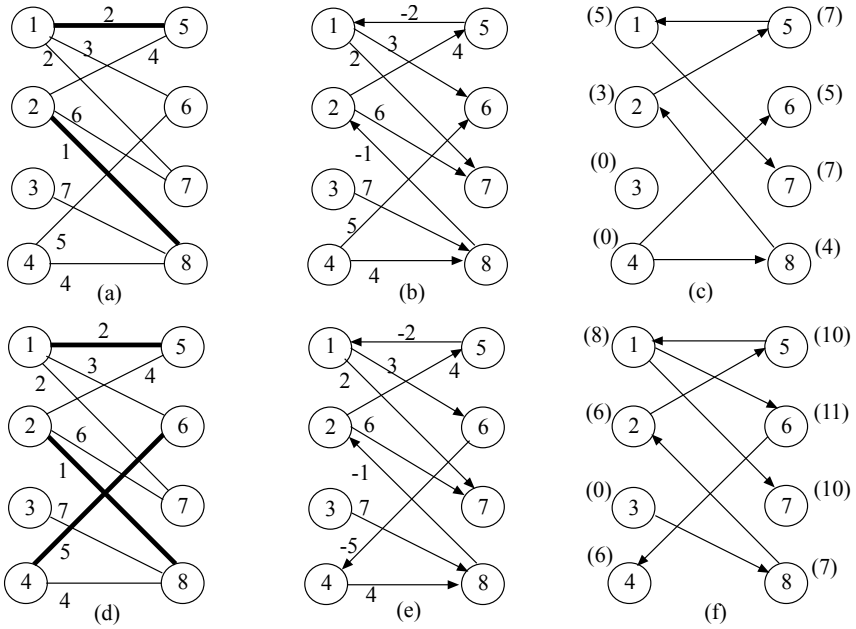
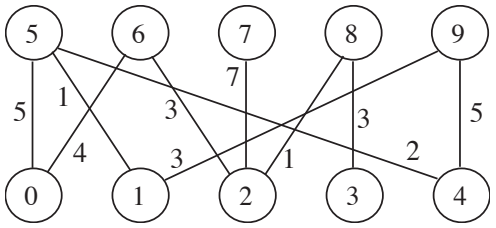


Figura 2.25: Due iterazioni di *Assegnamento-MinCost*

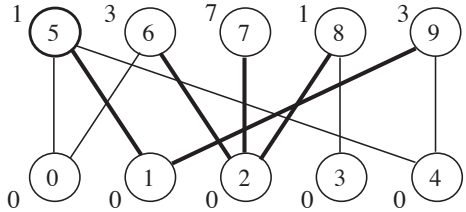
Esempio 2.31. Esecuzione di *Assegnamento-MinCost*

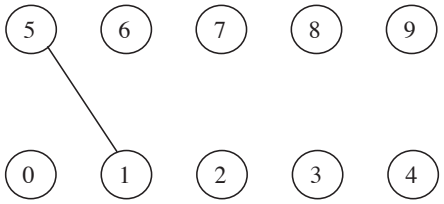
Si vuole risolvere il problema (MCA) per il grafo in figura mediante l'algoritmo dei cammini alternanti aumentanti minimi. Dato l'albero dei cammini minimi, per determinare lo specifico cammino da usare nell'operazione di composizione si sceglie la destinazione (le destinazioni sono i nodi della riga superiore) esposta con l'etichetta minima, ed a parità di etichetta quella di indice minimo.



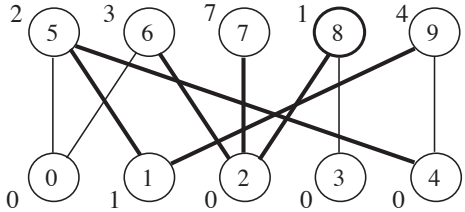
L'algoritmo parte dall'accoppiamento vuoto (flusso tutto nullo, dato che i costi degli archi sono tutti positivi) e compie le cinque iterazioni mostrate in seguito; l'albero dei cammini minimi (sul grafo originale, visto che l'orientamento degli archi in questo caso è ovvio) è mostrato a sinistra (archi più spessi), evidenziando la destinazione esposta selezionata, mentre a destra è mostrato l'accoppiamento ottenuto dopo l'applicazione dell'operazione di composizione col cammino scelto.

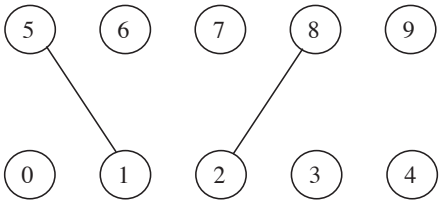
it. 1)



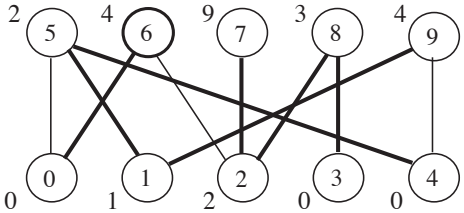


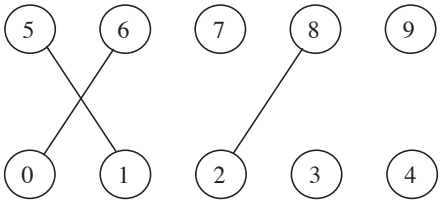
it. 2)



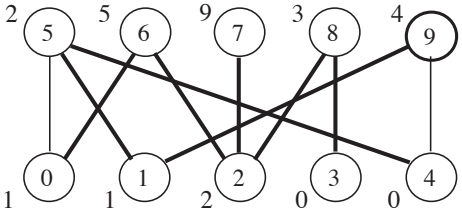


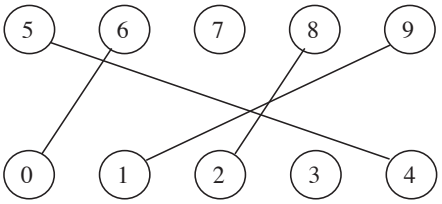
it. 3)



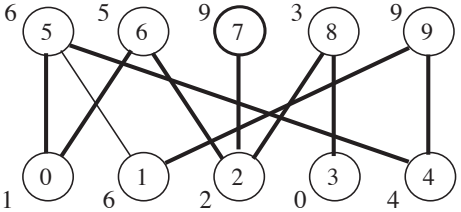


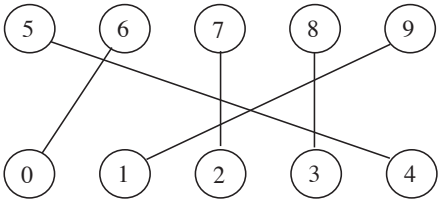
it. 4)





it. 5)





Alla quinta iterazione si è ottenuto un assegnamento (accoppiamento perfetto), che quindi è di costo minimo.

La correttezza dell'algoritmo deriva direttamente dalla correttezza della procedura *Cammini-Minimi-Successivi* per il problema del flusso di costo minimo; dall'analisi svolta per quella procedura risulta immediatamente che, se utilizziamo l'algoritmo *SPT.L.queue* per implementare *Cammino-AA-Minimo*, la complessità di *Assegnamento-MinCost* è $O(mn^2)$, essendo n il massimo numero di iterazioni. Con le tecniche descritte nel paragrafo 2.5.3 tale complessità può essere diminuita.

È anche possibile dimostrare che l'algoritmo *Assegnamento-MinCost* può essere usato anche per risolvere un problema più generale dell'assegnamento di costo minimo, ossia il problema dell'*accoppiamento di massima cardinalità e costo minimo*, nel quale si vuole determinare, tra tutti gli accoppiamenti di massima cardinalità (anche se non necessariamente perfetti), quello di costo minimo. Per fare questo è solamente necessario garantire che, ad ogni iterazione, il nodo $t \in D_M$ selezionato come destinazione sia uno di quelli di etichetta minima, ossia $d_t = \min\{ d_j : j \in D_M \}$, ovvero che il cammino alternante aumentante selezionato sia di costo minimo tra *tutti* i cammini che uniscono *un qualsiasi* nodo di O_M ad un nodo di D_M.

Esercizio 2.40. Si dimostri l'affermazione precedente. Si discuta inoltre se esiste una versione di questo risultato che si applica al problema di (MCF) (o meglio ad una sua generalizzazione).

2.6.3 Accoppiamento di Massima Cardinalità Bottleneck

Il problema dell'*Accoppiamento di Massima Cardinalità Bottleneck* (MCBM, da *Maximum Cardinality Bottleneck Matching*) non è facilmente formulabile come problema di (MCF), ma può essere facilmente risolto utilizzando come sottoprogrammi algoritmi visti nei paragrafi precedenti. Descriveremo le idee di base per il caso, più semplice, dell'assegnamento bottleneck, estendendole in seguito al caso più generale. Si supponga di conoscere il valore bottleneck ottimo

$$z = \min\{ V(M) : M \text{ è un assegnamento in } G \} \ ,$$

e si consideri il grafo parziale $G_v = (O \cup D , E_v)$, parametrico rispetto al valore reale v, dove

$$E_v = \{ \{ i , j \} \in E : c_{ij} \leq v \} \ .$$

Dalla definizione discende che se $v < z$, allora G_v non contiene accoppiamenti perfetti, altrimenti ($v \geq z$) G_v contiene almeno un assegnamento. Ciò suggerisce il seguente algoritmo:

1. si parte da un valore di v abbastanza piccolo per cui sicuramente valga $v \leq z$, ad esempio $v = \min\{ c_{ij} : \{ i , j \} \in E \}$;
2. si calcola un accoppiamento M di massima cardinalità in G_v: se $|M| = n$ ci si ferma, altrimenti si determina il più piccolo valore di v che permette di aggiungere archi ad E_v (cioè il minimo costo di un arco strettamente maggiore di v) e si itera.

Il processo di aggiunta di archi ad E_v viene iterato fino a che M non sia un accoppiamento perfetto, oppure sino a che $E_v = E$ e $|M| < n$: nel primo caso M è un assegnamento bottleneck, ossia ha valore $V(M) = v$ minimo tra tutti i possibili assegnamenti, mentre nel secondo caso il grafo G è privo di accoppiamenti perfetti. In questo caso, operando opportunamente, si può garantire che l'ultimo accoppiamento M prodotto sia bottleneck tra tutti gli accoppiamenti di massima cardinalità.

```
procedure ( M , v ) = Accoppiamento-Bottleneck( O , D , E , c ) {
( v , E_v ) = Inizializza( E , c ); M = ∅;
do { M = Accoppiamento-MaxCard( O , D , E_v , M );
     if( | M | ≥ n ) then break;
     v = min{ c_ij : { i , j } ∉ E_v }; E_v = E_v ∪ { { i , j } : c_ij = v };
     } while( E_v ⊊ E );
}
```

Procedura 2.12: Algoritmo *Accoppiamento-Bottleneck*

La procedura *Inizializza* determina un opportuno valore v ed il corrispondente insieme di archi E_v. Se si vuole trovare un assegnamento è possibile scegliere

$$v = \max\{ \min\{ c_{ij} : \{ i , j \} \in S(i)\} : i \in O \cup D \}$$

poiché per valori inferiori almeno un nodo risulterebbe isolato dagli altri, impedendo l'esistenza di un accoppiamento perfetto in G_v. Se invece si vuole risolvere il problema dell'accoppiamento bottleneck di massima cardinalità è possibile scegliere $v = \min\{ c_{ij} : \{ i , j \} \in E\}$, poiché per valori inferiori E_v è vuoto. La procedura *Accoppiamento-MaxCard* determina un accoppiamento di massima cardinalità

in G_v. Si noti che, durante il processo iterativo, è possibile utilizzare come accoppiamento di partenza l'accoppiamento di massima cardinalità M determinato all'iterazione precedente, che è sicuramente ancora un accoppiamento valido in quanto tutti gli archi che erano in E_v all'iterazione precedente ci sono anche in quella attuale (v è crescente). Poiché ad ogni iterazione si aggiunge ad E_v almeno un arco, non si effettueranno più di m iterazioni. Se si modifica la procedura *Accoppiamento-MaxCard* in modo da utilizzare come accoppiamento di partenza l'accoppiamento M in input, si determineranno al più n cammini aumentanti durante l'intera esecuzione dell'algoritmo; quindi che la complessità di tutte le chiamate ad *Accoppiamento-MaxCard* è $O(\,mn\,)$. Per determinare in modo efficiente il nuovo valore di v a ciascuna iterazione è sufficiente ordinare E all'inizio in modo tale che, globalmente, il costo di determinare il nuovo valore di v e i lati da aggiungere ad E_v sia $O(\,m\,)$. Siccome l'ordinamento costa $O(\,m \log n\,)$ e viene fatto una volta sola, la complessità di *Accoppiamento-Bottleneck* è $O(\,mn\,)$.

Esempio 2.32. Esecuzione di *Accoppiamento-Bottleneck*
Si vuole determinare un assegnamento bottleneck nel grafo G di Figura 2.26(a). Il valore di partenza è $v = 4$, corrispondente al minimo costo di lati uscenti dal nodo 4, poiché per valori inferiori il nodo 4 risulterebbe isolato. In Figura 2.26(b) vengono mostrati il grafo parziale G_4 e l'accoppiamento M con $|M| = 3$. Nella prima iterazione $v = 6$, ma l'aggiunta di $\{\,2\,,\,7\,\}$ non modifica l'accoppiamento. All'iterazione successiva, $v = 7$; in Figura 2.26(c) viene mostrato il grafo G_7 ed il nuovo accoppiamento $M = \{\{\,1\,,\,5\,\},\{\,2\,,\,8\,\},\{\,3\,,\,7\,\},\{\,4\,,\,6\,\}\}$ ottenuto dal precedente mediante il cammino aumentante $P = \{\{\,4\,,\,6\,\}\}$. M_7 è perfetto: si tratta quindi di un assegnamento bottleneck, con valore $V(M) = 7$.

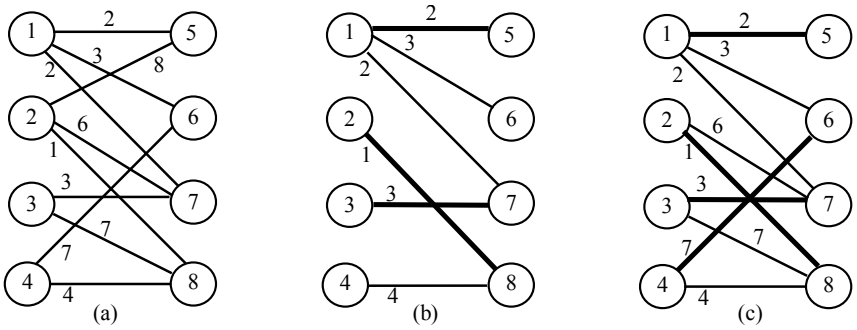


Figura 2.26: Alcune iterazioni di *Accoppiamento-Bottleneck*

Utilizzare la versione modificata di *Accoppiamento-MaxCard* che riparte dall'accoppiamento M in input non è soltanto un utile accorgimento che permette di velocizzare l'algoritmo, ma è necessario per la correttezza nel caso in cui non esistano accoppiamenti perfetti. Si consideri ad esempio il caso del grafo $G = (\,O \cup D\,,\,A\,)$ con $O = \{\,1\,,\,2\,\}$, $D = \{\,3\,,\,4\,\}$, $E = \{\{\,1\,,\,3\,\},\{\,2\,,\,3\,\}\}$, $c_{13} = 1$ e $c_{23} = 10$. Al primo passo della procedura $v = 1$ e quindi $E_v = M = \{\{\,1\,,\,3\,\}\}$. Al secondo passo $v = 10$ e quindi $E_v = E$. Se *Accoppiamento-MaxCard* partisse con l'accoppiamento iniziale $M = \emptyset$, potrebbe determinare come accoppiamento di massima cardinalità su G_v sia $M = \{\{\,1\,,\,3\,\}\}$ che $M' = \{\{\,2\,,\,3\,\}\}$, dato che entrambe sono accoppiamenti di cardinalità 1: chiaramente solo M è un accoppiamento bottleneck di massima cardinalità, per cui se venisse determinato M' l'algoritmo darebbe una risposta errata.

In generale, se non esistono assegnamenti ed il valore bottleneck è minore del massimo costo dei lati, l'algoritmo eseguirà una sequenza finale di iterazioni in cui cerca senza successo di costruire un accoppiamento di cardinalità maggiore di quello disponibile, finchè non esaurisce l'insieme dei lati e termina. Per la correttezza dell'algoritmo, è cruciale che durante queste iterazioni l'algoritmo non modifichi l'accoppiamento corrente costruendo un accoppiamento con la stessa cardinalità ma contenente lati a costo più alto: per garantire questo è sufficiente fare in modo che *Accoppiamento-MaxCard* riparta dal precedente accoppiamento M. Infatti, la procedura modificherà l'accoppiamento solo se può aumentarne la cardinalità: quindi, ad ogni iterazione di *Accoppiamento-Bottleneck* l'accoppiamento corrente M contiene solo lati il cui costo è minore od uguale del valore v' corrispondente all'ultima iterazione in cui la cardinalità di M è aumentata. Questo garantisce che, a terminazione, l'accoppiamento sia bottleneck anche nel caso in cui non sia perfetto.

Riferimenti Bibliografici

R.K. Ahuja, T.L. Magnanti, J.B. Orlin "Network flows. Theory, algorithms, and applications" Prentice Hall, 1993

M.S. Bazaraa, J.J. Jarvis, H.D. Sherali "Linear programming and network flows" Wiley, 1990

J. Lee "A First Course in Linear Optimization v4.06" 2022
https://github.com/jon77lee/JLee_LinearOptimizationBook/blob/master/JLee.4.06.zip

F. Schoen "Optimization Models" free e-book version, 2022
https://webgol.dinfo.unifi.it/OptimizationModels/contents.html

M. Pappalardo, M. Passacantando "Ricerca Operativa" Edizioni Plus, 2013

Capitolo 3

Programmazione Lineare

Questo capitolo è interamente dedicato agli algoritmi risolutivi, ed alla teoria che li supporta, per una classe particolarmente importante di problemi di ottimizzazione, ossia i problemi di *Programmazione Lineare* (*PL*) già introdotti a livello modellistico nel §1.2.1. Questi problemi sono caratterizzati dal fatto che tutte le relazioni tra le variabili, sia nei vincoli che nella funzione obiettivo, sono lineari; inoltre, le variabili non sono vincolate ad assumere insiemi discreti di valori (ad esempio solamente valori interi), ma possono assumere valori reali. Per quanto l'assunzione di linearità nei fenomeni rappresentati dal modello possa apparire piuttosto restrittiva, questa classe di problemi ha un forte interesse pratico. Infatti, in molte situazioni reali i componenti di un sistema reagiscono in modo almeno approssimativamente lineare alle decisioni prese, quindi molti problemi reali possono essere modellati con sufficiente precisione in termini di *PL*. Inoltre, per questa classe di problemi sono disponibili algoritmi risolutivi efficienti che consentono di risolvere istanze di dimensione elevata (fino a centinaia di migliaia o milioni di vincoli e variabili) anche su computer di potenza limitata. Infine, molti approcci per problemi complessi (si vedano i Capitoli 4 e 7) sono basati sulla risoluzione di un numero, spesso elevato, di sottoproblemi di *PL* che "approssimano" il problema.

I problemi di *PL* non sono la classe più grande di problemi per cui si conoscono algoritmi efficienti; in particolare, le classi dei problemi di *Programmazione Quadratica convessa*, *Programmazione Conica* (SOCP e SDP) ed *Ottimizzazione Nonlineare convessa* contengono tutte strettamente la *PL* (e si contengono l'un l'altra, nell'ordine dato) ed ammettono algoritmi efficienti sia in teoria che in pratica. Le definizioni di queste classi ed i corrispondenti algoritmi risolutivi sono però molto più complesse di quelle richieste per la *PL*, e per esse si rimanda alla letteratura citata ed a corsi più avanzati nell'area di Ricerca Operativa. È comunque sicuramente vero che la conoscenza della teoria della *PL* e dei relativi algoritmi permette di comprendere molto più facilmente quelle delle classi più complesse, e che quindi la *PL* è la classe di problemi dai quali è opportuno iniziare lo studio di algoritmi di ottimizzazione.

3.1 Problemi di Programmazione Lineare

Un problema di Programmazione Lineare è un problema di ottimizzazione (di massimo o di minimo) caratterizzato dalle seguenti proprietà:

1. un numero *finito* n di variabili, che possono assumere valori reali; in altri termini, il vettore $[\, x_i \,]_{i=1}^{n} = x \in \mathbb{R}^n$ delle variabili;
2. la funzione obiettivo $c(\, x \,) : \mathbb{R}^n \to \mathbb{R}$ è lineare, ovvero tale che $c(\, \alpha x + \beta y \,) = \alpha c(\, x \,) + \beta c(\, y \,)$ per ogni x, $y \in \mathbb{R}^n$ e per ogni α, $\beta \in \mathbb{R}$; ricordiamo che $c(\, x \,)$ è una funzione lineare se e solo se esiste $c \in \mathbb{R}^n$ tale che $c(\, x \,) = cx$.
3. l'insieme ammissibile è definito da un insieme finito di vincoli lineari del tipo $ax = b$ e/o $ax \leq b$ e/o $ax \geq b$, dove $a \in \mathbb{R}^n$ e $b \in \mathbb{R}$.

I vincoli di un problema di *PL* possono quindi essere sia di uguaglianza che di disuguaglianza, e questi ultimi possono essere sia di maggiore od uguale che di minore od uguale. Nell'analisi dei problemi di

PL conviene però adottare una qualche forma standard in cui tutti i vincoli abbiano lo stesso formato; nel seguito, utilizzeremo principalmente la forma standard

$$\max \{ cx \,:\, Ax \leq b \} \tag{3.1}$$

in cui A è una matrice reale $m \times n$, $b \in \mathbb{R}^m$ e $x \in \mathbb{R}^n$.

Esempio 3.1.
Per il problema della *Pintel*, descritto nell'Esempio 1.3, si ha $n = 2$, $m = 5$,

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 2 & 1 \\ -1 & 0 \\ 0 & -1 \end{bmatrix} \quad , \quad b = \begin{bmatrix} 400000 \\ 700000 \\ 900000 \\ 0 \\ 0 \end{bmatrix} \quad , \quad c = \begin{bmatrix} 500 \, , \, 200 \end{bmatrix} \ .$$

Si noti che, poiché le variabili possono assumere variabili reali, sarebbe stato possibile ad esempio scalare di un fattore 100000 i termini noti (ossia considerare come unità un lotto di 100000 processori) come in Figura 1.1; ciò verrà spesso fatto nel seguito per semplicità di notazione. Inoltre, data la forma che abbiamo scelto per il problema i vincoli di non negatività devono essere incorporati nella matrice A "cambiandone il verso", ossia rappresentando, ad esempio, "$x_1 \geq 0$" come "$-x_1 \leq 0$".

Assumere che un problema di *PL* sia dato nella forma (3.1) non comporta alcuna perdita di generalità, in quanto qualsiasi problema di *PL* può essere agevolmente ricondotto alla forma (3.1) introducendo oppure eliminando vincoli e/o variabili per mezzo delle seguenti equivalenze:

$$\begin{array}{lrcl}
i) & \max \sum_{j=1}^n c_j x_j & \equiv & -\min \sum_{j=1}^n (-c_j) x_j \\
ii) & \sum_{j=1}^n a_{ij} x_j = b_i & \equiv & \left\{ \begin{array}{ll} \sum_{j=1}^n a_{ij} x_j & \leq \ b_i \\ \sum_{j=1}^n a_{ij} x_j & \geq \ b_i \end{array} \right. \\
iii) & \sum_{j=1}^n a_{ij} x_j \geq b_i & \equiv & \sum_{j=1}^n (-a_{ij}) x_j \leq -b_i \\
iv) & \sum_{j=1}^n a_{ij} x_j \geq b_i & \equiv & \sum_{j=1}^n a_{ij} x_j - s_i = b_i, \quad s_i \geq 0 \\
v) & \sum_{j=1}^n a_{ij} x_j \leq b_i & \equiv & \sum_{j=1}^n a_{ij} x_j + s_i = b_i, \quad s_i \geq 0
\end{array} \tag{3.2}$$

Le variabili s_i in (3.2.*iv*) e (3.2.*v*) sono dette *variabili di scarto*, e sono variabili "nuove" (ausiliarie), introdotte appositamente nel modello, che appaiono solamente nei vincoli mostrati e non appaiono (ossia hanno coefficiente zero) in nessun altro vincolo e nella funzione obiettivo; le variabili x_i sono invece dette *variabili strutturali*, per evidenziare il loro scaturire dalla struttura del problema e non da manipolazioni algebriche effettuate sul modello. Infine, è sempre possibile trasformare un problema in cui una variabile x_i non è vincolata in segno in uno equivalente in cui al posto di x_i appaiono due variabili vincolate in segno, che possiamo indicare con x_i^+ e x_i^-, tramite la trasformazione algebrica

$$x_i = x_i^+ - x_i^- \quad , \quad x_i^+ \geq 0 \quad , \quad x_i^- \geq 0 \ . \tag{3.3}$$

Esempio 3.2. Trasformazioni equivalenti
Il problema della Fonderia dell'Esempio 1.4

$$\begin{array}{lrcrcrcrcl}
\min & 0.025x_1 & + & 0.030x_2 & + & 0.018x_3 & + & 10x_4 & & \\
 & 4x_1 & + & x_2 & + & 0.6x_3 & & & \geq & 3250 \\
 & 4x_1 & + & x_2 & + & 0.6x_3 & & & \leq & 5500 \\
 & 0.45x_1 & + & 0.5x_2 & + & 0.4x_3 & + & 100x_4 & \geq & 450 \\
 & x_1 & + & x_2 & + & x_3 & + & x_4 & = & 1000 \\
 & x_1 & , & x_2 & , & x_3 & , & x_4 & \geq & 0
\end{array}$$

può essere portato in forma (3.1) applicando la trasformazione (3.2.*i*) alla funzione obiettivo, la trasformazione (3.2.*ii*) al quarto vincolo, la trasformazione (3.2.*iii*) al primo ed al terzo vincolo, ed introducendo vincoli espliciti

per la non negatività delle variabili. Ciò porta alla formulazione

$$\begin{array}{rrrrrrrrrrl} -\max & -0.025x_1 & - & 0.030x_2 & - & 0.018x_3 & - & 10x_4 & & \\ & -4x_1 & - & x_2 & - & 0.6x_3 & & & \leq & -3250 \\ & 4x_1 & + & x_2 & + & 0.6x_3 & & & \leq & 5500 \\ & -0.45x_1 & - & 0.5x_2 & - & 0.4x_3 & - & 100x_4 & \leq & -450 \\ & x_1 & + & x_2 & + & x_3 & + & x_4 & \leq & 1000 \\ & -x_1 & - & x_2 & - & x_3 & - & x_4 & \leq & -1000 \\ & -x_1 & & & & & & & \leq & 0 \\ & & - & x_2 & & & & & \leq & 0 \\ & & & & - & x_3 & & & \leq & 0 \\ & & & & & & - & x_4 & \leq & 0. \end{array}$$

Si ottiene quindi

$$c = \begin{bmatrix} -0.025 & -0.030 & -0.018 & -10 \end{bmatrix}$$

$$A = \begin{bmatrix} -4 & -1 & -0.6 & 0 \\ 4 & 1 & 0.6 & 0 \\ -0.45 & -0.5 & -0.4 & -100 \\ 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}, \qquad b = \begin{bmatrix} -3250 \\ 5500 \\ -450 \\ 1000 \\ -1000 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Si noti che tramite le equivalenze (3.2) e (3.3) è anche possibile trasformare un problema avente forma (3.1) in un altro equivalente di forma diversa. Ad esempio, si potrebbe alternativamente utilizzare come forma standard la seguente:

$$\min\{\, cx \;:\; Ax = b\,,\; x \geq 0\,\}\ . \tag{3.4}$$

Esempio 3.3. Una diversa forma standard
Introducendo delle variabili di scarto ed esplicitando i vincoli di non negatività, il problema della Pintel può essere posto nella forma (3.4) con

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 & 1 \end{bmatrix} \quad , \quad b = \begin{bmatrix} 400000 \\ 700000 \\ 900000 \end{bmatrix} \quad , \quad c = \begin{bmatrix} -500\,, -200 \end{bmatrix}\ .$$

In questa nuova formulazione le ultime tre componenti del vettore $x \in \mathbb{R}^5$ sono le variabili di scarto.

Se il numero di variabili è pari a due oppure tre, un problema di *PL* può essere descritto (e risolto) mediante una sua rappresentazione geometrica nel piano o nello spazio. In Figura 3.1 ciò è mostrato per il problema della *Pintel*: avendo indicato con x_1 il numero di Pintium e con x_2 il numero di Coloron prodotti, l'area evidenziata col tratteggio rappresenta *l'insieme ammissibile*, o regione ammissibile, del problema, cioè l'insieme di tutti i punti che soddisfano i vincoli. Si tratta dell'intersezione di un certo numero di semipiani, uno per ogni vincolo: un insieme di questo tipo viene detto *poliedro* (*politopo* nel caso particolare in cui sia limitato, come nell'esempio che stiamo considerando). Nel seguito vedremo che tale insieme è convesso; infatti, si usa spesso il termine *poliedro convesso* per caratterizzare l'insieme ammissibile di un problema di *PL*. Nella figura sono evidenziati i vincoli per mezzo delle rette luogo dei punti che li soddisfano come uguaglianze: tali rette costituiscono la frontiera dei semipiani che essi definiscono. I vincoli di non negatività sono individuati dagli assi cartesiani. Le rette corrispondenti ai vincoli individuano nel poliedro delle *facce* e dei *vertici*: sono vertici ad esempio i punti $[\,4\,,\,1\,]$ e $[\,1\,,\,7\,]$, ed è una faccia il segmento che li unisce. I vertici e le

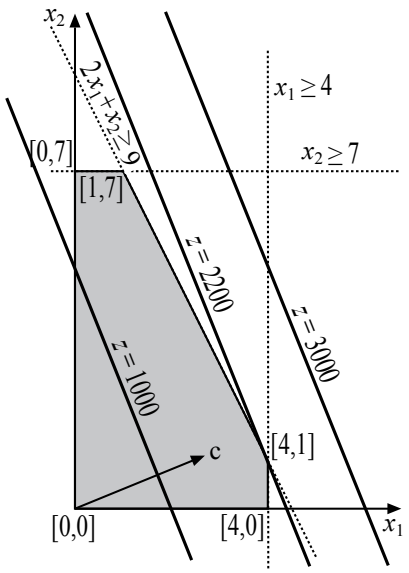


Figura 3.1: Il problema della Pintel

facce giocano un ruolo particolarmente importante nella *PL*: dimostreremo infatti che, sotto opportune ipotesi, se l'insieme delle soluzioni ottime di un problema di *PL* è non vuoto, allora almeno una soluzione ottima si trova in corrispondenza ad un vertice; inoltre, se un punto interno ad una faccia è soluzione ottima del problema, allora tutti i punti della faccia sono soluzioni ottime.

La verità di queste proprietà per il caso particolare in esame può essere facilmente compresa esaminando la figura. A questo scopo consideriamo la retta $500x_1 + 200x_2 = z$: essa definisce l'insieme delle soluzioni (eventualmente non ammissibili) che hanno valore della funzione obiettivo uguale a z. In figura sono indicate tre di tali rette, corrispondenti ai valori 1000, 2200 e 3000: al crescere di z le rette vengono traslate muovendosi nella direzione definita dal vettore $c = [\,500\,,\,200\,]$, *gradiente* della funzione obiettivo. Chiaramente, per ogni valore prefissato di z è possibile realizzare un profitto pari a quel valore se e solo se la corrispondente retta ha intersezione non vuota con la regione ammissibile: nel nostro esempio, ciò accade per $z = 1000$ e $z = 2200$, ma non per $z = 3000$. Pertanto, per trovare una soluzione ottima del nostro problema, basta traslare la retta nella direzione del gradiente fintanto che l'intersezione con la regione ammissibile si mantiene non vuota. Nel nostro caso, il massimo valore attribuibile a z è 2200: per tale valore, l'intersezione tra la retta e l'insieme ammissibile si riduce ad un solo punto, il vertice $[\,4\,,\,1\,]$, che è pertanto l'unica soluzione ottima del problema.

Nel nostro esempio la regione ammissibile del problema è limitata; in generale, però, la regione ammissibile di un problema di *PL* può essere non limitata lungo alcune direzioni. In questi casi, a seconda della direzione del gradiente della funzione obiettivo, possono esistere direzioni lungo le quali è possibile spostarsi mantenendo l'ammissibilità e facendo crescere il valore della funzione obiettivo senza mai raggiungere un valore massimo. Ad esempio, se nel problema della Pintel non ci fossero il secondo ed il terzo vincolo, potremmo fare crescere all'infinito il valore di z senza mai trovarne uno per cui la retta $500x_1 + 200x_2 = z$ abbia intersezione vuota con la regione ammissibile: questo è un caso di problema illimitato. Un caso in un certo senso opposto è quello in cui alcuni dei vincoli sono tra loro incompatibili, per cui

Esercizio 3.1. Costruire ulteriori esempi di *PL* nel piano. In particolare, fornire problemi per cui risulti rispettivamente: regione ammissibile vuota, problema illimitato, almeno due soluzioni ottime.

3.1.1 Geometria della Programmazione Lineare

In questo paragrafo introdurremo in maniera formale i concetti geometrici che permettono di caratterizzare l'insieme delle soluzioni ammissibili di un problema di *PL*. In un primo momento ci serviremo, come nella trattazione precedente, di esempi geometrici in due o tre dimensioni per introdurre alcuni concetti in modo solamente intuitivo, riservandoci di darne definizioni e dimostrazioni formali nel seguito.

3.1.1.1 Poliedri e coni

Una funzione lineare (affine) $f(x) = ax\ (+b)$ è (a meno del termine costante b) il *prodotto scalare* $ax = \sum_{i=1}^{n} a_i x_i$ tra i due vettori a (fisso) ed x (variabile) in $\mathbb{R}^n$. È quindi utile ricordare l'interpretazione geometrica dell'operazione di prodotto scalare: posto che la *norma* (Euclidea, o L_2) di un vettore $x \in \mathbb{R}^n$ è data da

$$\| x \| = \sqrt{xx} = \sqrt{\textstyle\sum_{i=1}^{n} x_i^2}$$

si ha

$$ax = \| a \| \| x \| \cos(\theta)$$

dove θ è l'angolo formato dai due vettori. Ciò significa che il prodotto scalare:

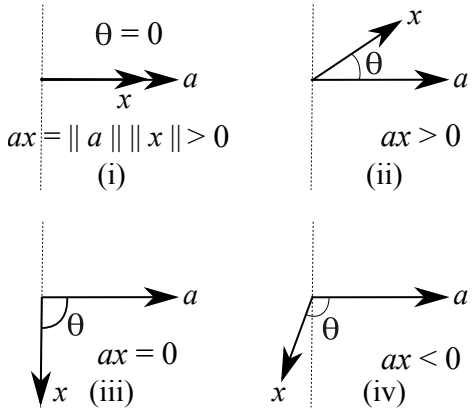


Figura 3.2: Geometria del prodotto scalare

- è positivo se i due vettori "puntano nello stesso semispazio", come in Figura 3.2(ii), ed in particolare è pari al prodotto delle norme se sono *collineari*, ossia $\theta = 0$, come in Figura 3.2(i);
- è nullo se i vettori sono *ortogonali*, come in Figura 3.2(iii);

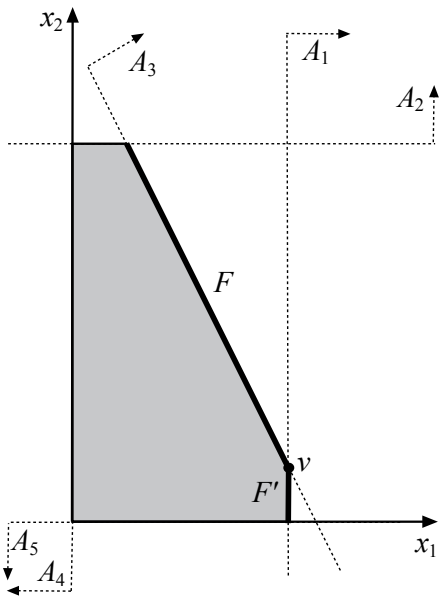
- è negativo se i due vettori "puntano in semispazi opposti", come in Figura 3.2(iv).

Possiamo adesso caratterizzare geometricamente i vincoli lineari, ossia ottenuti attraverso funzioni lineari. Nello spazio vettoriale $\mathbb{R}^n$, un vincolo lineare del tipo $A_i x = b_i$ individua l'*iperpiano affine* $P_i = \{ x : A_i x = b_i \}$, che costituisce la *frontiera* del *semispazio affine* $S_i = \{ x : A_i x \leq b_i \}$ individuato dal vincolo $A_i x \leq b_i$. Nel seguito, con una piccola imprecisione di linguaggio useremo semplicemente i termini *iperpiano* e *semispazio* sia per indicare un iperpiano / semispazio propriamente detto ($b_i = 0$) oppure una sua traslazione, cioè un iperpiano / semispazio affine ($b_i \neq 0$). Un insieme P è un *poliedro* se è esprimibile come intersezione di un numero finito m di semispazi, cioè se esistono una matrice $A \in \mathbb{R}^{m \times n}$ ed un vettore $b \in \mathbb{R}^m$ per cui

$$P \ = \ \{ \ x \ : \ Ax \leq b \ \} \ .$$

Esempio 3.4. Vincoli e facce

Nella figura qui accanto è mostrato ancora una volta il poliedro corrispondente all'insieme ammissibile del problema della Pintel. Ciascuna linea tratteggiata indica un iperpiano (retta) corrispondente ad uno dei vincoli $A_i x \leq b_i$, $i = 1, \ldots, 5$ del problema. Nella figura, a ciascun iperpiano è associato il vettore A_i (la corrispondente riga della matrice dei coefficienti), che è perpendicolare all'iperpiano e "punta" verso il semispazio (semipiano) in cui il vincolo è violato. Infatti, l'iperpiano è il luogo dei punti x percui $A_i x = b_i$, e la funzione lineare $A_i x$ cresce "andando in direzione A_i". Infatti si consideri un qualsiasi vettore $d \in \mathbb{R}^n$ tale che $A_i d > 0$, ossia che "punta nello stesso semispazio di A_i. È facile vedere che compiere un qualsiasi passo positivo $\alpha > 0$ lungo la direzione d a partire da un punto x dell'iperpiano porta ad un punto che non soddisfa il vincolo: $A_i(x + \alpha d) = A_i x + \alpha A_i d > A_i x = b_i$. Pertanto, i punti che soddisfano il vincolo sono quelli "nella direzione opposta di quella puntata da A_i".



Si noti come ciascun vincolo del problema sia in questo modo associato ad uno dei "lati" (facce) del poliedro. Le facce possono essere di dimensione diversa a seconda "di quanti iperpiani le caratterizzano". In Figura sono evidenziate le facce F ed F' del poliedro, corrispondenti rispettivamente ai vincoli $i = 3$ e $i = 1$; queste sono *facce massimali* o *faccette* in quanto di dimensione $n - 1 = 2 - 1 = 1$, e quindi "la più grande possibile" per un poliedro in $\mathbb{R}^n$ con $n = 2$. È inoltre evidenziato il *vertice* (ottimo) v corrispondente alla loro intersezione, e quindi alla coppia di vincoli $\{ 1 , 3 \}$; questa è una *faccia minimale* del poliedro in quanto ha dimensione 0.

Una delle principali proprietà dei poliedri è la loro *convessità*, dove ricordiamo che un insieme C è detto convesso se, comunque presi due punti x ed y appartenenti a C, il segmento avente x ed y per estremi è contenuto in C, cioè $conv(x , y) = \{ z = \alpha x + (1 - \alpha) y \ : \ \forall \alpha \in [0 , 1] \}$, è formato solamente da punti di C: $conv(x , y) \subseteq C$. Un semispazio è un insieme convesso, ed essendo l'intersezione di insiemi convessi a sua volta un insieme convesso, anche un poliedro è un insieme convesso.

Esercizio 3.2. Si dimostrino le due affermazioni precedenti.

Una definizione alternativa utilizza l'*inviluppo convesso* di insieme finito $X = \{ x^1 , \ldots , x^s \} \subset \mathbb{R}^n$

$$conv(X) \ = \ \left\{ \ x = \textstyle\sum_{i=1}^{s} \lambda_i x^i \ : \ \sum_{i=1}^{s} \lambda_i = 1 \ , \ \lambda_i \geq 0 \quad i = 1, \ldots, s \ \right\} \ ;$$

un insieme è convesso se contiene l'inviluppo convesso di ciascun sottoinsieme finito dei suoi punti.

Esempio 3.5. Un poliedro in $\mathbb{R}^3$

Si consideri il poliedro $P \subset \mathbb{R}^3$ definito dal seguente insieme di vincoli:

$$\begin{array}{rcrcll} -x_1 & & & \leq & 0 & (1) \\ & x_2 & & \leq & 1 & (2) \\ & & -x_3 & \leq & 0 & (3) \\ & -x_2 & & \leq & 0 & (4) \\ & & x_3 & \leq & 1 & (5) \\ & x_2 & + \; x_3 & \leq & 2 & (6) \end{array} \quad , \quad A = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} \quad , \quad b = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 2 \end{bmatrix} .$$

Il poliedro è il "parallelepipedo illimitato" rappresentato in Figura 3.3. In figura sono indicati i sei iperpiani che delimitano la frontiera dei sei semispazi corrispondenti ai sei vincoli: ad esempio, il semispazio $x_1 = 0$ corrisponde al piano verticale x_2x_3, che include il "lato sinistro" del parallelepipedo.

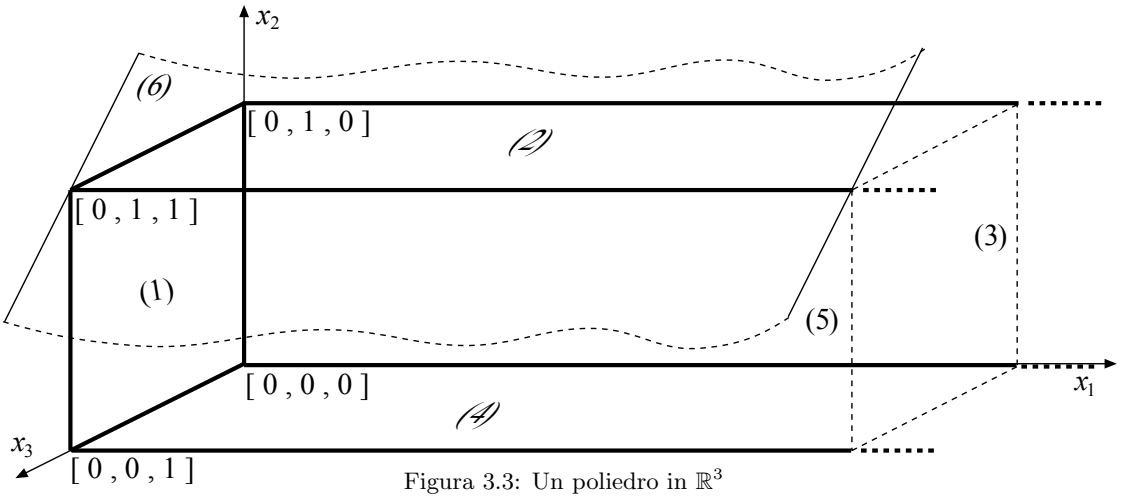


Figura 3.3: Un poliedro in $\mathbb{R}^3$

Anche in questo caso, i vincoli hanno una certa corrispondenza con le "facce" del poliedro. Ad esempio, il vincolo (1) individua il "lato sinistro" del parallelepipedo; questo può essere facilmente verificato considerando che qualora si rimuovesse il vincolo (1) si otterrebbe un "parallelepipedo infinito" avente come facce i quattro "rettangoli infiniti" corrispondenti al "lato superiore", al "lato posteriore", al "lato inferiore" ed al "lato anteriore" del parallelepipedo, "prolungati all'infinito a sinistra". In altri termini, la rimozione del vincolo fa scomparire la faccia corrispondente dal poliedro. Considerazioni analoghe possono essere fatte per i vincoli (2) ("lato superiore"), (3) ("lato posteriore"), (4) ("lato inferiore") e (5) ("lato anteriore"). Un caso diverso è quello del vincolo (6): la sua rimozione non cambia il poliedro, ossia il vincolo è *ridondante*. In effetti il vincolo non è associato ad alcuna faccia, nel senso intuitivo usato fin qui, del poliedro; vedremo in seguito che è comunque associato ad una faccia di dimensione minore (in particolare, allo spigolo "di fronte in alto").

Oltre alle facce "grandi" individuate dai vincoli, che in seguito definiremo come faccette o facce massimali, elementi geometricamente caratterizzanti un poliedro sono le sue facce "piccole" (minimali), ossia i suoi *vertici*. La definizione formale che corrisponde al concetto intuitivo è quella di *punto estremo*

$$x = \alpha x' + (1-\alpha)x'' \quad , \quad \alpha \in [\,0\,,\,1\,] \quad , \quad x' \in P \quad , \quad x'' \in P \qquad \Longrightarrow \qquad x' = x'' \ ,$$

ossia x è punto estremo se non può essere espresso come combinazione convessa (con coefficienti entrambi diversi da zero) di due punti diversi di P. Ad esempio, è facile verificare che il poliedro P di Figura 3.3 ha quattro vertici: $\left[\ 0\,,\,0\,,\,0\ \right]$, $\left[\ 0\,,\,1\,,\,0\ \right]$, $\left[\ 0\,,\,0\,,\,1\ \right]$ e $\left[\ 0\,,\,1\,,\,1\ \right]$. È immediato però osservare dall'esempio che i vertici non consentono di descrivere completamente il poliedro; i quattro "spigoli illimitati" di P hanno sì origine nei vertici, ma non terminano in vertici (si può pensare che lo facciano in "vertici all'infinito"). Per rappresentare formalmente questa proprietà si introduce il concetto di *direzione* (di recessione) di un poliedro, dove una direzione di un poliedro P è un qualsiasi vettore $d \in \mathbb{R}^n$ tale che

$$\forall\, x \in P \ \ \forall \lambda \geq 0 \qquad (x + \lambda d) \in P \ .$$

In particolare, il poliedro di Figura 3.3 ha una sola direzione, $\left[\ 1\,,\,0\,,\,0\ \right]$, che "identifica" i quattro spigoli illimitati.

Al fine di caratterizzare matematicamente le direzioni studiamo adesso la "forma particolarmente semplice" di poliedro

$$C = \{ x : Ax \leq 0 \} \tag{3.5}$$

in cui il lato destro dei vincoli è tutto nullo. Ciò implica immediatamente

$$x \in C \quad , \quad \alpha \geq 0 \qquad \Longrightarrow \qquad \alpha x \in C \ ; \tag{3.6}$$

un insieme con questa proprietà è detto *cono*, e dato che C è anche un insieme convesso, si parlerà di un *cono convesso*. Si noti che questo non è implicito nella definizione di cono, come mostrato in in Figura 3.4: A non è convesso, mentre B lo è. Poiché C è un cono (convesso) ed anche un poliedro, viene detto *cono poliedrico*.

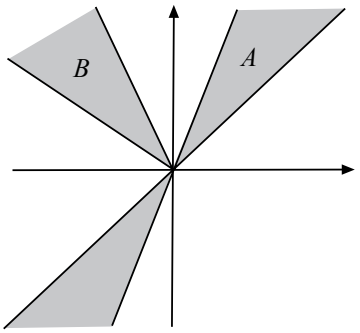


Figura 3.4: Coni convessi e non

È facile verificare che una definizione alternativa a (3.6), per il caso convesso, è

$$x, y \in C \quad , \quad \lambda, \mu \in \mathbb{R}_+ \qquad \Longrightarrow \qquad \lambda x + \mu y \in C \ . \tag{3.7}$$

La relazione (3.7) utilizza il concetto di *inviluppo conico* di due vettori; analogamente al caso dell'inviluppo convesso, questo può essere esteso ad un insieme finito $V = \{ v^1 , \ldots , v^t \} \subset \mathbb{R}^n$ come

$$cono(V) = \left\{ v = \sum_{i=1}^{t} \nu_i v^i \ : \ \nu_i \geq 0 \quad i = 1, \ldots, t \right\} \ . \tag{3.8}$$

Possiamo adesso caratterizzare l'insieme $rec(P)$ di tutte le direzioni (di recessione) di P.

Teorema 3.1. $rec(P) = \{ d \in \mathbb{R}^n : Ad \leq 0 \}$.

Dimostrazione È immediato verificare, direttamente dalla definizione, che se $Ad \leq 0$ allora d è una direzione di recessione. Viceversa, assumiamo che $d \in rec(P)$, ossia

$$A(x + \lambda d) \leq b \quad \Longrightarrow Ad \leq (b - Ax) / \lambda \qquad \forall \lambda > 0 \ ;$$

poiché $x \in P \Longrightarrow b - Ax \geq 0$ e la relazione vale per ogni $\lambda > 0$, per $\lambda \to \infty$ si ha $Ad \leq 0$. ◇

In altri termini, azzerando il lato destro dei vincoli si "cattura il comportamento all'infinito" di un poliedro. Il Teorema 3.1 giustifica il nome di *cono di recessione* dato a $rec(P)$, anche se la proprietà era già ovvia dalla definizione. Si noti che $rec(P)$ non è mai vuoto: chiaramente, 0 vi appartiene sempre. D'altro canto, 0 non è una direzione; quindi, si ha che o $rec(P,) = \{ 0 \}$, oppure esiste un $0 \neq d \in rec(P)$. È possibile verificare che questo caratterizza univocamente la *compattezza* di P: esiste una sfera di raggio finito che contiene interamente P se e solo se $rec(P) = \{ 0 \}$. Infatti, se $0 \neq d \in rec(P)$ allora P non è compatto: per ogni M grande a piacere esiste $x \in P$ tale che $\| x \| \geq M$. L'implicazione opposta è un ovvio corollario di risultati che saranno mostrati nel seguito. Un caso particolare, come vedremo, è quello in cui per una stessa d risulti

$$d \in rec(P) \quad , \quad -d \in rec(P) \ ;$$

un vettore siffatto si definisce *direzione di linealità* di P. Ad esempio, rimuovendo il vincolo (1) dal poliedro di Figura 3.3 si ottiene che $\left[\ 1 \, , \, 0 \, , \, 0 \ \right]$ è una direzione di linealità per il poliedro risultante. È un immediato corollario del Teorema 3.1 che d è una direzione di linealità se e solo se $Ad = 0$; in altri termini,

$$lin(P) = \{ d \in \mathbb{R}^n \ : \ Ad = 0 \}$$

è il *cono di linealità* di P. È anche immediato verificare che *se P ha direzioni di linealità allora non può avere vertici*; anche questo è confermato dall'esempio.

Esercizio 3.3. Dimostrare le affermazioni precedenti.

Le direzioni di P sono importanti nella nostra trattazione in quanto strettamente collegate al caso in cui i problemi di PL che hanno P come regione ammissibile siano superiormente illimitati. Infatti, se P non è vuoto e $d \in rec(P)$, allora tutti punti della forma $x(\lambda) = x + \lambda d$ per $\lambda \geq 0$ sono contenuti in P. Pertanto, se $cd > 0$ allora $cx(\lambda) = cx + \lambda cd$ è una funzione crescente di λ che non ammette estremo superiore finito. Quindi, determinare $d \in rec(P)$ tale che $cd > 0$ e verificare che $P \neq \emptyset$, ossia

determinare un qualsiasi $x \in P$, significa nei fatti aver risolto il problema di PL avendo ottenuto un *certificato* (x e d stessi) del fatto che il problema è superiormente illimitato.
Determinare se esiste oppure no una siffatta direzione sarebbe "facile" se il cono $rec(\,P\,)$, invece che nella forma (3.5), fosse dato nella forma (3.8), ossia se si conoscesse un insieme finito di vettori V tale che $rec(P) = cono(\,V\,)$.

Lemma 3.1. Esiste $d \in C = cono(\,V\,)$ tale che $cd > 0$ se e solo se esiste $v_i \in V$ tale che $cv_i > 0$.

Dimostrazione Una delle implicazioni è ovvia. Per dimostrare l'altra implicazione, si osservi che se $cv_i \leq 0$ per ogni $i = 1, \ldots, t$, allora $cd = c(\,\sum_{i=1}^{t} \nu_i v^i\,) = \sum_{i=1}^{t} \nu_i (cv^i) \leq 0$ per ogni $d \in cono(\,V\,)$ (dato che $\nu_i \geq 0$). ◇

In altri termini, sarebbe potenzialmente interessante se fosse possibile rappresentare un qualsiasi cono poliedrico, oltre che nella forma "per facce" (3.5), anche nella forma "per direzioni" (3.8). Dimostriamo che questo è in effetti possibile, in entrambe le direzioni.

Teorema 3.2. Dato $C = cono(\,V\,)$, esiste una matrice A per cui $C = \{\, x \,:\, Ax \leq 0 \,\}$.

Dimostrazione È chiaro che $cono(\,V\,)$ definito da (3.8) è un cono poliedrico nelle spazio esteso delle $[\,x\,,\,\nu\,] \in \mathbb{R}^{n+t}$: infatti i lati destri di tutti i vincoli sono nulli (alcuni dei vincoli sono di uguaglianza ma questo non è un problema, si veda (3.2.ii)). Occorre quindi dimostrare che la proiezione di un cono poliedrico su un sottospazio è ancora un cono poliedrico.
Questo risultato è vero in generale per i poliedri, e di dimostra mediante un procedimento algoritmico e costruttivo noto come *eliminazione di Fourier-Motzkin*. Il processo è iterativo e considera la proiezione di una variabile per volta; si consideri quindi un generico poliedro $\{\, x \,:\, Ax \leq b \,\} \subseteq \mathbb{R}^n$ e se ne voglia calcolare la proiezione sul sottospazio $\mathbb{R}^{n-1}$ delle variabili x_2, x_3, …, x_n. Per fare ciò si esamina il generico vincolo i-esimo, $i = 1, \ldots, m$,

$$\textstyle\sum_{j=1}^{n} a_{ij} x_j \leq b_i$$

e lo classifica in uno dei tre sottoinsiemi I^+, I^- ed I^0 a seconda che risulti $a_{i1} > 0$, $a_{i1} < 0$, ed $a_{i1} = 0$, rispettivamente. I vincoli in I^0 di fatto "non contengono la variabile x_1"; gli altri si possono riscrivere come

$$\begin{array}{rcl} i \in I^+ & \Longrightarrow & x_1 \leq \left(\, b_i - \sum_{j=2}^{n} a_{ij} x_j \,\right) / \, a_{i1} \\ i \in I^- & \Longrightarrow & x_1 \geq \left(\, b_i - \sum_{j=2}^{n} a_{ij} x_j \,\right) / \, a_{i1} \end{array}$$

A questo punto è possibile eliminare la variabile x_1 eliminando tutti i vincoli $i \in I^+ \cup I^-$ (e lasciando inalterati quelli in I^0 che non la contengono) e rimpiazzandoli con i vincoli

$$\left(\, b_h - \textstyle\sum_{j=2}^{n} a_{hj} x_j \,\right) / \, a_{h1} \leq \left(\, b_i - \sum_{j=2}^{n} a_{ij} x_j \,\right) / \, a_{i1} \tag{3.9}$$

per tutte le possibili coppie $(\,i\,,\,h\,) \in I^+ \times I^-$. Ripetendo tale procedimento un numero opportuno di volte si può calcolare la proiezione di un poliedro su un qualsiasi sottospazio delle sue variabili. Si noti che se il poliedro è un cono, ossia $b = 0$, allora $b_i = b_h = 0$ in (3.9) ed anche i nuovi vincoli introdotti hanno lato destro nullo, ossia la proiezione di un cono poliedrico è ancora un cono poliedrico. ◇

Un cono definito tramite la (3.8) si dice *finitamente generato*. Il Teorema 3.2 dice quindi che qualsiasi cono finitamente generato è un cono poliedrico.

Esempio 3.6. Trasformazione da cono finitamente generato a poliedrico
Si vuole esprimere cono finitamente generato

$$\mathcal{C} = cono\left(\left\{ \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} \right\} \right)$$

come un cono poliedrico. Seguendo la dimostrazione del Teorema 3.2, la definizione (3.8) da

$$\mathcal{C} = \left\{ \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \alpha_1 \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix} + \alpha_2 \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + \alpha_3 \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} + \alpha_4 \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} \;,\; [\,\alpha_1\,,\,\ldots\,,\,\alpha_4\,] \geq 0 \right\}$$

In altri termini, nello spazio esteso $[\,x\,,\,\alpha\,]$ il cono è definito dal seguente insieme di equazioni

$$\{\, x_1 = -\alpha_1 - \alpha_3 + \alpha_4 \;,\; x_2 = \alpha_2 - \alpha_3 + \alpha_4 \;,\; x_3 = \alpha_3 - \alpha_4$$

insieme ai vincoli di non-negatività sui moltiplicatori α_i; è quindi sufficiente eliminare questi ultimi, proiettando l'insieme sullo spazio delle sole x. Sarebbe possibile applicare pedissequamente l'eliminazione di Fourier-Motzkin

come vista nel Teorema, ma per velocizzare l'argomento utilizziamo prima degli argomenti ad-hoc. È innanzitutto evidente come α_3 ed α_4 appaiano sempre con lo stesso coefficiente cambiato di segno; in altri termini, le due variabili vincolate in segno sono equivalenti ad una sola variabile non vincolata in segno (si veda (3.3)). Il terzo vincolo permette di definire direttamente questa variabile non vincolata in segno come x_3: pertanto, il sistema diviene semplicemente

$$\{x_1 = \alpha_1 - x_3 \ , \ x_2 = \alpha_2 - x_3 \ , \ \alpha_1 \geq 0 \ , \ \alpha_2 \geq 0$$

che è facilmente riscrivibile come

$$\mathcal{C} = \left\{ \begin{array}{rcrcl} x_1 & & + & x_3 & \leq 0 \\ & - & x_2 & - & x_3 \leq 0 \end{array} \right.$$

(α_1 ed α_2 sono "variabili di scarto" utilizzate per esprimere vincoli di disuguaglianza come uguaglianza, si veda (3.2.iv)), ossia un cono poliedrico. Si noti che $\mathcal{C}$ contiene una direzione di linealità (gli ultimi due generatori sono opposti), il che corrisponde al fatto che—come cono poliedrico—è definito da "poche" facce (due, in $\mathbb{R}^3$) per cui la corrispondente matrice A non ha rango di colonna pieno; si veda il Teorema 3.5.

Un'osservazione rilevante è che, sfortunatamente, il numero di righe nella matrice A può essere una funzione esponenziale del numero di elementi di V. Infatti, nell'eliminazione di Fourier-Motzkin il numero di diseguaglianze che definiscono il poliedro proiettato può aumentare in modo esponenziale; nel primo passo si eliminano fino a $O(m)$ vincoli ma se ne aggiungono fino a $O(m^2)$, nel secondo passo questi ultimi possono dar luogo a $O(m^4)$ vincoli, e così via. Questo, peraltro, non è solo un rischio teorico, ma accade in effetti, a meno che i poliedri (coni) non siano strutturati in modo molto particolare; ciò è illustrato (per un poliedro) nell'Esempio 3.10, ed avrà rilevanza nel seguito.

Vogliamo adesso dimostrare la relazione inversa, ossia che qualsiasi cono poliedrico può essere espresso come cono finitamente generato. Per questo, una definizione cruciale è la seguente:

Definizione 3.1. Dato il cono (poliedrico) $C = \{ x \ : \ Ax \leq 0 \}$, il suo *cono duale* (finitamente generato) è $C^* = \{ x = \nu A \ : \ \nu \in \mathbb{R}^m_+ \}$.

Il cono duale C^* utilizza "gli stessi dati" di C, ma "in modo diverso". Ovviamente, C e C^* non sono uguali; in effetti, è facile dimostrare che

$$x \in C \quad , \quad d \in C^* \qquad \Longrightarrow \qquad dx \leq 0 \ ; \tag{3.10}$$

questo deriva immediatamente dal fatto che $x \in C \equiv Ax \leq 0$ e $d \in C^* \equiv d = \nu A$, quindi $dx = \nu Ax \leq 0$ perché $\nu \geq 0$. Geometricamente, questo significa che dato un qualsiasi vettore $d \in C^*$ l'intero cono C è compreso nel semispazio identificato da d, come mostrato ad esempio nella Figura 3.5. Algebricamente, ciò corrisponde al fatto che il vincolo $dx \leq 0$ è soddisfatto da tutti i punti di C, il che è ovvio perché tale vincolo si ottiene da una combinazione lineare dei vincoli $Ax \leq 0$ che definiscono C usando coefficienti non negativi. Una proprietà del cono duale che ci sarà utile nel seguito è:

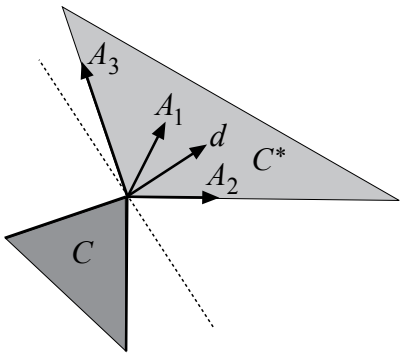


Figura 3.5: Un cono C ed il suo duale C^*

Lemma 3.2. $C^* = \{ x \ : \ Qx \leq 0 \}$ per un'opportuna matrice Q tale che tutte le righe di Q appartengono a C.

Dimostrazione L'esistenza della rappresentazione di C^* come cono poliedrico è data dal Teorema 3.2. Per il resto, è immediato verificare che $A_i \in C^*$ per ogni $i = 1, \ldots, m$ (si prenda $\nu_i = 1$, $\nu_h = 0$ per $h \neq i$); quindi $QA_i \leq 0$ per ogni $i = 1, \ldots, m$. In altre parole, per ogni riga Q_j si ha $Q_j A_i \leq 0$ per ogni $i = 1, \ldots, m$, ossia $Q_j \in C$. $\diamond$

Possiamo adesso dimostrare il risultato principale.

Teorema 3.3. Dato $C = \{ x \ : \ Ax \leq 0 \}$, esiste un insieme finito $V \subset \mathbb{R}^n$ per cui $C = cono(V)$.

Dimostrazione Si consideri $C^* = \{ x \ : \ Qx \leq 0 \}$ (cf. Lemma 3.2), ed il suo duale (il bi-duale di C)

$$C^{**} = \{ d = Q^T \nu \ : \ \nu \geq 0 \} \ ;$$

per il Teorema 3.2, $C^{**} = \{ x \ : \ Wx \leq 0 \}$ per una qualche matrice W. Vogliamo adesso dimostrare che $C^{**} = C$.

Per il Lemma 3.2, tutte le righe di Q sono elementi di C, quindi il loro inviluppo conico C^{**} è contenuto in C.

Viceversa, ogni $x \in C$ appartiene a C^{**}. Infatti, applicando ancora il Lemma 3.2 a C^{**} si ottiene che tutte le righe della matrice W appartengono a C^*; ciò per definizione di cono duale significa che data una qualsiasi W_j esistono moltiplicatori

$\nu_i^j \geq 0$ tali che

$$W_j = \sum_{i=1}^{m} \nu_i A_i \ .$$

Quindi, dal fatto che $A_i x \leq 0$ segue che $\sum_{i=1}^{m} \nu_i A_i x = W_j x \leq 0$, ossia $x \in C^{**}$. ◇

Esempio 3.7. Trasformazione da cono poliedrico a finitamente generato
Si vuole esprimere il cono poliedrico

$$\mathcal{C} = \left\{ \begin{array}{rcrcrcl} x_1 & & & & + & x_3 & \leq & 0 \\ & - & x_2 & - & x_3 & \leq & 0 \end{array} \right.$$

ottenuto nell'Esempio 3.6 come un cono finitamente generato. Per questo è sufficiente considerare il suo cono duale

$$\mathcal{C}^* = \left\{ \left[\begin{array}{c} x_1 \\ x_2 \\ x_3 \end{array} \right] = \alpha_1 \left[\begin{array}{c} 1 \\ 0 \\ 1 \end{array} \right] + \alpha_2 \left[\begin{array}{c} 0 \\ -1 \\ -1 \end{array} \right] \ , \ \alpha_1 \geq 0 \ , \ \alpha_2 \geq 0 \right\}$$

Come nell'Esempio 3.6, possiamo proiettare $\mathcal{C}^*$, un cono poliedrico in $\mathbb{R}^5$, sullo spazio $\mathbb{R}^3$ delle variabili originarie eliminando α_1 ed α_2. Per questo si potrebbe usare il metodo generale dell'eliminazione di Fourier-Motzkin, ma ancora una volta conviene procedere esaminando semplicemente il sistema corrispondente, ossia

$$\{ x_1 = \alpha_1 \ , \ x_2 = -\alpha_2 \ , \ x_3 = \alpha_1 - \alpha_2 \ , \ \alpha_1 \geq 0 \ , \ \alpha_2 \geq 0$$

Con semplici manipolazioni algebriche il sistema diventa

$$\{ \ -x_1 \leq 0 \ , \ x_2 \leq 0 \ , \ -x_1 - x_2 + x_3 \leq 0 \ , \ x_1 + x_2 - x_3 \leq 0$$

che è la rappresentazione canonica di $\mathcal{C}^*$ come cono poliedrico. Siccome $\mathcal{C}^{**} = \mathcal{C}$, si ottiene quindi

$$\mathcal{C} = cono\left(\left\{ \left[\begin{array}{c} -1 \\ 0 \\ 0 \end{array} \right] , \left[\begin{array}{c} 0 \\ 1 \\ 0 \end{array} \right] , \left[\begin{array}{c} -1 \\ -1 \\ 1 \end{array} \right] , \left[\begin{array}{c} 1 \\ 1 \\ -1 \end{array} \right] \right\} \right) \ ,$$

ossia il punto di partenza dell'Esempio 3.6. Questo illustra il fatto che "il duale del duale è il primale", ossia che applicando due volte l'operazione di dualità si ottiene di nuovo il cono originario.

La famiglia dei coni poliedrici coincide quindi con quella dei coni finitamente generati; questo risultato è alla base di un teorema fondamentale di caratterizzazione dei poliedri che presentiamo adesso. Si noti però che, ancora una volta, questa corrispondenza "non mantiene le dimensioni": una rappresentazione come cono finitamente generato di C si ottiene dalla rappresentazione $Qx \leq 0$ del suo duale C^* come cono poliedrico. Sfortunatamente questa rappresentazione può essere esponenzialmente più grande della rappresentazione come cono poliedrico di C^*, che ha le stesse dimensioni della rappresentazione di C (cf. Esempio 3.10). Questo significa che in pratica non è possibile determinare se esiste un elemento $d \in rec(P)$ tale che $cd > 0$ computando una rappresentazione di $rec(P)$ come cono finitamente generato e verificando il segno di tutti i prodotti scalari cv_i, perché questo potrebbe richiedere un tempo esponenziale nelle dimensioni di P. Questi risultati forniscono comunque spunti fondamentali per la progettazione di algoritmi efficienti, quali quelli che descriveremo nel seguito.

3.1.1.2 Il teorema di decomposizione dei poliedri

Possiamo adesso generalizzare i risultati precedenti all'intera classe dei poliedri. Il seguente teorema, dovuto a Motzkin, mostra infatti che *qualsiasi poliedro è la somma di un politopo più un cono*. In questo caso si sta utilizzando la definizione per cui dati due insiemi S e T in $\mathbb{R}^n$, entrambi non vuoti, la somma $Z = S + T$ è l'insieme di tutti quei punti z per cui esistano $x \in S$ e $y \in T$ tali che $z = x + y$. Il fatto che i due insiemi non siano vuoti può essere assunto senza perdita di generalità: infatti, è sempre possibile prendere uno dei due pari a $\{ 0 \}$, nel qual caso l'insieme somma coincide con l'altro insieme.

Teorema 3.4. L'insieme $P \subseteq \mathbb{R}^n$ è un poliedro se e solo se esistono due insiemi finiti non vuoti $X = \{ x^1, \ldots, x^s \} \subset \mathbb{R}^n$ e $V = \{ v^1, \ldots, v^t \} \subset \mathbb{R}^n$ per cui $P = conv(X) + cono(V)$. L'insieme $cono(V)$ coincide con $rec(P)$. In più, se P non contiene direzioni di linealità ($lin(P) = \{ 0 \}$), allora una rappresentazione minimale di X contiene tutti e soli i punti estremi di P.

Dimostrazione Le soluzioni $[\,x\,,\,\lambda\,,\,\nu\,]$ del sistema

$$x = \sum_{i=1}^{s} \lambda_i x^i + \sum_{j=1}^{t} \nu_j v^j \quad , \quad \sum_{i=1}^{s} \lambda_i = 1 \quad , \quad \lambda \geq 0 \quad , \quad \nu \geq 0 \tag{3.11}$$

che definisce $conv(\,X\,) + cono(\,V\,)$ formano un poliedro in $\mathbb{R}^{n+s+t}$; operando come nel Teorema 3.2, questo può essere proiettato sul sottospazio delle sole x. Viceversa, dato $P = \{\, x \,:\, Ax \leq b \,\}$ si consideri il cono

$$C = \left\{\, [\,x\,,\,\alpha\,] \,:\, Ax - b\alpha \leq 0\,,\ \alpha \geq 0 \,\right\} \subseteq \mathbb{R}^{n+1} \ ;$$

è chiaro che $P = \{\, x \,:\, [\,x\,,\,1\,] \in C \,\}$. Possiamo adesso applicare il Teorema 3.3 a C, ottenendo che esiste un insieme $Z = \{\, z_1\,,\,\ldots\,,\,z_k \,\} \subset \mathbb{R}^{n+1}$ tale che $C = cono(Z)$. Esplicitiamo adesso l'ultima componente di ciascuno di tali vettori come $z_i = [\,z_i'\,,\,z_i''\,]$ con $z_i' \in \mathbb{R}^n$ e $z_i'' \in \mathbb{R}$, ed assumiamo senza perdita di generalità che Z sia ordinato in modo tale che $z_i'' \neq 0$ per $i = 1,\ldots,s$, mentre $z_i'' = 0$ per $i = s+1,\ldots,k$. Si noti che, poiché $\alpha \geq 0$ per ogni $[\,x\,,\,\alpha\,] \in C$, deve sicuramente risultare $z_i'' > 0$ per ogni $i = 1,\ldots,s$, in quanto $\alpha = \sum_{i=1}^{s} \lambda_i z_i''$ e $\lambda \geq 0$. Definendo adesso $x_i = z_i' \,/\, z_i''$ per $i = 1,\ldots,s$ e $v_j = z_{s+j}'$ per $j = 1,\ldots,t = k-s$, si ottiene immediatamente (3.11) da $[\,x\,,\,1\,] \in cono(\,Z\,)$, dimostrando così la prima parte del teorema.

Mostriamo adesso che $cono(\,V\,) = rec(\,P\,)$. Il fatto che qualsiasi $d = \sum_{j=1}^{t} \nu_j v^j$ (per $\nu \geq 0$) appartenga al cono di recessione di P è ovvio dal fatto, appena dimostrato, che qualsiasi $x \in P$ può essere scritto nelle forma (3.11). Per l'implicazione inversa, si consideri che $d \in rec(\,P\,) \equiv Ad \leq 0 \equiv [\,d\,,\,0\,] \in C = cono(\,Z\,)$. Da ciò segue, con la partizione degli z_i e le definizioni date, che

$$d = \sum_{j=1}^{t} \nu_j v^j \quad , \quad 0 = \sum_{i=1}^{s} \lambda_i$$

e quindi $\lambda = 0$, da cui $d \in cono(\,V\,)$.

Infine, mostriamo che se $lin(\,P\,) = \{\,0\,\}$ allora una rappresentazione minimale di X contiene tutti e soli i punti estremi di P. Intanto, i punti estremi di P *devono* essere in X, poiché per definizione non c'è alcun modo di esprimerli come combinazione convessa dei altri punti $x \in P$. Assumiamo quindi per assurdo che X sia una rappresentazione minimale ma un certo $x_h \in X$ non sia un punto estremo, ossia

$$x_h = \gamma x' + (1-\gamma) x'' \quad , \quad x', x'' \in P \quad , \quad x' \neq x'' \quad , \quad \gamma \in (0,1) \ .$$

Per quanto dimostrato precedentemente si ha, per opportuni λ', ν', λ" e ν" (non negativi)

$$x' = \sum_{i=1}^{s} \lambda_i' x^i + \sum_{j=1}^{t} \nu_j' v^j \quad , \quad x'' = \sum_{i=1}^{s} \lambda_i'' x^i + \sum_{j=1}^{t} \nu_j'' v^j \ ;$$

definendo adesso

$$\lambda = \gamma\lambda' + (1-\gamma)\lambda'' \quad , \quad \nu = \gamma\nu' + (1-\gamma)\nu''$$

si ha chiaramente che $x_h = \sum_{i=1}^{s} \lambda_i x^i + \sum_{j=1}^{t} \nu_j v^j$, da cui

$$x_h(1-\lambda_h) = \sum_{i \in \{1,\ldots,s\} \setminus \{h\}} \lambda_i x^i + \sum_{j=1}^{t} \nu_j v^j \ . \tag{3.12}$$

Dobbiamo dividere adesso l'analisi in due casi distinti:

- $\lambda_h < 1$: dividendo entrambi i membri di (3.12) per $1/(1-\lambda_h) \geq 0$ e definendo $\bar\lambda_i = \lambda_i/(1-\lambda_h) \geq \lambda_i \geq 0$, $\bar\nu_j = \nu_j/(1-\lambda_h) \geq \nu_j \geq 0$ si ottiene

  $$x_h = \sum_{i \in \{1,\ldots,s\} \setminus \{h\}} \bar\lambda_i x^i + \sum_{j=1}^{t} \bar\nu_j v^j \quad , \quad \sum_{i \in \{1,\ldots,s\} \setminus \{h\}} \bar\lambda_i = 1$$

  come è facile verificare; ciò significa che $x_h \in conv(\,X' = X \setminus \{\, x_h \,\}\,) + cono(\,V\,)$, dal che segue facilmente che $P = conv(\,X'\,) + cono(\,V\,)$, contraddicendo la minimalità di X.

- $\lambda_h = 1$: questo implica che $\lambda_i = 0$ per $i \neq h$, e quindi usando (3.12) e la definizione di λ e ν si ottiene

  $$0 = \gamma \left(\sum_{j=1}^{t} \nu_j' v^j \right) + (1-\gamma) \left(\sum_{j=1}^{t} \nu_j'' v^j \right) = \gamma d' + (1-\gamma) d''$$

  dove chiaramente $d', d'' \in cono(\,V\,) = rec(\,P\,)$. Ma il fatto che $-\alpha d' = d'' \in rec(P)$ con $\alpha = \gamma/(1-\gamma) > 0$ implica immediatamente che $-d' \in rec(P)$, e quindi che $d' \in lin(\,P\,)$, contraddicendo l'ipotesi. L'unico caso a cui fare attenzione è che non possa essere $d' = d'' = 0$; considerato che si ha anche $\lambda_i' = \lambda_i'' = 0$ per ogni $i \neq h$ (altrimenti un qualche λ_i sarebbe $\neq 0$, essendo che $\gamma \in (\,0\,,\,1\,)$) questo implicherebbe che $x_h = x' = x''$, contro l'ipotesi.

Questo conclude la dimostrazione del teorema. ◇

Il significato geometrico del Teorema 3.4 è illustrato nella Figura 3.6, dove il poliedro P è dato dalla somma del triangolo $Q = conv(\,\{\, x^1\,,\,x^2\,,\,x^3 \,\}\,)$ con il cono $C = cono(\,\{\, v^1\,,\,v^2 \,\}\,)$. Si potrebbe anche dimostrare formalmente che, come suggerito dalla figura, una rappresentazione minimale di V usa solamente i *raggi estremi* del cono di recessione, definiti in modo analogo ai punti estremi; questo è lasciato per esercizio. Si noti che nel caso in cui $rec(\,P\,) = \{\,0\,\}$ si ha $P = conv(\,X\,)$. Viceversa, se P è un cono e $rango(\,A\,) = n$, ossia $lin(\,P\,) = \{\,0\,\}$, allora ha il solo vertice 0: quindi, $X = \{\,0\,\}$ e $P = cono(\,V\,) = rec(\,P\,)$.

Un altro esempio è fornito dal poliedro P di Figura 3.3, che può essere decomposto tramite i due insiemi finiti $X = \{[0,0,0],[0,1,0],[0,0,1],[0,1,1]\}$ e $V = ([1,0,0])$; quindi $Q = conv(X)$ è il "lato sinistro" del poliedro, mentre $C = cono(V)$ è il semiasse x_1 positivo, che è parallelo a tutti e quattro gli spigoli illimitati di P.

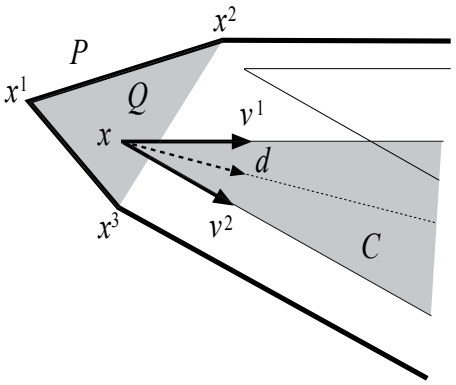


Figura 3.6: Decomposizione di un poliedro

Il Teorema 3.4 fornisce un'utile caratterizzazione teorica dei poliedri dalla quale possono essere fatte discendere molte proprietà, come ad esempio il fatto che P è compatto (e prende quindi il nome di *politopo*) se e solo se $rec(P) = \{0\}$, oppure il fatto che $conv(X)$ è il più piccolo poliedro che contiene tutti i punti di X. Dal punto di vista della PL, la proprietà più rilevante è la seguente:

Corollario 3.1. Sia $P = \{x : Ax \leq b\} = conv(X) + cono(V) \neq \emptyset$: allora il problema (3.1) ha ottimo finito se e solo se $cv^j \leq 0$ per $j = 1, \ldots, t$, ed in questo caso esiste un $h \in \{1, \ldots, s\}$ tale che x^h è una soluzione ottima del problema.

Dimostrazione Per il Teorema 3.4, il problema (3.1) è equivalente al seguente problema (di PL, ma con una forma particolarmente semplice dei vincoli) nelle variabili λ e ν:

$$\max\left\{ \sum_{i=1}^{s} \lambda_i (cx^i) + \sum_{j=1}^{t} \nu_j (cv^j) \ : \ \sum_{i=1}^{s} \lambda_i = 1 \ , \ \lambda \geq 0 \ , \ \nu \geq 0 \right\} .$$

Questo problema è ovviamente non vuoto, ed ha ottimo finito se e solo se $cv^j \leq 0$ per ogni $j = 1, \ldots, t$. Infatti, se fosse $cv^j > 0$ per un qualche indice j, allora facendo crescere ν_j all'infinito e fissando tutte le altre variabili il valore della funzione obiettivo crescerebbe anch'esso all'infinito. Se invece $cv^j \leq 0$ per ogni $j = 1, \ldots, t$, considerando un qualsiasi $x \in P$ ed i corrispondenti λ e ν risulta

$$cx = \sum_{i=1}^{s} \lambda_i (cx^i) + \sum_{j=1}^{t} \nu_j (cv^j) \leq \sum_{i=1}^{s} \lambda_i (cx^i) \leq \sum_{i=1}^{s} \lambda_i (cx^h) = cx^h$$

dove x^h è tale che $cx^h = \max\{cx^i : i = 1, \ldots, s\}$. Quindi il problema ha ottimo finito, ed inoltre x^h è una soluzione ottima di (3.1). ◇

Se $rango(A) = n$, questo teorema permette di affermare che tra le soluzioni ottime di un problema di PL, se ne esistono, c'è sempre almeno un punto estremo. Ciò renderebbe possibile, in teoria, studiare i problemi di PL tramite l'analisi dei soli punti estremi del poliedro e non di tutta la regione ammissibile.

Esempio 3.8. Risoluzione della PL tramite decomposizione

Consideriamo i tre vettori costo $c = [-1, 1, 1]$, $c' = [0, 1, 1]$ e $c'' = [1, 1, 1]$ per il poliedro P di Figura 3.3. Siccome $V = \{v_1 = [1, 0, 0]\}$, il Teorema 3.1 ci garantisce che i problemi di PL corrispondenti a c e c' hanno ottimo finito, mentre quello corrispondente a c'' è illimitato.

Infatti, è immediato verificare che l'unica soluzione ottima del problema corrispondente a c è il punto estremo $[\ 0, 1, 1\ ]$, mentre il problema corrispondente a c' ha infinite soluzioni ottime: tutti i punti dello spigolo "anteriore in alto" del parallelepipedo hanno lo stesso valore della funzione obiettivo, che risulta essere il valore ottimo. Questo corrisponde al fatto che $c'v_1 = 0$. Si noti che, comunque, tra i punti ottimi c'è almeno un punto estremo.

Per quanto riguarda il problema corrispondente al vettore c'', non esiste un valore ottimo: dato un qualsiasi punto ammissibile, ad esempio $[0, 0, 0]$, tutti i punti ottenuti muovendosi di un passo $\alpha \geq 0$ lungo il generatore del cono, ossia i punti della forma $[0, 0, 0] + \alpha[1, 0, 0] = [\alpha, 0, 0]$ sono ammissibili ed hanno valore della funzione obiettivo α; facendo crescere α si ottengono quindi punti ammissibili con valore della funzione obiettivo grande a piacere.

Se però $rango(A) < n$, ossia $lin(P) \neq \{0\}$, l'analisi precedente non si applica in quanto l'esistenza di direzioni di linealità è incompatibile con quella dei punti estremi. Questo è chiaramente visibile rimuovendo il vincolo (1) dalla definizione del poliedro di Figura 3.3: il corrispondente poliedro (la cui forma è già stata descritta) può essere decomposto tramite lo stesso insieme X già indicato e $V = \{[1, 0, 0], [-1, 0, 0]\}$, ma X non contiene punti estremi (in effetti esistono infiniti altri modi minimali di scegliere X). Questa condizione è potenzialmente problematica per lo sviluppo degli algoritmi che discuteremo, in quanto essi sono pensati per esplorare l'insieme dei vertici del poliedro. È però possibile dimostrare che in tal caso, senza perdita di generalità, ci si può ricondurre alla risoluzione di un problema definito su un poliedro che non ammette direzioni di linealità.

Teorema 3.5. Se $lin(P) \neq \{0\}$, ossia $rango(A) < n$, allora il problema (3.1) può essere risolto studiando in sua vece un diverso problema di *PL* la cui matrice dei coefficienti si ottiene eliminando una colonna da A.

Dimostrazione Se $rango(A) < n$, esiste almeno una colonna di A che può essere espressa come combinazione lineare delle altre; assumiamo senza perdita di generalità che si tratti dell'ultima. Abbiamo quindi che $A = [A', a^n]$, $c = [c', c_n]$, $x = [x', x_n]$, ed esiste un vettore $\mu \in \mathbb{R}^{n-1}$ tale che $A'\mu = a^n$. Questo mostra che $Ad = 0$ per $d = [\mu, -1] \neq 0$. Allora, il problema (3.1) può essere risolto studiando in sua vece il problema

$$\max\{c'x' : A'x' \leq b\} \ . \tag{3.13}$$

ossia rimuovendo l'ultima colonna di A e la corrispondente variabile. Per verificare questo notiamo che a qualsiasi soluzione ammissibile x' di (3.13) corrisponde una soluzione ammissibile $x = [x', 0]$ di (3.1) avente lo stesso costo, mentre da una qualsiasi soluzione ammissibile $x = [x', x_n]$ di (3.1) si può costruire la soluzione $x' + \mu x_n$ ammissibile per (3.13). Da questo segue che:

- se (3.13) non ha soluzione ammissibile allora non può averla nemmeno (3.1);
- se (3.13) è superiormente illimitato allora tale è pure (3.1).

Rimane adesso da esaminare il caso in cui si determini una soluzione ottima $\bar{x}'$ di (3.13). In questo caso, se $cd \neq 0$, allora (3.1) è superiormente illimitato: $x(\alpha) = [\bar{x}', 0] + \alpha d$ è ammissibile per ogni α, ed è sempre possibile scegliere α in modo opportuno (> 0 se $cd > 0$, < 0 se $cd < 0$) affinché $cx(\alpha)$ sia superiore a qualsiasi soglia fissata. Se invece $cd = 0 \equiv c_n = c'\mu$, (3.1) e (3.13) hanno lo stesso valore ottimo: infatti, data qualsiasi soluzione ammissibile $x = [x', x_n]$ di (3.1), la soluzione ammissibile corrispondente $x' + \mu x_n$ di (3.13) ha lo stesso costo

$$c'(x' + \mu x_n) = c'x' + (c'\mu)x_n = c'x' + c_n x_n \ .$$

Viceversa, a qualsiasi soluzione x' di (3.13) corrisponde una soluzione $x = [x', 0]$ di (3.1) con lo stesso costo. Pertanto, $\bar{x} = [\bar{x}', 0]$ è ottima per (3.1). ◇

Esempio 3.9. Riduzione al caso della matrice di rango pieno

Consideriamo ancora il poliedro P' ottenuto da quello di Figura 3.3 rimuovendo il vincolo (1); per quanto riguarda l'ottimizzazione su P', la variabile x_1 del problema può essere eliminata e trattata implicitamente lavorando sul politopo $\bar{P}' \subset \mathbb{R}^2$ definito da

$$A' = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & -1 \\ -1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix} \ ,$$

ossia il quadrato (ipercubo) unitario nello spazio $[x_2, x_3]$ (si noti che $\bar{P}'$ ha punti estremi, e $rango(A') = 2$). Dato qualsiasi punto $[x_2, x_3]$ ammissibile per $\bar{P}'$, è immediato costruire un punto $[x_1, x_2, x_3]$ ammissibile per P. Pertanto, se il coefficiente di x_1 in funzione obiettivo è diverso da zero, allora il problema è superiormente illimitato; altrimenti è possibile fissare x_1 in modo arbitrario (ad esempio al valore 0) e risolvere il problema ristretto alle rimanenti due variabili.

In ogni caso, la risoluzione di un problema di *PL* tramite l'enumerazione di tutti i punti ed i raggi estremi del poliedro corrispondente è chiaramente non praticabile per problemi che non siano di piccole dimensioni. Infatti, anche qualora si assuma che la matrice A che caratterizza il poliedro abbia un numero di righe "ragionevolmente contenuto" (per quanto questo non sia sempre vero, né strettamente necessario, come vedremo), il numero di punti estremi del poliedro può essere esponenzialmente grande. Ciò è illustrato nel seguente esempio, che mostra come la dimensionalità delle reppresentazioni "per facce" e "per punti estremi" dei poliedri possa (e, normalmente, sia) drammaticamente differente. L'esempio mostra anche come la proiezione di un poliedro in uno spazio a dimensione più piccola possa aumentare esponenzialmente il numero di diseguaglianze necessarie per esprimerlo (in altre parole "poche variabili equivalgono a molti vincoli"), ossia che l'eliminazione di Fourier-Motzkin possa in effetti portare ad un incremento esponenziale del numero di vincoli, come già anticipato nel §3.1.1.1.

Esempio 3.10. La proiezione dell'n-co-cubo

Consideriamo la palla di raggio unitario e centro l'origine nella norma L_1, ossia l'insieme $B_1 = \{x \in \mathbb{R}^n : \|x\|_1 \leq 1\}$. Con la tecnica vista nell'Esempio 1.19 questo si può scrivere come

$$B_1 = \left\{ x \in \mathbb{R}^n : \sum_{i=1}^n v_i \leq 1 \ , \ -v_i \leq x_i \leq v_i \ \ i = 1, \ldots, n \right\}$$

ossia un poliedro in $\mathbb{R}^{2n}$ con "pochi" $(2n+1)$ vincoli. Utilizzando l'eliminazione di Fourier-Motzkin si vuole scrivere B_1 come un poliedro in $\mathbb{R}^n$. Per semplicità svolgeremo l'esempio solamente per $n = 3$, ossia

$$B_1 = \{\, x \in \mathbb{R}^3 \,:\, v_1 + v_2 + v_3 \le 1 \,,\; -v_1 \le x_1 \le v_1 \,,\; -v_2 \le x_2 \le v_2 \,,\; -v_3 \le x_3 \le v_3 \,\}$$

in quanto la generalizzazione ad n qualsiasi è poi ovvia.

Iniziamo proiettando la variabile ausiliaria v_1, che appare in tre vincoli: $v_1 + v_2 + v_3 \le 1$ (con coefficiente 1, quindi in I^+) $-x_1 - v_1 \le 0$, $x_1 - v_1 \le 0$ (entrambi con coefficiente -1, quindi in I^-); tutti gli altri vincoli andranno quindi in I^0 e passeranno senza modifiche allo step successivo. Seguendo la procedura di Fourier-Motzkin riscriviamo i tre vincoli come

$$v_1 \le 1 - v_2 - v_3 \;\;,\;\; -x_1 \le v_1 \;\;,\;\; x_1 \le v_1$$

e poi li combiniamo ottenendone due

$$-x_1 \le 1 - v_2 - v_3 \;\;,\;\; x_1 \le 1 - v_2 - v_3$$

che sostituiamo ai tre originali nella definizione, terminando il primo passo.

Proiettiamo adesso la variabile ausiliaria v_2, che appare nei due vincoli $-x_2 \le v_2$ e $x_2 \le v_2$ (in I^-) così come nei due vincoli appena creati

$$-x_1 + v_2 + v_3 \le 1 \;\;,\;\; x_1 + v_2 + v_3 \le 1 \;\;,$$

entrambi in I^+. Occorre quindi creare tutte le possibili 2×2 coppie di tali vincoli, il che ne produce i seguenti quattro

$$-x_1 - x_2 \le 1 - v_3 \;\;,\;\; x_1 - x_2 \le 1 - v_3 \;\;,\;\; -x_1 + x_2 \le 1 - v_3 \;\;,\;\; x_1 + x_2 \le 1 - v_3 \;\;.$$

Al terzo ed ultimo passo proiettiamo v_3, che appare come al solito in due soli vincoli in I^-, $-x_3 \le v_3$ e $x_3 \le v_3$, ma in tutti e quattro i vincoli in I^+ appena creati. Formando le 4×2 coppie di tali vincoli si determinano gli 8

$$\begin{array}{c} -x_1 - x_2 - x_3 \le 1 \;\;,\;\; x_1 - x_2 - x_3 \le 1 \;\;,\;\; -x_1 + x_2 - x_3 \le 1 \;\;,\;\; x_1 + x_2 - x_3 \le 1 \\ -x_1 - x_2 + x_3 \le 1 \;\;,\;\; x_1 - x_2 + x_3 \le 1 \;\;,\;\; -x_1 + x_2 + x_3 \le 1 \;\;,\;\; x_1 + x_2 + x_3 \le 1 \end{array}$$

che finalmente forniscono la rappresentazione di B_1 in $\mathbb{R}^3$. È facile vedere che questi vincoli possono essere rappresentati con la forma

$$\pm x_1 \pm x_2 \pm x_3 \le 1$$

ossia in tutti i vincoli sono presenti tutte le variabili originali con tutte le possibili $2^3 = 8$ combinazioni di coefficienti 1 e -1. Generalizzando ad un qualsiasi n si ottiene

$$B_1 = \{\, x \in \mathbb{R}^n \,:\, \pm x_1 \pm x_2 \ldots \pm x_n \le 1 \,\}$$

ossia un poliedro che ha 2^n vincoli in n variabili al posto dei $2n + 1$ vincoli in $2n$ variabili della forma originale, mostrando come il numero di vincoli può crescere esponenzialmente (e ciò avviene in effetti facilmente in pratica) con il numero di variabili eliminate.

B_1 viene detto "n-co-cubo" in quanto è il poliedro "duale" dell'ipercubo unitario $[\,-1\,,\,1\,]^n$. Non approfondiremo questa relazione, notando solo che l'ipercubo unitario ha $2n$ vincoli ($-1 \le x_i \le 1$ per $i = 1, \ldots, n$) ma notoriamente 2^n vertici, che hanno la forma $[\,\pm 1\,,\,\pm 1\,,\,\ldots\,,\,\pm 1\,]$; dualmente B_1 ha 2^n vincoli, uno per ciascun vertice dell'ipercubo unitario, ma solamente $2n$ vertici, uno per ciascun vincolo dell'ipercubo unitario (che si può facilmente verificare avere la forma $[\,0\,,\,\ldots\,,\,\pm 1\,,\,\ldots\,0\,]$ per tutti gli n modi possibili di scegliere la posizione del ± 1). Ciò incidentalmente mostra come i poliedri possano avere sia "molti vertici e pochi lati" che "pochi vertici e molti lati", ma il primo caso è (purtroppo, come vedremo) nettamente più frequente nelle applicazioni.

3.1.1.3 L'algebra dei vincoli e delle facce

Vogliamo adesso riesaminare i concetti geometrici sviluppati nei paragrafi precedenti con strumenti algebrici. Dato il poliedro $P = \{\, x \,:\, Ax \le b \,\}$, sia $I \subseteq \{\, 1\,,\,\ldots\,,\,m \,\}$ un qualsiasi sottoinsieme dell'insieme degli indici di riga; indichiamo allora con A_I e b_I, rispettivamente, la sottomatrice di A ed il sottovettore di b ristretti alle righe i cui indici sono in I. Indicando con $\bar{I} = \{\, 1\,,\,\ldots\,,\,m \,\} \setminus I$ il complemento di I, l'insieme

$$P_I = \{\, x \,:\, A_I x = b_I \;\;,\;\; A_{\bar{I}} x \le b_{\bar{I}} \,\}$$

è chiaramente un sottoinsieme di P: se $P_I \neq \emptyset$, tale insieme viene detto *faccia* di P. Da (3.2.*ii*) segue immediatamente che ogni faccia di un poliedro è a sua volta un poliedro. Il numero di facce distinte di un poliedro è al più pari al numero di sottoinsiemi distinti di $\{\, 1\,,\,\ldots\,,\,m \,\}$, e quindi è finito, anche se potenzialmente esponenziale in m. Osserviamo che, se si sceglie $I = \emptyset$, si ottiene una faccia particolare che è il poliedro stesso.

Una faccia propria (cioè non coincidente con tutto il poliedro P) che non sia contenuta in nessun'altra

faccia è detta *faccia massimale* o *faccetta*, mentre una faccia che non contenga nessuna faccia distinta da essa è detta *faccia minimale*. Si noti che $P_A \subseteq P_B$ se $B \subseteq A$: quindi le facce massimali corrispondono ad insiemi "piccoli" di indici I, mentre le facce minimali corrispondono ad insiemi "grandi" di indici I. La *dimensione* di una faccia è definita come la dimensione del più piccolo sottospazio (affine) che la contiene; è possibile verificare che una faccia determinata da una matrice A_I di rango k ha dimensione $n-k$ o inferiore.

Esempio 3.11. Facce e faccette in Figura 3.3

Per il poliedro P in Figura 3.3, le facce $P_{\{1\}}$, $P_{\{2\}}$, $P_{\{3\}}$, $P_{\{4\}}$ e $P_{\{5\}}$ sono tutte e sole le faccette del poliedro; esse corrispondono rispettivamente al "lato sinistro", al "lato superiore", al "lato posteriore", al "lato inferiore" ed al "lato anteriore" del parallelepipedo. Tutte queste facce hanno dimensione $n-k=3-1=2$, infatti il più piccolo sottospazio che contiene ciascuna di esse è un piano. Invece la faccia $P_{\{6\}}$ non è una faccetta del poliedro, ed ha dimensione $1 < n-k=2$.

Si noti che la faccia $P_{\{6\}}$ coincide con le facce $P_{\{2,5\}}$, $P_{\{2,6\}}$ e $P_{\{5,6\}}$ ed ognuna di esse individua lo spigolo "anteriore in alto" del poliedro; si osservi che se è vero che ad ogni insieme $I \subset \{1, \ldots, m\}$ può corrispondere una faccia, è anche vero che ad insiemi diversi può corrispondere la stessa faccia. Inoltre, non a tutti gli insiemi I corrisponde necessariamente una faccia: questo è il caso ad esempio di $P_{\{3,6\}}$, che è vuoto, in quanto i vincoli (3) e (6) individuano la retta $x_3 = 0, x_2 = 2$ che ha intersezione vuota con il poliedro.

Le facce determinate da sottomatrici A_I di rango n, se ne esistono, hanno dimensione 0, cioè sono punti: infatti, in questo caso il sistema lineare $A_I x = b_I$ ammette una ed una sola soluzione. Tali facce, dette i *vertici* di P, sono ovviamente minimali, e si può verificare che questo concetto coincide con quello di punto estremo precedentemente analizzato. A tal fine introduciamo adesso alcuni concetti che ci saranno comunque utili nel seguito.

Dato un punto $\bar{x} \in P$, i vincoli che vengono soddisfatti da $\bar{x}$ come uguaglianze vengono detti vincoli *attivi* in $\bar{x}$; indichiamo con $I(\bar{x})$ l'*insieme degli indici dei vincoli attivi*:

$$I(\bar{x}) = \{ i : A_i \bar{x} = b_i \} \ .$$

Osserviamo che $P_{I(\bar{x})}$ è una faccia del poliedro che contiene $\bar{x}$. In generale, qualsiasi $I \subseteq I(\bar{x})$ definisce una faccia P_I che contiene $\bar{x}$: chiaramente, $I(\bar{x})$ definisce la faccia minimale tra tutte queste. Un punto $\bar{x}$ è *interno* ad una faccia P_I se è contenuto in P_I ma non è contenuto in nessuna faccia propria di P_I; questo vuol dire che $A_i \bar{x} < b_i$ per ogni $i \notin I$. Ovviamente, $\bar{x}$ è interno alla faccia $P_{I(\bar{x})}$.

Dato $\bar{x} \in P$, un vettore $\xi \in \mathbb{R}^n$ è detto una *direzione ammissibile* per $\bar{x}$ se esiste un $\bar{\lambda} > 0$ per cui $x(\lambda) = \bar{x} + \lambda\xi$ è ammissibile per (P) per ogni $\lambda \in [0, \bar{\lambda}]$, cioè per ogni $i = 1, \ldots, m$ vale

$$A_i x(\lambda) = A_i \bar{x} + \lambda A_i \xi \leq b_i \ . \tag{3.14}$$

Chiaramente, qualsiasi direzione $\xi \in rec(P)$ è una direzione ammissibile per qualsiasi $\bar{x}$, ma in generale l'insieme è più grande. In particolare, è facile verificare che per ogni $i \in I(\bar{x})$, ovvero per cui $A_i \bar{x} = b_i$, (3.14) è verificata se e solo se $A_i \xi \leq 0$. Se invece $i \notin I(\bar{x})$, ovvero $A_i \bar{x} < b_i$, allora (3.14) è verificata da *ogni direzione* $\xi \in \mathbb{R}^n$ purché il passo λ sia sufficientemente piccolo. Possiamo pertanto caratterizzare algebricamente le direzioni ammissibili nel modo seguente:

Proprietà 3.1. ξ è una direzione ammissibile per $\bar{x}$ se e solo se $\xi \in C(\bar{x}) = \{ d \in \mathbb{R}^n : A_{I(\bar{x})} d \leq 0 \}$.

L'insieme $C(\bar{x})$ di tutte le direzioni ammissibili per $\bar{x}$ è perciò un cono poliedrico; si noti che se $\bar{x}$ è un punto interno al poliedro ($A\bar{x} < b \equiv I(\bar{x}) = \emptyset$) allora $C(\bar{x}) = \mathbb{R}^n$: qualunque vettore $\xi \in \mathbb{R}^n$ è una direzione ammissibile per $\bar{x}$. Possiamo adesso dimostrare il risultato annunciato:

Teorema 3.6. $\bar{x}$ è un punto estremo di P se e solo se $A_I \bar{x} = b_I$ per una sottomatrice A_I di rango n.

Dimostrazione La dimostrazione si basa sulla seguente caratterizzazione: $\bar{x}$ è un punto estremo se e solo se il cono delle direzioni ammissibili $C(\bar{x})$ per $\bar{x}$ non contiene direzioni di linealità, il che a sua volta equivale a $rango(A_{I(\bar{x})}) = n$. Quindi, per ogni punto estremo $\bar{x}$ esiste $I = I(\bar{x})$ con $rango(A_I) = n$ tale che $P_I = \{\bar{x}\}$, e, viceversa, qualsiasi faccia P_I con $rango(A_I) = n$ individua un punto estremo. ◇

Come ulteriore verifica, si noti che se $rango(A) < n$ allora $rango(A_I) < n$ per ogni insieme I, ossia le facce minimali hanno dimensione maggiore di zero; infatti sappiamo che questo coincide con il fatto che P possegga direzioni di linealità, e quindi non possegga vertici.

Un insieme di indici B, di cardinalità n, tale che la sottomatrice quadrata A_B sia invertibile viene detto una *base*; A_B viene detta *matrice di base* corrispondente a B, e $\bar{x} = A_B^{-1} b_B$ viene detta *soluzione di base*. Se $\bar{x} \in P$ allora B viene detta *ammissibile*; se invece $\bar{x} \notin P$, B viene detta *non ammissibile*. Dall'analisi precedente segue immediatamente:

Corollario 3.2. $\bar{x}$ è un punto estremo di P se e solo se esiste una base B per cui $\bar{x} = A_B^{-1} b_B$.

Se $I(\bar{x})$ è una base, e $|I(\bar{x})| = n$, allora il vertice (e con esso la base) si dice *non degenere*; altrimenti ($|I(\bar{x})| > n$) il vertice si dice *degenere*. Le facce individuate da sottomatrici A_I di rango $n-1$ hanno dimensione (al più) 1, sono cioè segmenti se sono limitate, semirette o rette altrimenti; tali facce sono dette *spigoli* di P.

Esempio 3.12. Basi e vertici in Figura 3.3
Ancora con riferimento al poliedro P in Figura 3.3, i quattro vertici $[0, 0, 0]$, $[0, 1, 0]$, $[0, 0, 1]$ e $[0, 1, 1]$ corrispondono a opportune basi. Infatti, $[0, 0, 0]$ corrisponde alla base $\{1, 3, 4\}$, $[0, 0, 1]$ corrisponde alla base $\{1, 4, 5\}$, $[0, 1, 0]$ corrisponde alla base $\{1, 2, 3\}$, e $[0, 1, 1]$ corrisponde alle basi $\{1, 2, 5\}$, $\{1, 2, 6\}$ e $\{1, 5, 6\}$. Si noti quindi che mentre una base ammissibile identifica univocamente un vertice del poliedro, il viceversa non è vero: esistono vertici ai quali sono associate più basi. Esistono anche basi non ammissibili, ossia che non corrispondono a vertici del poliedro: nel nostro esempio questo è il caso delle basi $\{1, 3, 6\}$ e $\{1, 4, 6\}$, che individuano le soluzioni di base non ammissibili $[0, 2, 0]$ e $[0, 0, 2]$. I tre vertici $[0, 0, 0]$, $[0, 1, 0]$ e $[0, 0, 1]$ sono non degeneri, mentre $[0, 1, 1]$ è degenere.

Il poliedro ha quattro spigoli limitati, corrispondenti al "lato sinistro", e quattro spigoli illimitati, corrispondenti ai "lati infiniti" del parallelepipedo.

Scopo fondamentale del nostro studio è sviluppare algoritmi per risolvere i problemi di *PL*. Assumendo senza perdita di generalità che la matrice dei coefficienti A abbia rango massimo, come consentito dal Teorema 3.5, abbiamo come conseguenza del Corollario 3.1 che un tale algoritmo può limitarsi a visitare i vertici del poliedro corrispondente; infatti, se il problema non è superiormente illimitato (né vuoto) allora esiste sempre una soluzione ottima che coincide con un vertice. Il Corollario 3.2 fornisce un meccanismo algebrico per generare un vertice del poliedro in modo "efficiente", ossia al costo della soluzione di un sistema lineare $n \times n$ ($O(n^3)$ per matrici dense non strutturate usando metodi elementari, molto meno in pratica).

Anche i raggi estremi di $rec(P)$, che è pure (in linea di principio) necessario esaminare ai sensi del Teorema 3.1, possono essere enumerati attraverso basi. Ciò discende da una proprietà particolare dei coni poliedrici associati alle matrici di base, detti *coni simpliciali*, enunciata (e dimostrata) dalla seguente serie di semplici equivalenze:

$$\{ x : A_B x \leq 0 \} = \{ x : -A_B x = \nu , \nu \geq 0 \} = \{ x = -A_B^{-1} \nu : \nu \geq 0 \} . \tag{3.15}$$

Questo significa che mentre per la famiglia dei coni poliedrici la conversione in cono finitamente generato, sebbene sempre possibile (cf. Teorema 3.3), può avere un costo esponenziale in quanto il cono finitamente generato può avere un numero di generatori esponenziale, i coni simpliciali hanno esattamente n generatori, corrispondenti all'opposto delle colonne della matrice inversa A_B^{-1}, e quindi una loro descrizione si può calcolare con complessità $O(n^3)$ o inferiore. Si potrebbe anche dimostrare (cosa che eviteremo di fare) l'equivalente del Corollario 3.2:

Teorema 3.7. Dato un cono poliedrico $C = \{ x : Ax \leq 0 \}$, v è un generatore di C se e solo se esiste una base B tale che v è un generatore del *cono di base* $C_B = \{ x : A_B x \leq 0 \}$ (che si ottiene quindi attraverso (3.15)) e $v \in C$.

Esempio 3.13. Generatori di un cono simpliciale
Si consideri il poliedro P in Figura 3.3 e la base $B = \{1, 4, 5\}$. Si ha quindi

$$A_B = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad , \quad b_B = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad , \quad -A_B^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} .$$

Calcolando $A_B^{-1}b_B$ si ottiene infatti il vertice $[\,0\,,\,0\,,\,1\,]$ del poliedro; inoltre, il cono simpliciale associato alla base,

$$-x_1 \leq 0 \quad , \quad -x_2 \leq 0 \quad , \quad x_3 \leq 0$$

corrisponde al cono finitamente generato dai vettori

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \quad , \quad \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \quad , \quad \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$$

ossia l'opposto delle colonne di A_B^{-1} (che in questo caso, ma solamente per caso, coincidono con le colonne di A_B). In questo caso, poiché il vertice non è degenere, i generatori del cono individuano i tre spigoli del poliedro incidenti nel vertice; ciò, come vedremo, sarà fondamentale per gli algoritmi. È possibile vedere che non sempre i generatori dei coni simpliciali associati a basi ammissibili (che individuano vertici del poliedro) individuano spigoli del poliedro: si lascia per esercizio di verificare che, ad esempio, questo non capita per tutti i generatori del cono simpliciale corrispondente alla base $\{\,1\,,\,3\,,\,6\,\}$ che individua il vertice degenere $[\,0\,,\,1\,,\,1\,]$. Questa occorrenza verrà discussa nel dettaglio nel seguito.

Ovviamente, enumerare tutte le basi non porta ad un approccio efficiente, perché il numero di basi è esponenziale in n ed m. Infatti, gli algoritmi che discuteremo non enumerano ciecamente le basi, ma cercano di "esplorare lo spazio delle basi" in modo "guidato", e quindi più efficiente.

3.2 Teoria della Dualità

I problemi di *PL* hanno complessità polinomiale: una motivazione risiede nel fatto che esistono metodi efficienti per verificare se una data soluzione sia o meno ottima. In generale, per fare questo servono tecniche che permettano di valutare il valore ottimo della funzione obiettivo del problema. Nel caso di un problema di *PL*, questo avviene con l'ausilio di un altro problema di *PL*, detto *duale* del problema dato.

3.2.1 Coppie di problemi duali

Prima di formalizzare il concetto di problema duale, analizziamo due esempi.

Esempio 3.14. Un problema di polli e pillole

L'allevamento (lager) "Pollo Felice" (PF) usa due tipi di cereali, A e B per il mangime. Per ottimizzare il benessere (il numero di uova prodotte e la velocità con cui crescono e quindi possono essere uccisi e venduti) dei polli, il mangime deve soddisfare certi requisiti nutritivi: deve contenere almeno 8 unità di carboidrati, 15 unità di proteine e 3 unità di vitamine per unità di peso. Il contenuto unitario di carboidrati, proteine e vitamine ed il costo unitario di A e B sono riportati nella seguente tabella, insieme ai requisiti minimi giornalieri.

	A	B	min. giornaliero
carboidrati	5	7	8
proteine	4	2	15
vitamine	2	1	3
costo unitario	1200	750	

Siano x_1 ed x_2, rispettivamente, il numero di unità di cereale di tipo A e B impiegate nel mangime; il numero di unità di carboidrati presenti nel mangime è allora dato da $5x_1 + 7x_2$, e poiché il fabbisogno minimo di carboidrati è di 8 unità, deve risultare $5x_1 + 7x_2 \geq 8$; analogamente, per le unità di proteine deve risultare $4x_1 + 2x_2 \geq 15$ e per le unità di vitamine $2x_1 + x_2 \geq 3$. Ai tre vincoli precedenti si devono aggiungere le ovvie condizioni di non negatività delle variabili, $x_1 \geq 0$, $x_2 \geq 0$. Infine, la funzione obiettivo è $1200x_1 + 750x_2$. La dieta di costo minimo per la PF è data dunque da una soluzione del seguente problema di *PL*:

$$\begin{array}{lrcrcl} \min & 1200x_1 & + & 750x_2 & & \\ & 5x_1 & + & 7x_2 & \geq & 8 \\ & 4x_1 & + & 2x_2 & \geq & 15 \\ & 2x_1 & + & x_2 & \geq & 3 \\ & x_1 & , & x_2 & \geq & 0 \end{array}$$

Al problema in esame è "naturalmente" associato un altro problema. Il venditore di cibo in pillole "Soylent Green" (SG) vuole entrare in forza nel mercato del mangime per animali, cominciando da quello per i polli. Deve quindi stabilire i prezzi di vendita di pillole rispettivamente di carboidrati, proteine e vitamine in modo che il ricavato della vendita sia massimo e che i prezzi siano competitivi, ossia che PF ritenga non svantaggioso acquistare le pillole

invece dei cereali. Supponiamo che ciascuna pillola contenga un'unità del corrispondente elemento nutritivo, e siano y_1, y_2 e y_3 i prezzi, rispettivamente, di una pillola di carboidrati, proteine e vitamine: poiché PF deve percepire la dieta a base di pillole non più costosa di quella a base di cereali, dovrà risultare $5y_1 + 4y_2 + 2y_3 \leq 1200$, cioè il costo dell'equivalente (da un punto di vista nutritivo) in pillole del cereale A deve essere non superiore a 1200. Analogamente, per il cereale B si ottiene $7y_1 + 2y_2 + y_3 \leq 750$. Per non rischiare guai con l'antitrust (che non permetterebbe di vendere alcune pillole in perdita per aiutare l'entrata nel mercato) i prezzi di vendita devono essere non negativi. Il ricavato della vendita è dato $8y_1 + 15y_2 + 3y_3$, dato che 8, 15 e 3 sono il minimo numero di pillole di carboidrati, proteine e vitamine necessari alla corretta alimentazione del pollo. Il problema del venditore di pillole per polli è dunque

$$\begin{array}{lrcrcrcl} \max & 8y_1 & + & 15y_2 & + & 3y_3 & & \\ & 5y_1 & + & 4y_2 & + & 2y_3 & \leq & 1200 \\ & 7y_1 & + & 2y_2 & + & y_3 & \leq & 750 \\ & y_1 & , & y_2 & , & y_3 & \geq & 0 \end{array}$$

I due problemi sono riassunti nella seguente tabella

	x_1	x_2		max
y_1	5	7		8
y_2	4	2	$\geq$	15
y_3	2	1		3
	$\mid\wedge$			
min	1200	750		

La coppia di problemi appena costruita gode di un'importante proprietà: comunque si scelgano i prezzi y_1, y_2 e y_3 delle pillole, il ricavo si SG è sicuramente minore o uguale del costo di *qualsiasi* dieta ammissibile che PF possa ottenere dai due mangimi. Infatti, i vincoli di PF assicurano che le pillole siano più convenienti dei singoli mangimi, e quindi anche di ogni loro combinazione. La proprietà può essere verificata algebricamente nel seguente modo: moltiplicando per x_1 e x_2 i due vincoli del problema di SG e sommando le due diseguaglianze così ottenute, si ha

$$x_1(5y_1 + 4y_2 + 2y_3) + x_2(7y_1 + 2y_2 + y_3) \leq 1200x_1 + 750x_2 \ .$$

Riordinando i termini in modo da mettere in evidenza le variabili y_i, si ottiene

$$y_1(5x_1 + 7x_2) + y_2(4x_1 + 2x_2) + y_3(2x_1 + x_2) \leq 1200x_1 + 750x_2 \ .$$

A questo punto possiamo utilizzare i vincoli del problema di PF per minorare le quantità tra parentesi, ottenendo

$$8y_1 + 15y_2 + 3y_3 \leq 1200x_1 + 750x_2 \ .$$

Il costo di una qualsiasi dieta ammissibile fornisce quindi una valutazione superiore del massimo ricavo per SG e, analogamente, qualsiasi ricavo ammissibile per SG fornisce una valutazione inferiore del costo della miglior dieta possibile per PF. Ciascuno dei due problemi fornisce quindi, attraverso il costo di qualsiasi soluzione ammissibile, una valutazione (inferiore o superiore) del valore ottimo dell'altro problema.

Esempio 3.15. Un problema di birra e prezzi

Il primo produttore mondiale di birra, BugWasser (BW) deve organizzare il trasporto della birra dalle sue n fabbriche, $F = \{1, \ldots, n\}$, ai suoi m depositi $D = \{1, \ldots, m\}$. È nota la capacità produttiva a_i di ciascuna fabbrica $i \in F$ e la capacità di ciascun deposito $j \in D$ (in migliaia di tonnellate), così come il costo unitario di trasporto c_{ij} dalla fabbrica i al deposito j. Assumiamo per semplicità che sia $\sum_{i \in F} a_i = \sum_{j \in D} b_j$, cioè che la produzione totale eguagli la capacità totale dei depositi (si veda §2.1.2 per il caso in cui ciò non fosse vero): si vuole trasportare la birra dalle fabbriche ai depositi in modo da minimizzare il costo di trasporto.
Indicando con x_{ij} la quantità di birra trasportata da i a j, oltre alle ovvie condizioni di non negatività delle variabili ($x_{ij} \geq 0$) dobbiamo imporre i vincoli che

- da ogni fabbrica $i \in F$ venga trasportata tutta la birra prodotta, e cioè $\sum_{j \in D} x_{ij} = a_i$;
- la birra trasportata ad ogni deposito $j \in D$ saturi la sua capacità, e cioè $\sum_{i \in F} x_{ij} = b_j$.

Il costo di trasporto è $\sum_{i \in F} \sum_{j \in D} c_{ij} x_{ij}$, quindi il problema può essere quindi formulato come

$$\begin{array}{ll} \min & \sum_{i \in F} \sum_{j \in D} c_{ij} x_{ij} \\ & \sum_{j \in D} x_{ij} = a_i \qquad i \in F \\ & \sum_{i \in F} x_{ij} = b_j \qquad j \in D \\ & x_{ij} \geq 0 \qquad (i, j) \in F \times D \end{array} \ .$$

Si tratta chiaramente di un particolare problema di Flusso di Costo Minimo (cf. §1.2.6 e §2.1). In particolare, il grafo sottostante è bipartito (come nel caso dei problemi di assegnamento, cf. §2.6, ma con domande ed offerte generiche), che viene detto *problema di trasporto* (transportation problem).

Anche in questo caso possiamo definire un altro problema che utilizza gli stessi dati del problema di trasporto e che è ad esso strettamente legato. La ditta di logistica globale EIM vuole sfruttare la sua infrastruttura esistente (navi, aerei, camion, ...) per sfondare nel lucrativo mercato dei trasporti di alcolici, iniziando da quello della birra che ha i volumi più alti. Per questo EIM si offre a BW come alternativa al trasporto "in house": per ogni (i, j) comprerà la birra alla fabbrica $i \in F$ al prezzo unitario λ_i, rivendendola al deposito $j \in D$ al prezzo unitario μ_j. Il problema di EIM è quindi di definire i valori di λ_i e μ_j in modo da massimizzare il guadagno ottenuto dall'operazione, che è dato da $-\sum_{i\in F} a_i\lambda_i + \sum_{j\in D} b_j\mu_j$ (ricavo dalla vendita meno costo dell'acquisto). Affinché tutte le offerte vengano accettate, però, l'offerta deve risultare per BW competitiva rispetto al costo di trasporto "in house"; pertanto dovrà risultare $-\lambda_i + \mu_j \le c_{ij}$ per ogni $(i, j) \in F \times D$. Il problema di EIM può quindi essere formulato come

$$\begin{array}{ll} \max & -\sum_{i\in F} a_i\lambda_i + \sum_{j\in D} b_j\mu_j \\ & -\lambda_i + \mu_j \le c_{ij} \qquad (i, j) \in F \times D \end{array}$$

I due problemi, nel caso $n = 2$ e $m = 3$, sono rappresentati in modo compatto nella seguente tabella, in cui il generico vincolo $\sum_{j\in D} x_{ij} = a_i$ è stato trasformato nel vincolo equivalente $-\sum_{j\in D} x_{ij} = -a_i$.

	x_{11}	x_{12}	x_{13}	x_{21}	x_{22}	x_{23}		max
λ_1	-1	-1	-1					$-a_1$
λ_2				-1	-1	-1		$-a_2$
μ_1	1			1			=	b_1
μ_2		1			1			b_2
μ_3			1			1		b_3
			$\|\wedge$					
min	c_{11}	c_{12}	c_{13}	c_{21}	c_{22}	c_{23}		

Ancora una volta, è facile verificare algebricamente che il costo di una qualsiasi soluzione ammissibile di uno dei due problemi fornisce una valutazione del valore ottimo dell'altro. Utilizzando i vincoli del problema di EIM, moltiplicati ciascuno per la corrispondente variabile x_{ij} del problema di BW e sommati, si ottiene

$$\sum_{(i,j)\in F\times D} c_{ij}x_{ij} \ge \sum_{(i,j)\in F\times D} (\mu_j - \lambda_i)x_{ij} \ .$$

Mettendo in evidenza μ_j e λ_i ed utilizzando i vincoli del problema di BW si ottiene

$$\sum_{(i,j)\in F\times D} c_{ij}x_{ij} \ge \sum_{j\in F} b_j\mu_j - \sum_{i\in D}^{n} a_i\lambda_i \ ,$$

ossia, ancora, il ricavo ottenibile per EIM da qualsiasi scelta dei prezzi di acquisto e vendita è sicuramente inferiore al costo di qualsiasi piano di trasporto ammissibile per BW.

Nei due esempi precedenti si è visto come ad un problema di *PL* è associabile un altro problema di *PL* avente con esso una stretta relazione. Generalizzando, dato un problema di *PL* nella forma

$$(P) \qquad \max\{\, cx \,:\, Ax \le b \,,\ x \ge 0 \,\} \tag{3.16}$$

possiamo sempre associare ad esso un problema così definito:

$$(D) \qquad \min\{\, yb \,:\, yA \ge c \,,\ y \ge 0 \,\} \ . \tag{3.17}$$

I problemi (3.16) e (3.17) costituiscono una *coppia di problemi duali*: spesso (3.16) viene chiamato il *primale* e (3.17) il *duale*, ma la nomencltura è arbitraria in quanto vale il seguente teorema.

Teorema 3.8. Il duale del duale è il primale.

Dimostrazione Utilizzando (3.2.*i*) ed (3.2.*iii*) il problema (3.17) può essere trasformato in

$$-\max\{\, -yb \,:\, -yA \le -c \,,\ y \ge 0 \,\} \ ;$$

secondo la definizione (3.16)–(3.17), il duale di questo problema è

$$-\min\{\, -cx \,:\, -Ax \ge -b \,,\ x \ge 0 \,\} \ ,$$

che, utilizzando di nuovo (3.2.*i*) ed (3.2.*iii*), può essere trasformato in (3.16). ◇

La coppia di problemi duali che abbiamo introdotto è detta *coppia simmetrica*. Una definizione equivalente di coppia di problemi duali si ha con la *coppia asimmetrica*

$$(P) \quad \max\{\, cx \,:\, Ax \le b \,\} \qquad\qquad (D) \quad \min\{\, yb \,:\, yA = c \,,\ y \ge 0 \,\} \ .$$

Infatti, (P) può essere scritto equivalentemente, utilizzando (3.3), come

$$\max\{\, cx^+ - cx^- \,:\, Ax^+ - Ax^- \le b \,,\ x^+ \ge 0 \,,\ x^- \ge 0 \,\} \ ;$$

applicando la definizione di coppia simmetrica si ottiene il duale

$$\min\{\, yb \,:\, yA \geq c\,,\ -yA \geq -c\,,\ y \geq 0\,\}$$

che, via (3.2.*ii*), è equivalente a (D).

Esercizio 3.4. Si dimostri il viceversa, cioè che partendo dalla definizione di coppia asimmetrica si ottiene la definizione di coppia simmetrica.

Esercizio 3.5. Si dimostri il Teorema 3.8 per la coppia asimmetrica.

In generale, il duale di un qualunque problema di PL può essere scritto applicando le corrispondenze (P)–(D) indicate nella seguente tabella, dove A_i e A^j indicano, rispettivamente, la i-esima riga e la j-esima colonna della matrice A.

max	c	b	$A_i x \leq b_i$	$A_i x \geq b_i$	$A_i x = b_i$	$x_j \geq 0$	$x_j \leq 0$	$x_j \gtrless 0$
min	b	c	$y_i \geq 0$	$y_i \leq 0$	$y_i \gtrless 0$	$yA^j \geq c_j$	$yA^j \leq c_j$	$yA^j = c_j$

(3.18)

Per dimostrare la tabella è sufficiente considerare il generico problema di PL scritto, senza perdita di generalità, come

$$\max\left\{ \begin{bmatrix} c^+ , c^- , c^0 \end{bmatrix} \begin{bmatrix} x^+ \\ x^- \\ x^0 \end{bmatrix} \,:\, \begin{bmatrix} A^+_+ & A^-_+ & A^0_+ \\ A^+_- & A^-_- & A^0_- \\ A^+_0 & A^-_0 & A^0_0 \end{bmatrix} \begin{bmatrix} x^+ \\ x^- \\ x^0 \end{bmatrix} \begin{matrix} \leq \\ \geq \\ = \end{matrix} \begin{bmatrix} b_+ \\ b_- \\ b_0 \end{bmatrix} \,,\quad \begin{matrix} x^+ \geq 0 \\ x^- \leq 0 \end{matrix} \right.$$

Utilizzando le trasformazioni equivalenti (3.2) e (3.3) il problema può essere portato nella forma

$$\max\left\{ \begin{bmatrix} c^+ , c^- , c^0 \end{bmatrix} \begin{bmatrix} x^+ \\ x^- \\ x^0 \end{bmatrix} \,:\, \begin{bmatrix} A^+_+ & A^-_+ & A^0_+ \\ -A^+_- & -A^-_- & -A^0_- \\ A^+_0 & A^-_0 & A^0_0 \\ -A^+_0 & -A^-_0 & -A^0_0 \\ -I & 0 & 0 \\ 0 & I & 0 \end{bmatrix} \begin{bmatrix} x^+ \\ x^- \\ x^0 \end{bmatrix} \leq \begin{bmatrix} b_+ \\ -b_- \\ b_0 \\ -b_0 \\ 0 \\ 0 \end{bmatrix} \right.$$

il cui duale è quindi

$$\min\left\{ \begin{bmatrix} y_+ , y_- , y^+_0 , y^-_0 , s^+ , s^- \end{bmatrix} \begin{bmatrix} b_+ \\ -b_- \\ b_0 \\ -b_0 \\ 0 \\ 0 \end{bmatrix} \,:\, \begin{bmatrix} y_+ , y_- , y^+_0 , y^-_0 , s^+ , s^- \end{bmatrix} \begin{bmatrix} A^+_+ & A^-_+ & A^0_+ \\ -A^+_- & -A^-_- & -A^0_- \\ A^+_0 & A^-_0 & A^0_0 \\ -A^+_0 & -A^-_0 & -A^0_0 \\ -I & 0 & 0 \\ 0 & I & 0 \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \\ \| \\ \begin{bmatrix} c^+ & c^- & c^0 \end{bmatrix} \end{matrix} \right.$$
$$\left. \begin{bmatrix} y_+ , y_- , y^+_0 , y^-_0 , s^+ , s^- \end{bmatrix} \geq 0 \right.$$

Utilizzando ancora le trasformazioni equivalenti, ed in particolare ridefinendo $y_- = -y_-$ (quindi $y_- \leq 0$), definendo $y_0 = y^+_0 - y^-_0$ (quindi $y_0 \gtrless 0$) ed eliminando le variabili di scarto s^+ ed s^-, il problema diviene

$$\min\left\{ \begin{bmatrix} y_+ , y_- , y_0 \end{bmatrix} \begin{bmatrix} b_+ \\ b_- \\ b_0 \end{bmatrix} \,:\, \begin{bmatrix} y_+ , y_- , y_0 \end{bmatrix} \begin{bmatrix} A^+_+ & A^-_+ & A^0_+ \\ A^+_- & A^-_- & A^0_- \\ A^+_0 & A^-_0 & A^0_0 \\ \vee| & \wedge| & \| \\ \begin{bmatrix} c^+ & c^- & c^0 \end{bmatrix} \end{bmatrix} \,,\quad \begin{matrix} y_+ \geq 0 \\ y_- \leq 0 \end{matrix} \right.$$

il che dimostra la validità della tabella.

Esempio 3.16. Applicazione delle corrispondenze (P)–(D)
Il duale del problema di minimo

$$\begin{array}{lrcrcrl} \min & 12x_1 & + & 7x_2 & & & \\ & 5x_1 & + & 7x_2 & = & 8 & \\ & 4x_1 & + & 2x_2 & \geq & 15 & \\ & 2x_1 & + & x_2 & \leq & 3 & \\ & x_1 & & & \geq & 0 & \end{array} \qquad \text{è} \qquad \begin{array}{lrcrcrcrl} \max & 8y_1 & + & 15y_2 & + & 3y_3 & & & \\ & 5y_1 & + & 4y_2 & + & 2y_3 & \leq & 12 & \\ & 7y_1 & + & 2y_2 & + & y_3 & = & 7 & . \\ & & & y_2 & & & \geq & 0 & \\ & & & & & y_3 & \leq & 0 & \end{array}$$

3.2.2 Il teorema debole della dualità

Come già evidenziato negli esempi 3.14 e 3.15, i problemi (P) e (D) non sono legati soltanto da relazioni di tipo sintattico: il seguente teorema fornisce una prima relazione tra i valori delle funzioni obiettivo dei due problemi. Qui, come nel seguito, salvo indicazione contraria useremo la forma asimmetrica della dualità, ma i risultati ottenuti sono indipendenti dalla particolare forma usata.

Teorema 3.9. (Teorema debole della dualità) Se $\bar{x}$ e $\bar{y}$ sono soluzioni ammissibili per (P) e (D) rispettivamente, allora $c\bar{x} \leq \bar{y}b$.

Dimostrazione
$$\left.\begin{array}{rcl} \bar{y}A = c & \Longrightarrow & \bar{y}A\bar{x} = c\bar{x} \\ A\bar{x} \leq b \ , \ \bar{y} \geq 0 & \Longrightarrow & \bar{y}A\bar{x} \leq \bar{y}b \end{array}\right\} \Longrightarrow \ c\bar{x} \leq \bar{y}b \ .$$
◇

La dimostrazione di questo teorema, per quanto elementare, merita di essere commentata, riprendendo i concetti già espressi negli esempi. L'idea fondamentale è che, poiché i vincoli $A_i x \leq b_i$ sono rispettati da tutti i punti della regione ammissibile, allora anche qualsiasi loro combinazione lineare a coefficienti non negativi

$$(\, yA \,)x = \left(\, \sum_{i=1}^{m} y_i A_i \,\right)x \leq \left(\, \sum_{i=1}^{m} y_i b_i \,\right) = yb$$

ha la stessa proprietà; si dice quindi che $(\, yA \,)x \leq yb$ è una *disuguaglianza valida* per il problema. Geometricamente, questo corrisponde ad un semispazio (affine) che contiene interamente il poliedro. Scegliendo le y in modo opportuno, ossia in modo tale che $yA = c$, l'iperpiano che definisce il vincolo ha lo stesso gradiente della funzione obiettivo; una disuguaglianza di tipo $cx \leq \gamma$ definisce una *curva di livello* della funzione obiettivo, ossia l'insieme di tutti i punti che hanno valore della funzione minore od uguale a γ. Determinare che una disuguaglianza di tipo $cx \leq \gamma$ è valida per il poliedro del problema (la sua regione ammissibile) dimostra chiaramente che nessun punto del poliedro può avere valore della funzione obiettivo maggiore di $\gamma (= yb)$, e quindi che questa è una corretta valutazione superiore del valore ottimo di (P).

Corollario 3.3. Se (P) è illimitato, allora (D) è vuoto.

In generale, se (P) e (D) sono non vuoti, si può quindi affermare che

$$\max\{\, cx \ : \ Ax \leq b \,\} \ \leq \ \min\{\, yb \ : \ yA = c \ , \ y \geq 0 \,\} \ .$$

Questa relazione vale in generale assumendo che il valore ottimo di un problema vuoto sia $-\infty$ se il problema è di massimo e $+\infty$ se il problema è di minimo. Una conseguenza apparentemente banale, ma in realtà cruciale, di questi risultati è:

Corollario 3.4. Se $\bar{x}$ e $\bar{y}$ sono soluzioni ammissibili rispettivamente per (P) e (D) e $c\bar{x} = \bar{y}b$, allora $\bar{x}$ e $\bar{y}$ sono anche soluzioni ottime.

Data una soluzione primale ammissibile $\bar{x}$, una soluzione duale ammissibile $\bar{y}$ tale che $c\bar{x} = \bar{y}b$ è detta un *certificato di ottimalità* di $\bar{x}$; si noti che, viceversa, $\bar{x}$ è un certificato di ottimalità di $\bar{y}$.

Esempio 3.17. Certificati di ottimalità per il problema della *Pintel*
Nell'Esempio 1.3 abbiamo già evidenziato con considerazioni puramente geometriche (cf. Figura 1.1) come il vertice $[\, 4 \,,\, 1 \,]$ sia la soluzione ottima: possiamo ora utilizzare il teorema debole della dualità per fornire una prova algebrica

della sua ottimalità. Per questo, scriviamo il problema ed il suo duale in forma di coppia asimmetrica:

$$\begin{array}{rrrrrl} \max & 500x_1 & + & 200x_2 & & \\ & x_1 & & & \leq & 4 \\ & & & x_2 & \leq & 7 \\ & 2x_1 & + & x_2 & \leq & 9 \\ & -x_1 & & & \leq & 0 \\ & & - & x_2 & \leq & 0 \end{array} \qquad \begin{array}{rrrrrrrrrrrrr} \min & 4y_1 & + & 7y_2 & + & 9y_3 & & & & & & \\ & y_1 & & & + & 2y_3 & - & y_4 & & & = & 500 \\ & & & y_2 & + & y_3 & & & - & y_5 & = & 200 \\ & y_1 & , & y_2 & , & y_3 & , & y_4 & , & y_5 & \geq & 0 \end{array}$$

È facile verificare che la soluzione duale $\bar{y} = [\,100\,,\,0\,,\,200\,,\,0\,,\,0\,]$ è ammissibile con $\bar{y}b = 2200$: poiché $\bar{x} = [\,4\,,\,1\,]$ è ammissibile per il problema primale ed ha anch'essa valore della funzione obiettivo $c\bar{x} = 2200$, il Corollario 3.4 garantisce l'ottimalità di entrambe le soluzioni (per il rispettivo problema). Si osservi che il caso in cui i valori delle funzioni obiettivo siano diversi tra loro non permette di certificare la *non* ottimalità delle soluzioni considerate. Ad esempio, avendo avuto $\bar{y}' = [\,0\,,\,100\,,\,200\,,\,0\,,\,0\,]$ con $\bar{y}' = 2500$ non avremmo potuto dichiarare che $\bar{x}$ non è ottima per il primale, così come $\bar{x}' = [\,1\,,\,7\,]$ con $c\bar{x}' = 1900$ non permette di dichiarare che $\bar{y}$ non è ottima per il duale.

3.2.3 Il teorema forte della dualità e sue conseguenze

Data la coppia asimmetrica di problemi duali (P) e (D), sia $\bar{x}$ una soluzione ammissibile per (P). Abbiamo già introdotto (cf. la Proprietà 3.1) il concetto di direzione ammissibile ξ per $\bar{x}$. Definiamo ξ una *direzione di crescita* per $\bar{x}$ se è possibile effettuare uno spostamento $\lambda > 0$ lungo ξ che migliori il valore della funzione obiettivo, cioè tale che

$$cx(\lambda) = c\bar{x} + \lambda c\xi > c\bar{x} \quad .$$

È facile verificare che questo non dipende dal particolare punto $\bar{x}$, e che vale

Proprietà 3.2. ξ è una direzione di crescita se e solo se $c\xi > 0$.

Si noti che se $c = 0$ non esistono direzioni di crescita; infatti, la funzione obiettivo vale sempre zero e tutte le soluzioni ammissibili sono quindi ottime. Quando invece $c \neq 0$, se esiste una direzione ammissibile per $\bar{x}$ che sia anche di crescita, allora $\bar{x}$ non può essere soluzione ottima di (P): infatti in tal caso sarebbe possibile effettuare un passo di spostamento $\lambda > 0$ lungo ξ che migliori il valore della funzione obiettivo. Quindi, la non esistenza di *direzioni ammissibili di crescita* per $\bar{x}$ è condizione necessaria affinché $\bar{x}$ sia ottima; si può dimostrare che la condizione è anche sufficiente.

Lemma 3.3. Una soluzione ammissibile $\bar{x}$ è ottima per (P) se e solo se non ammette direzioni ammissibili ξ che siano anche di crescita.

Dimostrazione Se $c = 0$ non esistono direzioni di crescita e ciascuna soluzione $\bar{x}$ è ammissibile, quindi il risultato vale; pertanto consideriamo il caso $c \neq 0$. Una delle implicazioni è già stata discussa; supponiamo ora per assurdo che $\bar{x}$ non sia ottima ma non esistano direzioni ammissibili di crescita per il punto. Deve esistere una soluzione ammissibile x' tale che $cx' > c\bar{x}$, cioè tale che $c\xi = c(\,x' - \bar{x}\,) > 0$: ξ è quindi una direzione di crescita, ma è anche ammissibile perchè la regione ammissibile del problema è convessa, il che fornisce una contraddizione. ◇

In particolare, nessun punto $\bar{x}$ interno al poliedro può essere una soluzione ottima per (P) (a meno che sia $c = 0$), in quanto $\xi = c$ costituisce una sua direzione ammissibile di crescita.

I risultati precedenti non usano, in realtà, il fatto che la regione ammissibile sia un poliedro, ma solo il fatto che è convessa; queste considerazioni possono quindi essere immediatamente estese ai problemi di ottimizzazione non lineare i cui vincoli definiscono una regione convessa.

Combinando le Proprietà 3.1 e 3.2 ed il Lemma 3.3 possiamo affermare che una soluzione ammissibile $\bar{x}$ è ottima per (P) se e solo se il sistema

$$\begin{cases} A_{I(\bar{x})}\xi & \leq 0 \\ c\xi & > 0 \end{cases} \tag{3.19}$$

non ammette soluzione. Questa caratterizzazione è di fondamentale importanza per stabilire un forte legame tra il problema primale ed il suo duale, come mostrato nel prossimo paragrafo.

3.2.3.1 Il Lemma di Farkas e la sua interpretazione geometrica

Determinare l'ottimalità di una soluzione ammissibile $\bar{x}$ è equivalente a determinare se il corrispondente sistema (3.19) ammette oppure no soluzioni. In altri termini, si deve stabilire se almeno uno degli

elementi del cono delle direzioni ammissibili $C(\bar{x})$ ha prodotto scalare strettamente positivo con c. Per questo si possono utilizzare gli strumenti introdotti nella §3.1.1.1. In particolare, se fosse disponibile una rappresentazione di $C(\bar{x})$ come cono finitamente generato, ossia $C(\bar{x}) = \{\xi = \nu V : \nu \geq 0\}$, il Lemma 3.1 permetterebbe immediatamente di stabilire se (3.19) abbia soluzione oppure no semplicemente calcolando il prodotto tra c e tutte le righe di V (i generatori di $C(\bar{x})$). Normalmente si ha peró a disposizione la matrice A ed il punto $\bar{x}$, e quindi la rappresentazione di $C(\bar{x})$ attraverso $A_{I(\bar{x})}$; come abbiamo visto, il numero di righe di V può essere esponenziale nel numero delle righe di tale matrice (ossia, $|I(\bar{x})|$), il che non rende praticabile l'approccio di costruire esplicitamente V per verificare se (3.19) ha soluzione. Si noti peró che questo problema non si pone in un caso rilevante: quello in cui $C(\bar{x})$ sia un cono simpliciale, ossia $\bar{x}$ sia una soluzione di base corrispondente ad una base *non degenere* B. In questo caso, infatti, (3.15) mostra che è possibile determinare se (3.19) ammette oppure no soluzioni in modo efficiente; come vedremo, questo è esattamente ciò che fanno gli algoritmi che presenteremo. In questo paragrafo vogliamo peró studiare in modo generale il problema di determinare se (3.19) ammette soluzioni. Consideriamo quindi per semplicità di notazione un generico cono (poliedrico) $C = (\xi : A\xi \leq 0)$ ed un vettore $c \in \mathbb{R}^n$, ed introduciamo il seguente Lemma, anche noto come *Teorema Fondamentale delle Disuguaglianze Lineari*.

Teorema 3.10. Data $A \in \mathbb{R}^{m\times n}$ e $c \in \mathbb{R}^n$, i due sistemi

$$(S_P) \quad \begin{cases} A\xi & \leq 0 \\ c\xi & > 0 \end{cases} \qquad\qquad (S_D) \quad \begin{cases} \nu A & = c \\ \nu & \geq 0 \end{cases}$$

sono mutuamente esclusivi, cioè o ha soluzione il sistema (S_P) oppure ha soluzione il sistema (S_D).

Forniamo adesso una breve dimostrazione algebrica utilizzando i concetti presentati nella §3.1.1.1. In particolare, dato C ricordiamo che il suo cono duale C^* ha la proprietà (3.10). Introduciamo quindi il *cono polare* di C come l'insieme di tutti i vettori che hanno quella proprietà, ossia

$$C^\circ = \{ d \in \mathbb{R}^n : d\xi \leq 0 \quad \forall \xi \in C \} \ .$$

È quindi ovvio che $C^* \subseteq C^\circ$, ma in realtà i due coni coincidono.

Lemma 3.4. $C^\circ = C^*$

Dimostrazione Una delle due inclusioni è già stata discussa, quindi occorre solo dimostrare $C^\circ \subseteq C^*$. Per questo basta richiamare il Lemma 3.2:

$$C^* = \{ d = \nu A : \nu \geq 0 \} = \{ d : Qd \leq 0 \}$$

per un'opportuna matrice Q tale che *tutte le righe di Q appartengono a C*. Si fissi dunque un qualsiasi $\bar{d} \in C^\circ$; per definizione, $\bar{d}\xi \leq 0$ per ogni $\xi \in C$, e poiché $Q_i \in C$ per ogni riga di Q ne consegue che $Q_i\bar{d} \leq 0$ per ogni i, ossia $Q\bar{d} \leq 0$, ossia $\bar{d} \in C^*$ come volevasi dimostrare. ◇

La dimostrazione del Teorema 3.10 è immediata dato il Lemma 3.4. Dato il vettore c, il teorema afferma che o $c \in C^*$, oppure esiste $\xi \in C$ tale che $c\xi > 0$. Che le due proprietà non possano verificarsi contemporaneamente discende direttamente da (3.10): $c \in C^*$ implica che $c\xi \leq 0$ per ogni $\xi \in C$, per cui se (S_D) ha soluzione allora (S_P) non può averne, e viceversa. Il punto critico è dimostrare che i due sistemi non possono essere entrambi vuoti, ma questo è un'ovvia conseguenza del Lemma 3.4. Infatti, ovviamente risulta che o $c \in C^*$, oppure no. Nel primo caso, evidentemente (S_D) ha soluzione. Se invece $c \notin C^* = C^\circ$, per definizione di cono polare si ha che deve esistere $\xi \in C$ tale che $c\xi > 0$, e quindi ha soluzione (S_P).

Si noti che è possibile (sotto opportune assunzioni) derivare una diversa dimostrazione da risultati relativi agli algoritmi che discuteremo nel seguito; si veda il Teorema 3.19.

Per interpretare geometricamente il Lemma di Farkas, basta notare che un vettore $\xi \in C$ tale che $c\xi > 0$ definisce un iperpiano che *separa* c da C^*: infatti c appartiene al semispazio (aperto) $\{ d : d\xi > 0 \}$, mentre $C^* \subseteq \{ d : d\xi \leq 0 \}$. Questa interpretazione geometrica è illustrata nella Figura 3.7, dove nel caso (a) si ha che $c \in C^*$, ossia c può essere espresso come combinazione lineare non negativa dei vettori A_i—in effetti utilizzando i soli due vettori A_2 ed A_3, per cui $c \in cono(\{ A_2 , A_3 \})$—mentre nel caso (b) è indicato il piano (ortogonale a $\bar{\xi}$) che separa c dal cono C^*. Il Lemma di Farkas è in effetti solo un esempio di una rilevante classe di teoremi, detti *teoremi di separazione*, che riguardano

il fatto che (sotto opportune ipotesi) dato un insieme convesso (in questo caso C^*) ed un punto che non gli appartiene (in questo caso c) esiste sempre un iperpiano che li separa.

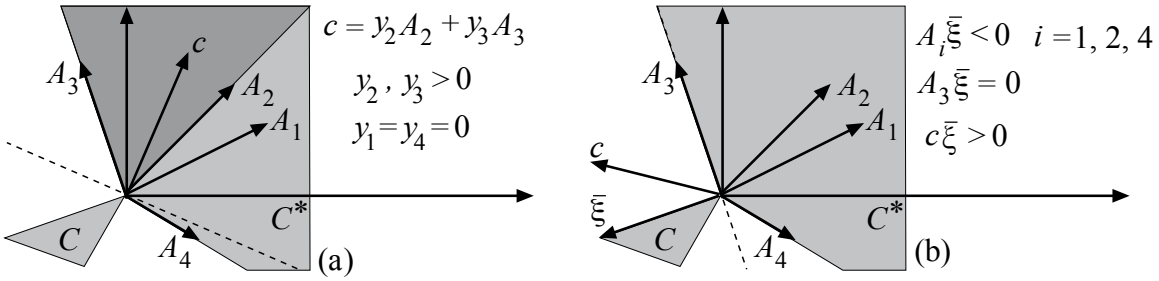


Figura 3.7: Interpretazione geometrica del Lemma di Farkas

3.2.3.2 Il teorema forte della dualità

Consideriamo il sistema che caratterizza le direzioni ammissibili di crescita per una soluzione ammissibile $\bar{x}$ e il sistema ad esso associato dal Lemma di Farkas. Per semplificare la notazione scriviamo $I = I(\bar{x})$: i due sistemi risultano essere

$$(P_R) \quad \begin{cases} A_I\xi & \leq 0 \\ c\xi & > 0 \end{cases} \qquad (D_R) \quad \begin{cases} y_IA_I & = c \\ y_I & \geq 0 \end{cases}$$

e verranno indicati, rispettivamente, come *Primale Ristretto* e *Duale Ristretto* (ai soli vincoli attivi). Si noti che per $c = 0$ il sistema (P_R) non può avere soluzioni; infatti, il sistema (D_R) ammette come ovvia soluzione $\bar{y}_I = 0$. Basandoci sul Lemma di Farkas, possiamo caratterizzare l'ottimalità di (P) in termini della risolubilità del sistema duale ristretto.

Corollario 3.5. Sia $\bar{x}$ una soluzione ammissibile per (P). Allora, $\bar{x}$ è una soluzione ottima per (P) se e solo se esiste una soluzione $\bar{y}_I$ del sistema (D_R).

Vale quindi il seguente fondamentale risultato

Teorema 3.11. (Teorema forte della dualità) Se (P) e (D) ammettono entrambi soluzioni ammissibili, allora

$$z(P) = \max\{\, cx \,:\, Ax \leq b \,\} \;=\; \min\{\, yb \,:\, yA = c \,,\; y \geq 0 \,\} = z(D) \;\; .$$

Dimostrazione Per il Teorema debole della dualità, poiché (D) ammette soluzioni ammissibili (P) non può essere illimitato; essendo non vuoto, (P) ha allora ottimo finito. Se $c = 0$, allora $z(P) = 0$ e $y = 0$, ammissibile per (D), è quindi ottima. Assumiamo perciò $c \neq 0$, e sia $\bar{x}$ una soluzione ottima per (P) con il relativo insieme di indici dei vincoli attivi I; come osservato in precedenza, $I \neq \emptyset$. Per il Corollario 3.5, il sistema (D_R) ammette almeno una soluzione $\bar{y}_I$. La soluzione $\bar{y} = [\,\bar{y}_I \,,\, 0\,]$ è ammissibile per (D), poiché $\bar{y}A = \bar{y}_IA_I = c$ e $\bar{y}_I \geq 0$ implica $\bar{y} \geq 0$; inoltre, risulta $\bar{y}b = \bar{y}_Ib_I = \bar{y}_IA_I\bar{x} = c\bar{x}$. Per il Teorema debole della dualità $\bar{y}$ è ottima per (D) e la tesi segue. ◇

Esempio 3.18. Applicazione del Teorema forte della dualità

Si consideri la seguente coppia di problemi duali:

$$(P) \quad \begin{array}{lrcrcl} \max & 2x_1 & + & x_2 & & \\ & x_1 & & & \leq & 5 \\ & & & x_2 & \leq & 5 \\ & x_1 & + & x_2 & \leq & 10 \\ & -x_1 & - & x_2 & \leq & -5 \end{array} \qquad (D) \quad \begin{array}{lrcrcrcl} \min & 5y_1 & + & 5y_2 & + & 10y_3 & - & 5y_4 \\ & y_1 & & & + & y_3 & - & y_4 = 2 \\ & & & y_2 & + & y_3 & - & y_4 = 1 \\ & y_1 & , & y_2 & , & y_3 & , & y_4 \geq 0 \end{array}$$

Sia $\bar{x} = [\,5\,,\,5\,]$ una soluzione ammissibile di (P), con $c\bar{x} = 15$. L'insieme degli indici dei vincoli attivi è $I = \{\,1\,,\,2\,,\,3\,\}$, a cui corrisponde la coppia di sistemi

$$(P_R) \quad \begin{array}{rcrcrcl} & & \xi_1 & & & \leq & 0 \\ & & & & \xi_2 & \leq & 0 \\ & & \xi_1 & + & \xi_2 & \leq & 0 \\ & & 2\xi_1 & + & \xi_2 & > & 0 \end{array} \qquad (D_R) \quad \begin{array}{rcrcrcl} y_1 & & & + & y_3 & = & 2 \\ & & y_2 & + & y_3 & = & 1 \\ y_1 & , & y_2 & , & y_3 & \geq & 0 \end{array}$$

Ogni direzione ammissibile e di crescita per $\bar{x}$ deve essere soluzione di (P_R), ma è facile verificare che tale sistema non ha soluzione: infatti, dovendo essere ξ_1 e ξ_2 entrambe non positive, non è possibile che $2\xi_1 + \xi_2$ risulti positivo. Per il Lemma di Farkas deve quindi avere soluzione il sistema (D_R), che ha infatti ne ha infinite; ponendo $y_3 = \alpha$ e sostituendo nelle equazioni si ottiene che $y_I(\alpha) = [\,2 - \alpha\,,\, 1 - \alpha\,,\, \alpha\,]$ soddisfa le equazioni del sistema duale per ogni valore di α. Imponendo anche le condizioni di non-negatività si ottiene che $y_I(\alpha)$ è una soluzione del sistema duale per ogni $\alpha \in [\,0\,,\,1\,]$. Fissando a zero tutte le y_i per $i \notin I$ si ottiene $y(\alpha) = [\,2 - \alpha\,,\, 1 - \alpha\,,\, \alpha\,,\, 0\,]$, che è ammissibile per (D) per ogni $\alpha \in [\,0\,,\,1\,]$. Quindi $\bar{x}$ è una soluzione ottima per (P) e $y(\alpha)$ è una soluzione ottima per (D) per ogni $\alpha \in [\,0\,,\,1\,]$: infatti, $c\bar{x} = y(\alpha)b = 15$.

Teorema 3.12. Se (P) ha ottimo finito, allora (D) ha ottimo finito.

Dimostrazione Poiché (P) è non vuoto, dal Teorema debole della dualità segue che (D) è limitato inferiormente, e quindi ha ottimo finito se e solo se è non vuoto. Per assurdo, supponiamo che il duale non abbia soluzione: per il Lemma di Farkas esiste quindi un vettore ξ tale che $A\xi \leq 0 \equiv \xi \in rec(\,P\,)$ e $c\xi > 0$; come abbiamo visto questo implica che (P) è superiormente illimitato in contraddizione con l'ipotesi. ◇

Esercizio 3.6. Dimostrare il teorema opposto: se (D) ha ottimo finito, allora (P) ha ottimo finito.

Teorema 3.13. Data la coppia di problemi duali (P) e (D), si può verificare uno solo tra i casi indicati con $*$ nella seguente tabella:

	(P)	ottimo finito	illimitato	vuoto
(D)	ottimo finito	$*$		
	illimitato			$*$
	vuoto		$*$	$*$

Dimostrazione Il seguente esempio mostra che (P) e (D) possono essere entrambi vuoti, come è immediato verificare:

$$(P)\ \max\{\, x_2 \,:\, -x_1 - x_2 \leq -1\,,\ x_1 + x_2 \leq -1\,\} \quad (D)\ \min\{\, -y_1 - y_2 \,:\, -y_1 + y_2 = 0\,,\ -y_1 + y_2 = 1\,,\ \ y_1, y_2 \geq 0\,\}\ .$$

Il completamento della dimostrazione è lasciato per esercizio. ◇

3.2.4 Il teorema degli scarti complementari

I teoremi debole e forte della dualità permettono di caratterizzare l'ottimalità di una coppia di soluzioni di (P) e (D). Più precisamente, il Corollario 3.4 e il Teorema 3.11 garantiscono che, date una soluzione $\bar{x}$ ammissibile per (P) e una soluzione $\bar{y}$ ammissibile per (D), queste sono ottime se e solo se i valori delle rispettive funzioni obiettivo concidono, ovvero $c\bar{x} = \bar{y}b$. Poiché la catena di equivalenze

$$c\bar{x} = \bar{y}b \quad \Longleftrightarrow \quad \bar{y}A\bar{x} = \bar{y}b \quad \Longleftrightarrow \quad \bar{y}(\,b - A\bar{x}\,) = 0$$

vale quando $\bar{y}A = c$ indipendentemente dall'eventuale ammissibilità delle soluzioni, la seguente definizione risulterà utile nel seguito.

Definizione 3.2. Le soluzioni $\bar{x} \in \mathbb{R}^n$ e $\bar{y} \in \mathbb{R}^m$ formano una *coppia di soluzioni complementari* se $\bar{y}A = c$ e viene verificata la seguente proprietà, detta degli *scarti complementari*:

$$\bar{y}(\,b - A\bar{x}\,) = 0\ . \tag{3.20}$$

Si noti che la definizione non richiede l'ammissibilità delle soluzioni; qualora siano entrambe ammissibili, ciò è sufficiente a garantirne l'ottimalità. Infatti, da quando detto sopra si deduce immediatamente il seguente teorema, noto come *Teorema degli scarti complementari*:

Teorema 3.14. Date due soluzioni $\bar{x}$ e $\bar{y}$, ammissibili rispettivamente per (P) e (D), esse sono ottime se e solo se verificano le condizioni degli scarti complementari (3.20).

Esplicitando il prodotto scalare, l'equazione (3.20) può essere riscritta

$$\bar{y}(\,b - A\bar{x}\,) = \textstyle\sum_{i=1}^{m} \bar{y}_i(\,b_i - A_i\bar{x}\,) = 0\ .$$

Quindi, una condizione *sufficiente* perché valga (3.20) è

$$\bar{y}_i(\,b_i - A_i\bar{x}\,) = 0 \qquad\qquad i = 1, \ldots, m \tag{3.21}$$

in quanto la somma di termini nulli da sicuramente zero; a sua volta, (3.21) è equivalente a

$$\begin{array}{rcl} \bar{y}_i > 0 & \Longrightarrow & A_i\bar{x} = b_i \\ A_i\bar{x} < b_i & \Longrightarrow & \bar{y}_i = 0 \end{array} \qquad i = 1, \ldots, m \ . \tag{3.22}$$

Si noti che (3.21) e (3.22) sono *equivalenti* a (3.20) quando $\bar{x}$ e $\bar{y}$ sono ammissibili rispettivamente per (P) e (D): infatti, l'ammissibilità implica che ciascun termine $\bar{y}_i(\ b_i - A_i\bar{x}\ )$ è non negativo, e la somma di m termini non negativi è zero se e solo se ciascun addendo è zero.

Per poter utilizzare il teorema degli scarti complementari direttamente, è necessario avere sia una soluzione primale sia una soluzione duale. Comunque, quando sia nota soltanto una soluzione primale ammissibile, è possibile verificare se sia ottima o meno tramite la ricerca di una soluzione duale ammissibile che formi con essa una coppia di soluzioni complementari. Infatti, vale la seguente caratterizzazione dell'ottimalità primale:

Proposizione 3.1. Sia $\bar{x}$ una soluzione ammissibile per (P). Allora, $\bar{x}$ è ottima se e solo se esiste $\bar{y}$ ammissibile per (D) complementare a $\bar{x}$.

Dimostrazione La parte "se" del teorema è un'immediata conseguenza del teorema degli scarti complementari. Consideriamo allora il caso in cui $\bar{x}$ sia una soluzione ottima per (P): per il Teorema 3.12 esiste una soluzione $\bar{y}$ ottima per (D) e per il Teorema 3.11 si ha $c\bar{x} = \bar{y}b$, ovvero vale (3.20). ◇

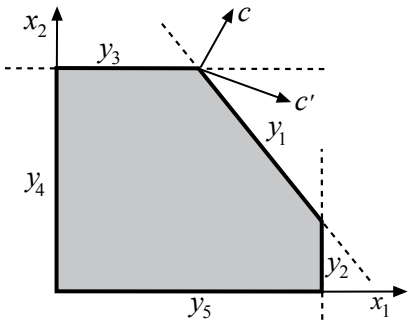
Una soluzione ammissibile $\bar{y}$ complementare a $\bar{x}$ soddisfa (3.22), e quindi deve valere $\bar{y}_i = 0$ per ogni $i \notin I$ (dove $I = I(\bar{x})$ indica l'insieme degli indici dei vincoli attivi). Poiché $\bar{y}A = c$, risulta anche $\bar{y}_I A_I = c$ e pertanto $\bar{y}_I$ è una soluzione del Duale Ristretto (D_R). Ritroviamo quindi che $\bar{x}$ è una soluzione ottima se e solo se esiste una soluzione $\bar{y}_I$ del Duale Ristretto (D_R), come già espresso dal Corollario 3.5. La condizione degli scarti complementari mostra quindi la strettissima relazione tra le soluzioni ottime del primale e quelle del duale: *una sola* soluzione ottime di uno qualsiasi dei due problemi consente di caratterizzare l'insieme di *tutte* le soluzioni ottime dell'altro. Mostriamo adesso alcuni modi in cui questa proprietà può essere sfruttata.

Esempio 3.19. Applicazione del Teorema degli scarti complementari
Si consideri la seguente coppia di problemi duali:

$$(P)\quad \begin{array}{lrcl} \max & x_1 + 2x_2 & & \\ & x_1 + x_2 & \leq & 5 \\ & x_1 & \leq & 4 \\ & x_2 & \leq & 3 \\ & -x_1 & \leq & 0 \\ & -x_2 & \leq & 0 \end{array} \qquad (D)\quad \begin{array}{lrcl} \min & 5y_1 + 4y_2 + 3y_3 & & \\ & y_1 + y_2 - y_4 & = & 1 \\ & y_1 + y_3 - y_5 & = & 2 \\ & y_1, y_2, y_3, y_4, y_5 & \geq & 0 \end{array}$$

La soluzione ottima di (P) è $\bar{x} = [\,2\,,\,3\,]$, di valore $c\bar{x} = 8$, come si può verificare dalla figura qui a fianco. Per il Teorema 3.14 si ha quindi che per *qualsiasi* soluzione ottima $\bar{y}$ di (D) deve valere

$$\begin{array}{rcl} \bar{x}_1 < 4 & \Longrightarrow & \bar{y}_2 = 0 \\ -\bar{x}_1 < 0 & \Longrightarrow & \bar{y}_4 = 0 \\ -\bar{x}_2 < 0 & \Longrightarrow & \bar{y}_5 = 0 \end{array} \ .$$



Quindi, le uniche componenti possibilmente non nulle di qualsiasi soluzione ottima $\bar{y}$ di (D) devono soddisfare il sistema

$$y_1 = 1 \quad , \quad y_1 + y_3 = 2 \ .$$

Tale sistema ammette un unica soluzione $[\,y_1\,,\,y_3\,] = [\,1\,,\,1\,]$, e pertanto esiste una sola candidata ad essere una soluzione duale ottima: $\bar{y} = [\,1\,,\,0\,,\,1\,,\,0\,,\,0\,]$. Poiché $\bar{y}$ è ammissibile ($\bar{y} \geq 0$), per il Teorema 3.14 essa è effettivamente ottima: infatti, anch'essa ha valore $\bar{y}b = 8$.

Se invece il vettore dei costi di (P) fosse stato $c' = \left[\ 3\,,\,-1\ \right]$, ripetendo gli stessi ragionamenti si sarebbe giunti a

$$y_1 = 3 \quad , \quad y_1 + y_3 = -1$$

che ha come unica soluzione $[\,y_1\,,\,y_3\,] = [\,3\,,\,-4\,]$; di conseguenza, l'unica soluzione che avrebbe rispettato gli scarti complementari con $\bar{x}$ sarebbe stata $\bar{y} = [\,3\,,\,0\,,\,-4\,,\,0\,,\,0\,] \not\geq 0$, quindi non ammissibile. Questo dimo-

stra *algebricamente* che $[\,2\,,\,3\,]$ non è ottimo per il problema con vettore dei costi c', come è facile verificare *geometricamente*.

Esempio 3.20. Scarti complementari parametrici primali

Utilizzando il Teorema degli scarti complementari, si vuole verificare per quali valori reali del parametro α la soluzione $\bar{x} = [\,3\,,\,1\,]$ è ottima per il problema di *PL* dato. Per ciascuno di tali valori si vuole inoltre individuare l'insieme delle soluzioni duali ottime.

$$\begin{array}{lrcrcl} \max & 4x_1 & + & \alpha x_2 & & \\ & 2x_1 & + & x_2 & \leq & 7 \\ & x_1 & - & x_2 & \leq & 3 \\ & x_1 & & & \leq & 3 \\ & 2x_1 & - & x_2 & \leq & 5 \end{array}$$

Scriviamo la coppia asimmetrica che comprende il problema in esame:

$$(P) \quad \begin{array}{lrcrcl} \max & 4x_1 & + & \alpha x_2 & & \\ & 2x_1 & + & x_2 & \leq & 7 \\ & x_1 & - & x_2 & \leq & 3 \\ & x_1 & & & \leq & 3 \\ & 2x_1 & - & x_2 & \leq & 5 \end{array} \qquad (D) \quad \begin{array}{lrcrcrcrcl} \min & 7y_1 & - & 3y_2 & + & 3y_3 & + & 5y_4 & & \\ & 2y_1 & + & y_2 & + & y_3 & + & 2y_4 & = & 4 \\ & y_1 & - & y_2 & & & - & y_4 & = & \alpha \\ & y_1 & , & y_2 & , & y_3 & , & y_4 & \geq & 0 \end{array}$$

La Proposizione 3.1 assicura che $\bar{x} = [\,3\,,\,1\,]$, se è ammissibile per (P), è ottima se e solo se esiste $\bar{y}$ ammissibile per (D) complementare a $\bar{x}$. È immediato verificare che $\bar{x}$ è ammissibile per (P) per qualsiasi valore di α (che, apparendo solo in funzione obiettivo, non influenza i vincoli e quindi l'ammissibilità). L'insieme degli indici dei vincoli attivi in $\bar{x}$ è $I(\bar{x}) = \{\, i \in \{1,\ldots,m\} \,:\, A_i\bar{x} = b_i \,\} = \{\,1\,,\,3\,,\,4\,\}$. Pertanto, una soluzione duale $\bar{y}$, tale che $\bar{y}A = c$, che formi con $\bar{x}$ una coppia di soluzioni complementari deve soddisfare la condizione $\bar{y}_2 = 0$. Affinché $\bar{y}$ sia ammissibile per (D), essa deve quindi soddisfare il sistema

$$\left\{ \begin{array}{rcrcrl} 2y_1 & + & y_3 & + & 2y_4 & = 4 \\ y_1 & & & - & y_4 & = \alpha \\ y_1 & , & y_3 & , & y_4 & \geq 0 \end{array} \right.$$

Per ogni fissato valore di α, tale sistema ammette infinite soluzioni della forma

$$[\; y_1\,,\, y_3\,,\, y_4 \;] = [\; 1+\alpha/2-y_3/4\,,\, y_3\,,\, 1-\alpha/2-y_3/4 \;] \;\; .$$

Pertanto, $\bar{x}$ è ottima per un dato valore di α se e solo se almeno una di tali soluzioni è ammissibile, ossia

$$1+\alpha/2-y_3/4 \geq 0 \;\;,\;\; y_3 \geq 0 \;\;,\;\; 1-\alpha/2-y_3/4 \geq 0 \;\;,$$

che equivale a

$$4+2\alpha \geq y_3 \;\;,\;\; y_3 \geq 0 \;\;,\;\; 4-2\alpha \geq y_3 \;\;.$$

Proiettando tale poliedro sullo spazio della sola variabile α attraverso l'eliminazione di Fourier-Motzkin si ottiene

$$-2 \leq \alpha \leq 2 \;\;;$$

pertanto, questi sono i valori di α per cui $\bar{x}$ è ottima per (P). Per ciascuno di questi valori, le soluzioni duali ottime hanno quindi la forma

$$\bar{y} = [\; 1+\alpha/2-y_3/4\,,\, 0\,,\, y_3\,,\, 1-\alpha/2-y_3/4 \;]$$

per tutti i valori di y_3 che rispettano

$$\min\{\, 4+2\alpha\,,\, 4-2\alpha \,\} \geq y_3 \geq 0 \;\;.$$

Si noti che per i valori "estremi" $\alpha = 2$ e $\alpha = -2$ la soluzione duale ottima è pertanto unica (e degenere, avendo tre componenti pari a zero), mentre per i valori intermedi esistono infinite soluzioni duali ottime.

Analogamente, è possibile verificare l'ottimalità di una soluzione duale tramite la ricerca di una soluzione primale ammissibile ad essa complementare.

Proposizione 3.2. Sia $\bar{y}$ una soluzione ammissibile per (D). Allora, $\bar{y}$ è ottima se e solo se esiste $\bar{x}$ ammissibile per (P) complementare a $\bar{y}$.

Esempio 3.21. Scarti complementari parametrici duali

Si consideri il problema di *PL* dato, parametrico in ε. Utilizzando il Teorema degli scarti complementari si vuole verificare per quali valori di ε la soluzione $\bar{y} = [\,1\,,\,0\,,\,0\,,\,1\,]$ è ottima per il duale del problema. Per tutti questi valori si vogliono caratterizzare tutte le soluzione ottime del primale.

$$\begin{array}{lrcrcrcl} \max & 3x_1 & & & & & & \\ & x_1 & - & 3x_2 & + & 3x_3 & \leq & 2-\varepsilon \\ & -x_1 & + & x_2 & + & x_3 & \leq & 0 \\ & -x_1 & - & x_2 & - & x_3 & \leq & 4 \\ & 2x_1 & + & 3x_2 & - & 3x_3 & \leq & 3+\varepsilon \end{array}$$

Il duale del problema in oggetto è:

$$\begin{array}{lrcrcrcrcl} \min & (2-\varepsilon)y_1 & & & + & 4y_3 & + & (3+\varepsilon)y_4 & & \\ & y_1 & - & y_2 & - & y_3 & + & 2y_4 & = & 3 \\ & -3y_1 & + & y_2 & - & y_3 & + & 3y_4 & = & 0 \\ & 3y_1 & + & y_2 & - & y_3 & - & 3y_4 & = & 0 \\ & y_1 & , & y_2 & , & y_3 & , & y_4 & \geq & 0 \end{array}$$

ed è immediato verificare che la soluzione $\bar{y} = [\,1\,,\,0\,,\,0\,,\,1\,]$ è ammissibile per qualsiasi valore di ε (che, apparendo solo nel lato destro dei vincoli del primale appare solamente nella funzione obiettivo del duale e quindi non ne influenza l'ammissibilità). Per la Proposizione 3.2, $\bar{y}$ è quindi ottima se e solo se esiste $\bar{x}$ ammissibile per (P) ed essa complementare. L'insieme degli indici delle variabili duali positive in $\bar{y}$ è $\{\,1\,,\,4\,\}$: di conseguenza, una soluzione primale $\bar{x}$ che formi con $\bar{y}$ una coppia di soluzioni complementari deve soddisfare la condizione $b_i - A_i\bar{x} = 0$ per $i \in \{\,1\,,\,4\,\}$, ovvero il primo ed il quarto vincolo devono essere attivi. Pertanto, $\bar{x}$ deve risolvere il sistema

$$\left\{ \begin{array}{rcrcrcl} x_1 & - & 3x_2 & + & 3x_3 & = & 2-\varepsilon \\ 2x_1 & + & 3x_2 & - & 3x_3 & = & 3+\varepsilon \end{array} \right. .$$

Posto $3(x_2 - x_3) = \alpha$, tale sistema può essere riscritto come

$$\left\{ \begin{array}{rcrcl} x_1 & - & \alpha & = & 2-\varepsilon \\ 2x_1 & + & \alpha & = & 3+\varepsilon \end{array} \right. ;$$

questo sistema ammette unica soluzione $[\,x_1\,,\,\alpha\,] = [\,5/3\,,\,\varepsilon - 1/3\,]$. In altri termini, il sistema originale ammette infinite soluzioni della forma $[\,x_1\,,\,x_2\,,\,x_3\,] = [\,5/3\,,\,x_3 + \varepsilon/3 - 1/9\,,\,x_3\,]$ per qualsiasi valore di x_3.

Dobbiamo adesso verificare se esistono valori di ε per cui queste soluzioni sono ammissibili. Il secondo vincolo impone che

$$-x_1 + x_2 + x_3 \leq 0 \implies -5/3 + x_3 + \varepsilon/3 - 1/9 + x_3 \leq 0 \implies \varepsilon \leq 16/3 - 6x_3 \equiv x_3 \leq 16/18 - \varepsilon/6 ,$$

mentre il terzo vincolo impone che

$$-x_1 - x_2 - x_3 \leq 4 \implies -5/3 - x_3 - \varepsilon/3 + 1/9 - x_3 \leq 4 \implies -50/3 - 6x_3 \leq \varepsilon \equiv -50/18 - \varepsilon/6 \leq x_3 .$$

Pertanto, comunque fissato ε, le soluzioni della forma

$$\left[\ x_1\,,\,x_2\,,\,x_3\ \right] = \left[\ 5/3\,,\,x_3 + \varepsilon/3 - 1/9\,,\,x_3\ \right] \quad \text{per} \quad -50/18 - \varepsilon/6 \leq x_3 \leq 16/18 - \varepsilon/6 ,$$

sono ammissibili per il primale e rispettano le condizioni degli scarti complementari con $\bar{y}$, e pertanto sono ottime. Poiché tale insieme di soluzioni è sempre non vuoto, $\bar{y}$ è ottima per il duale per qualsiasi valore di ε.

Tutti i risultati precedenti sono validi per la coppia asimmetrica di problemi duali; tuttavia, si possono definire opportunamente condizioni degli scarti complementari per coppie duali in qualsiasi forma. Ad esempio, il seguente corollario fornisce quelle per la coppia simmetrica di problemi duali.

Corollario 3.6. Siano $\bar{x}$ e $\bar{y}$ soluzioni ammissibili rispettivamente per (3.16) e (3.17), allora $\bar{x}$ e $\bar{y}$ sono soluzioni ottime se e solo se $\bar{y}(\ b - A\bar{x}\ ) = 0$ e $(\ \bar{y}A - c\ )\bar{x} = 0$.

Esercizio 3.7. Dimostrare il corollario 3.6.

Esempio 3.22. Scarti complementari simmetrici

Si consideri il problema di *PL* dato, in forma simmetrica. Utilizzando il Teorema degli Scarti Complementari si vuole determinare per quali valori di $\alpha \in \mathbb{R}$ la soluzione duale $\bar{y} = [\,0\,,\,0\,,\,1\,]$ à ottima. Per ciascuno di tali valori si vuole anche capire se $\bar{y}$ sia l'unica soluzione ottima e se il problema primale abbia soluzione ottima unica.

$$(P_\alpha) \qquad \begin{array}{lrcrcrcl} \max & x_1 & & & & & & \\ & -x_1 & + & x_2 & \leq & 1 \\ & 2x_1 & - & x_2 & \leq & 1 \\ & x_1 & + & x_2 & \leq & \alpha \\ & x_1 & , & x_2 & \geq & 0 \end{array}$$

Secondo la definizione di coppia simmetrica, il duale di (P_α) è

$$(D_\alpha) \qquad \begin{array}{lrcrcrcl} \min & y_1 & + & y_2 & + & \alpha y_3 & & \\ & -y_1 & + & 2y_2 & + & y_3 & \geq & 1 \\ & y_1 & - & y_2 & + & y_3 & \geq & 0 \\ & y_1 & , & y_2 & , & y_3 & \geq & 0 \end{array}$$

È immediato verificare che $\bar{y}$ è ammissibile per ogni valore di α. Possiamo quindi utilizzare il Corollario 3.6 per affermare che $\bar{y}$ è ottima se e solo se esiste una soluzione primale ammissibile che rispetta con essa le condizioni

degli scarti complementari (simmetrici), ossia

$$\begin{array}{l} y_4(b_3 - A_3x) = 0 \;\equiv\; 1(\,\alpha - x_1 - x_2\,) = 0 \;\equiv\; x_1 + x_2 = \alpha \\ x_2(\,yA^2 - c_2\,) = 0 \;\equiv\; x_2(\,1 - 0\,) = 0 \;\equiv\; x_2 = 0 \end{array} .$$

Le altre condizioni degli scarti complementari non impongono nulla: quelle duali $y_i(\,b_i - A_ix\,) = 0$ per $i = 1, 2$ hanno $y_i = 0$ e quindi sono verificate per qualsiasi valore di x, e quella primale $x_j(\,yA^j - c_j\,) = 0$ per $j = 1$ ha $yA^1 - c_1 = 1 - 1 = 0$ e quindi è verificata per qualsiasi valore di x_1.

La soluzione primale deve quindi avere la forma $\bar{x} = [\,\alpha\,,\,0\,]$, ed è quindi unica se è ammissibile, dimostrando in questo caso che $\bar{y}$ è ottima per quel valore di α. L'ammissibilità richiede quindi intanto $\alpha \geq 0$ per i vincoli di segno del primale. Inoltre deve risultare

$$\left\{ \begin{array}{rcrcl} -\alpha & + & 0 & \leq & 1 \\ 2\alpha & - & 0 & \leq & 1 \end{array} \right.$$

(il terzo vincolo non deve essere considerato in quanto ovviamente è rispettato per ciascuna soluzione della forma scelta). Ciò corrisponde a

$$\alpha \leq 1/2 \quad , \quad \alpha \geq -1$$

che considerato il vincolo precedentemente enunciato mostra finalmente che $\bar{y}$ è ottima per tutti i valori di $\alpha \in [\,0\,,\,1/2\,]$, a cui corrisponde l'unica soluzione ottima primale $\bar{x} = [\,\alpha\,,\,0\,]$.

Per tutti i valori $\alpha < 1/2$ tutti i vincoli del primale tranne il terzo sono soddisfatti come stretta disiguaglianza, il che implica $y_1 = y_2 = 0$ in qualsiasi soluzione ottima duale e quindi che $\bar{y}$ è unica. Per $\alpha = 1/2$, però, anche il secondo vincolo viene soddisfatto come uguaglianza da $\bar{x} = [\,\alpha\,,\,0\,]$, aprendo alla possibilità di soluzioni duali ottime multiple. Per le condizioni degli scarti complementari tali soluzioni devono rispettare all'uguaglianza il primo vincolo (dato che $x_1 > 0$) e comunque devono rispettare tutti gli altri, compresi quelli di segno, ossia

$$2y_2 + y_3 = 1 \;\;,\;\; -y_2 + y_3 \geq 0 \;\;,\;\; y_2 \geq 0 \;\;,\;\; y_3 \geq 0 \;\;.$$

Tale sistema ammette altre soluzioni oltre a $[\,y_2\,,\,y_3\,] = [\,0\,,\,1\,]$, ad esempio $[\,y_2\,,\,y_3\,] = [\,1/3\,,\,1/3\,]$; pertanto $[\,y_1\,,\,y_2\,,\,y_3\,] = [\,0\,,\,1/3\,,\,1/3\,]$ è una diversa soluzione duale ottima per $\alpha = 1/2$, e quindi $\bar{y}$ non è unica.

3.2.5 Interpretazione economica delle variabili duali

Ricordando i problemi di *product mix* del §1.2.1, è possibile interpretare un problema di *PL* nella forma (P) come quello di decidere il livello ottimo di n attività—le variabili x_j, corrispondenti alle colonne della matrice A—in modo da massimizzare il profitto ottenuto. Per realizzare le attività sono però necessarie m *risorse* disponibili in quantità limitata. In questa interpretazione, ciascun coefficiente a_{ij} della matrice A indica quante unità della risorsa i sono necessarie per effettuare un'unità dell'attività j; per questo A viene anche chiamata *matrice delle tecnologie*. Di conseguenza, il lato destro b_i dell'i-esimo vincolo—la i-esima riga della matrice A—rappresenta la quantità disponibile dell'i-esima risorsa. Un problema che si pone in molti contesti reali è quello di stimare il *valore* di ogni risorsa, definito come l'incremento di profitto che si potrebbe ottenere se si aumentasse la quantità disponibile b_i di questa risorsa. Questa informazione è ovviamente di grande utilità al momento in cui si consideri l'acquisizione di nuove risorse—o, viceversa, la dismissione di risorse attualmente disponibili e la corrispondente diminuzione di profitto—in quelli che sono tipicamente indicati come *problemi strategici* (relativi all'acquisizione ottima di risorse) per distinguerli da quelli *tattici* od *operativi* (relativi all'utilizzo ottimo delle risorse disponibili).

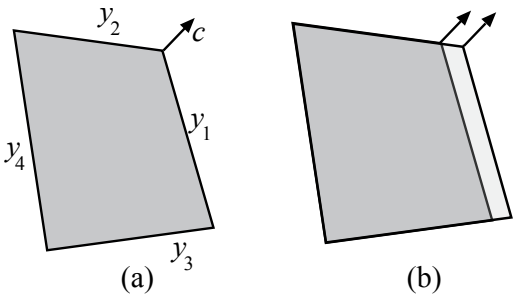


Figura 3.8: Costi ombra delle risorse

Matematicamente, sia $\bar{x}$ una soluzione ottima di (P) e $\bar{y}$ una soluzione ottima di (D). Consideriamo adesso una piccola variazione del vettore b, della forma $b(\varepsilon) = b + \varepsilon$, che lasci la soluzione ottima nell'intersezione degli iperpiani in cui era precedentemente. Un esempio è mostrato nella Figura 3.8: in (a) il poliedro originale, in (b) quello dopo la modifica (in particolare, la crescita di b_1 che causa la traslazione della frontiera del semispazio corrispondente al primo vincolo). Chiaramente, l'aver modificato i termini noti non ha alcun effetto sull'ammissibilità di $\bar{y}$; inoltre, indicando con $\bar{x}(\varepsilon)$ il nuovo valore assunto da $\bar{x}$, le condizioni del Teorema degli scarti complementari valgono anche per la coppia di soluzioni $\bar{x}(\varepsilon)$ e $\bar{y}$. In altre parole, $\bar{x}(\varepsilon)$ e $\bar{y}$ sono soluzioni ottime, e la variazione del problema primale non ha avuto effetto sulla soluzione ottima del duale. Tale variazione ha però effetto sul valore

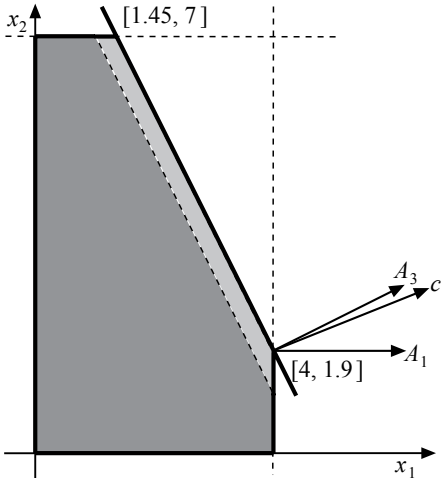
della funzione obiettivo, che diventa $c\bar{x}(\varepsilon) = \bar{y}(b + \varepsilon) = \bar{y}b + \bar{y}\varepsilon$. Pertanto, se ε è abbastanza piccolo per cui effettivamente $\bar{y}$ resta ottima (ammesso che ciò sia possibile, il che non è sempre vero; si veda il §3.3.4), il vettore $\bar{y}$ rappresenta il *gradiente* del valore ottimo della funzione obiettivo espresso in funzione della variazione ε di b, calcolato nell'origine ($\varepsilon = 0$). In altri termini, ciascuna componente $\bar{y}_i$ fornisce la variazione di valore ottimo della funzione obiettivo per una variazione unitaria della quantità b_i di risorsa disponibile, sotto l'ipotesi che una tale variazione non alteri la soluzione ottima del duale. Questo è il valore della i-esima risorsa, in quanto indica il massimo prezzo che è ragionevole pagare per disporre di un'unità aggiuntiva di tale risorsa: in questo senso si dice che i valori ottimi delle variabili duali forniscono i *valori marginali* (detti anche *prezzi o costi ombra*) delle risorse. I valori ottimi delle variabili duali forniscono quindi una valutazione del valore che hanno le diverse risorse in quanto utilizzate nel processo produttivo definito dal problema (P). Le condizioni degli scarti complementari implicano che la variabile duale corrispondente ad un vincolo soddisfatto all'ottimo come stretta disuguaglianza (nell'esempio di Figura 3.8(a), y_3 e y_4, corrispondenti alle risorse b_3 e b_4) sia necessariamente nulla. Questo porta alla cosiddetta *interpretazione economica degli scarti complementari*: una risorsa sovrabbondante ha valore nullo.

Esempio 3.23. Costi ombra per il problema della Pintel

Riprendiamo, per esemplificare i concetti esposti, il problema della Pintel: $\bar{x} = [\,4\,,\,1\,]$ e $\bar{y} = [\,100\,,\,0\,,\,200\,,\,0\,,\,0\,]$ sono rispettivamente soluzioni ottime del primale e del duale, con valore della funzione obiettivo pari a 2200. Chiediamoci adesso cosa accadrebbe se il reparto tecnico fosse in grado, noleggiando altri macchinari, di aumentare la produzione settimanale di wafers del 10%, portandola a 3300 unità: questo cambierebbe il terzo vincolo in

$$2x_1 + x_2 \leq 9.9$$

(la risorsa aumenta del 10%), lasciando tutti gli altri inalterati. Come si vede nella figura qui a fianco, questo corrisponde graficamente a traslare la faccia corrispondente al terzo vincolo nel verso di A_3, cosicché il vertice intersezione delle facce corrispondenti al primo ed al terzo vincolo si sposta nel punto $[\,4\,,\,1.9\,]$. Poiché non sono cambiati i vincoli attivi, né i vincoli del duale, $\bar{y}$ continua ad essere ammissibile per il duale. Essa inoltre soddisfa le condizioni degli scarti complementari con la nuova soluzione primale $[\,4\,,\,1.9\,]$: infatti, il vettore c è ancora esprimibile come combinazione non negativa dei gradienti del primo e del terzo vincolo, con gli stessi coefficienti. Pertanto, il vertice intersezione delle facce corrispondenti al primo ed al terzo vincolo continua ad essere ottimo: la posizione del vertice in $\mathbb{R}^2$ è però diversa, ed infatti cambia il valore della funzione obiettivo, che è ora 2380. Osserviamo che l'incremento è dato dal valore della variabile duale corrispondente al terzo vincolo ($\bar{y}_3 = 200$) per l'incremento della terza risorsa (0.9); quindi, è possibile stimare se il noleggio dei nuovi macchinari sia o no conveniente confrontando il costo di noleggio con il prezzo ombra del vincolo corrispondente ai wafer.



Lo stesso ragionamento può essere ripetuto per tutti gli altri vincoli del primale, anche quelli che non sembrano rappresentare risorse nel problema reale. Si consideri ad esempio il caso in cui il reparto marketing della Pintel fosse in grado, attraverso un'opportuna campagna pubblicitaria, di garantire la vendita di 100000 Pintium in più del previsto, aumentando la "prima risorsa" di un'unità; il valore marginale del secondo vincolo potrebbe essere usato per stimare la convenienza dell'operazione, noti i costi della campagna pubblicitaria. È però necessario rilevare che il corretto uso dei prezzi ombra richiede cautela: bisogna sempre verificare che la soluzione primale resti ammissibile, e quindi ottima, in corrispondenza al previsto aumento di risorse. Ad esempio, l'aumento di un'unità della "prima risorsa", trasformando il primo vincolo in $x_1 \leq 5$, porterebbe l'intersezione del primo e terzo vincolo nel punto $[\,5\,,\,-1\,]$, che non è ammissibile: è facile vedere che, per via del vincolo 3, il massimo incremento della prima risorsa che produce un effettivo miglioramento del valore della funzione obiettivo è pari a 0.5. È comunque possibile, sfruttando alcune informazioni fornite dagli algoritmi che vedremo nel prossimo paragrafo, stimare l'intervallo in cui la variazione di una risorsa porta ad un miglioramento della funzione obiettivo esattamente pari a quello previsto dal prezzo ombra (cf. §3.3.4).

L'interpretazione economica delle variabili duali ha molte utili applicazioni, ad esempio nella teoria economica dei mercati; ne discutiamo adesso un esempio rilevante.

Esempio 3.24. Il mercato elettrico

Il sistema elettrico presenta livelli di complessità che raramente sono percepiti dai non esperti. Questo è principalmente dovuto al fatto che l'energia elettrica è difficile da immagazzinare in grandi quantità, e quindi per la più

parte deve essere prodotta esattamente nello stesso momento in cui viene consumata. In molti paesi del mondo, tra cui l'Italia, per rendere possibile questa contemporaneità è istituito il *Mercato del Giorno Prima* (MGP), in cui produttori e consumatori di energia elettrica si accordano per assicurarsi che in ciascuna ora del giorno successivo siano disponibili impianti di generazione in grado di erogare sufficiente energia per le necessità degli utenti.

Il mercato viene eseguito separatamente per ciascuna ora, e quindi possiamo omettere l'indicazione del tempo. Per l'ora in questione, ciascun produttore di energia elettrica sottomette al *Gestore del Mercato Elettrico* (GME) *offerte di vendita* nella forma di una o più coppie (prezzo , quantità). Ad esempio, un'offerta di vendita del tipo $\{ (100, 200), (150, 250) \}$ indica che il produttore è disposto a vendere fino a 200 MWh di energia ad un prezzo non inferiore a 100 €/MWh, e successivamente fino ad ulteriori 250 MWh di energia ad un prezzo non inferiore a 150 €/MWh. Si noti che il prezzo è sempre assunto essere crescente: le offerte col prezzo più basso sono valide per le quantità iniziali di energia, mentre le quantità addizionali hanno prezzi man mano più alti. Simmetricamente, i consumatori di energia elettrica sottomettono al GME *offerte di acquisto* ancora nella forma di coppie (prezzo , quantità). In questo caso però la priorità è inversa: si assume che siano inizialmente valide le offerte col prezzo più alto, e man mano quelle con prezzi minori. Si noti che i consumatori (ma non i produttori) possono sottomettere offerte senza indicazione di prezzo, che significa che sono in principio disposti a pagare qualsiasi cifra pur di vedere soddisfatta la loro richiesta: si parla in questo caso di *domanda anelastica*, ed in effetti una frazione significativa della domanda in pratica è di questo tipo. Questo può essere facilmente incorporato nella trattazione, ma lo evitiamo per semplicità.

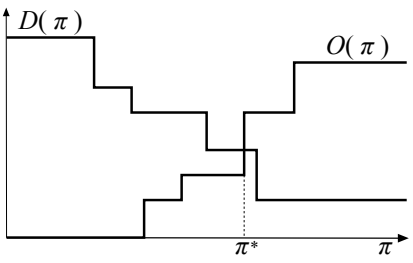
Matematicamente si ha quindi un insieme $\{ (sp_j, sq_j) : j \in S \}$ di offerte di vendita (S per "sell") ed un insieme $\{ (bp_i, bq_i) : i \in B \}$ di offerte di acquisto (S per "buy"). Ricevute le offerte il GME risolve il Problema Primale di Market Clearing

$$\begin{array}{lll} \text{(PPMC)} \quad \max & \sum_{i\in B} bp_i b_i \; - \; \sum_{j\in S} sp_j s_j & \\ & 0 \le b_i \le bq_i & i \in B \\ & 0 \le s_j \le sq_j & j \in S \\ & \sum_{i\in B} b_i \; = \; \sum_{j\in S} s_j & \end{array} \tag{3.23}$$

le cui variabili b_i ed s_j sono, rispettivamente, le quantità di energia accettate per l'offerta di acquisto i / vendita j. Come mostrato nel §3.3, risolvere (3.23) implica necessariamente risolvere il suo duale

$$\begin{array}{lll} \text{(PDMC)} \quad \min & \sum_{i\in B} bq_i \mu_i \; + \; \sum_{j\in S} sq_j \eta_j & \\ & \mu_i + \pi \ge bp_i \;\; , \;\; \mu_i \ge 0 & i \in B \\ & \eta_j - \pi \ge -sp_j \;\; , \;\; \eta_j \ge 0 & j \in S \end{array} \tag{3.24}$$

Questa coppia di problemi potrebbe a tutta prima apparire un pò misteriosa, ma l'analisi attraverso la teoria della dualità ne rivela facilmente il significato. Si considerino la *curva di offerta* $\mathcal{O}(\pi) = \sum_{j\,;\,sp_j \le \pi} sq_j$ e la *curva di domanda* $\mathcal{D}(\pi) = \sum_{i\,;\,bp_i \ge \pi} bq_i$. $\mathcal{O}(\pi)$ indica la massima quantità di energia che i produttori sono disponibili a vendere al prezzo π, ed è chiaramente non decrescente in π; simmetricamente, $\mathcal{D}(\pi)$ indica la massima quantità di energia che i consumatori sono disponibili a comprare al prezzo π, ed è chiaramente non crescente in π. Il prezzo di mercato dell'energia è il valore π^* in cui le due curve si incontrano, come illustrato nella figura qui accanto (assumiamo per semplicità che ciò accada, altrimenti il mercato "fallisce", ma ci sono modi per affrontare il problema che non è il caso di dettagliare in questa sede). Si noti che tutte le trensazioni di acquisto e vendita vengono valorizzate al prezzo di chiusura del mercato π^*, detto *market clearing price*. Infatti, questo tipo di mercato è detto *Price as Clear*, e per ragioni teoriche e pratiche è considerato il miglior paradigma di mercato per beni fungibili come l'energia, ancorché alcuni sviluppi recenti abbiano messo in discussione questo assunto.



Mostriamo adesso che le soluzioni ottime di (3.23) ed (3.24) implementano esattamente questa scelta. Questo discende dalle condizioni degli scarti complementari; per semplicità ci limitiamo a riportare quelle relative alle offerte di vendita, ossia

$$\eta_j(sq_j - s_j) = 0 \quad , \quad (\eta_j - \pi + sp_j)s_j = 0 \qquad j \in S \;\; .$$

Da queste segue che:

1. $\eta_j > 0$ implica che $s_j = sq_j$, ossia l'offerta j è integralmente accettata;
2. $s_j > 0 \Longrightarrow \pi - \eta_j = sp_j$, e poiché $\eta_j \ge 0$ si ha $sp_j \le \pi$: ossia, affinché una offerta possa essere anche solo parzialmente accettata il suo costo non puó essere superiore a quello di mercato;
3. infatti, riscrivendo il vincolo come $sp_j \ge \pi - \eta_j$ mostra che se $sp_j > \pi$, poiché $\eta_j \ge 0$ deve necessariamente risultare $sp_j > \pi - \eta_j$: il vincolo è rispettato come stretta disuguaglianza e quindi $s_j = 0$ per gli scarti complementari, ossia nessuna offerta il cui prezzo è superiore a π viene accettata.
4. per contro sia $sp_j < \pi$: perché il vincolo sia rispettato deve certamente essere $\eta_j > 0$, e per quanto detto sopra $s_j = sq_j$, ossia l'offerta viene integralmente accettata.

Per riassumere: tutte le offerte per cui $sp_j > \pi$ vengono integralmente rifiutate, mentre quelle per cui $sp_j < \pi$ vengono integralmente accettate. Se esistono offerte accettate parzialmente, ossia per cui $0 < s_j < sq_j$, allora

$s_j > 0$ implica $\pi - \eta_j = sp_j$ mentre $s_j < sq_j$ implica $\eta_j = 0$, e pertanto $\pi = sp_j$. Quelle sono le offerte "critiche" il cui prezzo determina quello di mercato. Un discorso completamente simmetrico vale per le offerte di acquisto: vengono integralmente rifiutate quelle con prezzo inferiore a quello del mercato ed integralmente accettate quelle con prezzo superiore. Le offerte "critiche" (siano esse di acquisto o vendita) sono "libere", nel senso che nella soluzione ottima le variabili corrispondenti possono avere qualsiasi valore permesso dai bounds: chiaramente esse saranno scelte in modo che il mercato sia "in equilibrio di quantità", ossia la produzione equivalga esattamente alla domanda, come richiesto dal vincolo di uguaglianza in (3.23) di cui π è la variabile duale. È facile verificare che questo corrisponde esattamente alla soluzione del problema di mercato con il metodo grafico precedentemente delineato.

Poiché la soluzione del mercato (il prezzo di chiusura π^* e quali offerte di acquisto e vendita siano accettate) può essere ottenuta facilmente col semplice metodo grafico delineato in precedenza, si potrebbe argomentare che la descrizione in termini di *LP* sia inutilmente barocca. Tale formalizzazione ha però interessanti risvolti teorici; ad esempio, interpretando i prezzi delle offerte di acquisto come il valore che la risorsa ha per i compratori e quelli di quelle di vendita come il costo che ha per i venditori, la funzione obiettivo di (3.23) (la cui forma è cruciale per il funzionamento dell'analisi) può essere vista come la massimizzazione del beneficio complessivo del mercato, inteso come il valore ottenuto al netto del costo incorso per ottenerlo.

Al di lá di argomentazioni teoriche, il modello di mercato (3.23)–(3.24) può essere facilmente esteso per includere *vincoli* sulle transazioni che il processo di soluzione grafico non può incorporare. Nel caso del mercato elettrico (come in altri) un vincolo cruciale riguarda la capacità di trasportare l'energia dal luogo di produzione a quello di consumo. Ciascun impianto di produzione (e quindi, ciascuna delle offerte di vendita) appartiene infatti as una delle *zone* in cui è diviso il sistema elettrico, e similmente per ciascun consumatore (offerte di acquisto). All'interno di ciascuna zona la capacità di trasporto è sufficientemente alta da permettere lo scambio di qualsiasi quantità di energia (con i limiti, noti, alla quantità di produzione ed assorbimento degli impianti); ma questo non vale per il trasporto tra zone diverse, che avviene attraverso elettrodotti (ad alta ed altissima tensione) che potrebbero non permettere di trasferire tutta l'energia che si vorrebbe. Si ha quindi un grafo non orientato $G = (\mathcal{K}, \mathcal{L})$ i cui vertici sono le zone ed i cui lati sono gli elettrodotti che le collegano. L'energia elettrica non si comporta però come un flusso al pari di quelli studiati nel §1.2.6 e nel Capitolo 2: in effetti le equazioni di conservazione di flusso (2.2) sono solo una delle due *Leggi di Kirchhoff* che governano il funzionamento dei circuiti elettrici. È però possibile costruire una ragionevole approssimazione lineare del funzionamento della rete elettrica attraverso i *coefficienti di sensitività* S_k^l che misurano il transito di energia sul collegamento $l \in \mathcal{L}$ rispetto all'iniezione netta (offerte di vendita accettate meno offerte di acquisto accettate) nella zona $k \in \mathcal{K}$; il calcolo di tali coefficienti non è rilevante per questa discussione. Dato G, i coefficienti S_k^l, gli insiemi $I(k)$ e $J(k)$ rispettivamente delle offerte di acquisto e offerte di vendita relativi alla zona $k \in \mathcal{K}$—e, viceversa, $k(i)$ e $k(j)$ rispettivamente la zona a cui l'offerta di acquisto i/vendita j fanno riferimento—ed m_l, M_l rispettivamente i minimi e massimi transiti ammissibili di energia attiva lungo il collegamento l, è possibile scrivere il Problema Primale di Market Clearing Vincolato

$$\begin{array}{lll} \text{(PPMCV)} \quad \max & \sum_{i\in B} bp_i b_i \; - \; \sum_{j\in S} sp_j s_j & \\ & 0 \le b_i \le bq_i & i \in B \\ & 0 \le s_j \le sq_j & j \in S \\ & \sum_{i\in B} b_i \; = \; \sum_{j\in S} s_j & \\ & m_l \le \sum_{k\in\mathcal{K}} S_l^k \left(\sum_{i\in I(k)} b_i - \sum_{j\in J(k)} s_j \right) \le M_l & l \in \mathcal{L} \end{array} \tag{3.25}$$

semplicemente aggiungendo a (3.24) vincoli che impongono che l'energia possa fluire sulla rete rispettando le capacità dei collegamenti. Il significato del duale

$$\begin{array}{lll} \min & \sum_{i\in B} bq_i \mu_i \; + \; \sum_{j\in S} sq_j \eta_j \; + \; \sum_{l\in\mathcal{L}} (M_l \lambda_l^+ - m_l \lambda_l^-) & \\ & \mu_i + \pi + \sum_{l\in\mathcal{L}} S_l^{k(i)} (\lambda_l^+ - \lambda_l^-) \ge bp_i \;\; , \;\; \mu_i \ge 0 & i \in B \\ & \eta_j - \pi - \sum_{l\in\mathcal{L}} S_l^{k(j)} (\lambda_l^+ - \lambda_l^-) \ge -sp_j \;\; , \;\; \eta_j \ge 0 & j \in S \\ & \lambda_l^+, \lambda_l^- \ge 0 & l \in \mathcal{L} \end{array}$$

diviene più chiaro riscrivendolo nella forma seguente:

$$\begin{array}{lll} \text{(PDMCV)} \quad \min & \sum_{i\in B} bq_i \mu_i \; + \; \sum_{j\in S} sq_j \eta_j \; + \; \sum_{l\in\mathcal{L}} (M_l \lambda_l^+ - m_l \lambda_l^-) & \\ & \pi^k = \pi + \sum_{l\in\mathcal{L}} S_l^k (\lambda_l^+ - \lambda_l^-) & k \in \mathcal{K} \\ & \mu_i + \pi^{k(i)} \ge bp_i \;\; , \;\; \mu_i \ge 0 & i \in B \\ & \eta_j - \pi^{k(j)} \ge -sp_j \;\; , \;\; \eta_j \ge 0 & j \in S \\ & \lambda_l^+, \lambda_l^- \ge 0 & l \in \mathcal{L} \end{array} \tag{3.26}$$

Questo modello definisce *prezzi zonali* π^k che determinano l'accettazione delle offerte di vendita ed acquisto in modo diversificato per ciascuna zona. Il valore ottimo di π^k sarà quindi il corrispettivo al quale sarà valorizzata l'energia venduta nella zona $k \in \mathcal{K}$, ed all'interno di essa vale l'analisi vista in precedenza rispetto all'accettazione (parziale o totale) oppure no delle singole offerte. Il prezzo diversificato per zona corrisponde al fatto che, ad esempio, in alcune di esse la produzione può essere di molto superiore alla domanda, ma non tutto il surplus può

essere trasportato in zone in cui accade l'inverso per via della capacità della rete di trasmissione; pertanto il valore dell'energia in quell'area è più basso che in altre e possono essere localmente accettate anche offerte di acquisto che non lo sarebbero altrove. Non è possibile in questa sede discutere tutti i dettagli di questo interessante modello, ma ne accenniamo almeno uno rilevante: la funzione obiettivo di (3.26) implica che non tutto il ricavato dalle offerte di acquisto viene ottenuto dai venditori. Infatti, una parte è il *costo di congestione* $\sum_{l\in\mathcal{L}}(M_l\lambda_l^+ - m_l\lambda_l^-)$ relativo ai vincoli di capacità della rete, le cui variabili duali λ_l^+ e λ_l^- misurano l'impatto economico che tali vincoli hanno sul mercato (di quanto potrebbe aumentare il beneficio complessivo del mercato se le capacità dei corrispondenti collegamenti di rete aumentassero). Tali variabili sono quindi utilizzate per incentivare il gestore della rete elettrica (tipicamente un monopolio naturale) ad incrementare tali capacità.

Esistono molte altre varianti di problemi di mercato che tengono in considerazione altri vincoli e meccanismi; ad esempio, nel marcato elettrico italiano i produttori hanno effettivamente prezzi diversificati per zona, ma i consumatori pagano l'energia allo stesso prezzo su tutto il territorio nazionale. Queste richieste tipicamente rendono i problemi di mercato più complessi, ma la loro analisi parte comunque sempre dai modelli qui illustrati. Al di lá dei dettagli, nei quali non è possibile entrare in queste dispense, l'esempio mostra quindi come le variabili duali di un problema di *PL* possano essere cruciali per determinare i prezzi delle risorse in qualche modo associate ai vincoli del problema.

Esercizio 3.8. Si modifichino i modelli appena visti per tener conto della possibilità che le offerte di vendita siano senza indicazione di prezzo e si discuta l'impatto che questo ha sull'analisi mostrata.

3.2.6 Soluzioni complementari e basi

Data la coppia asimmetrica di problemi duali, abbiamo già visto che ad una *base* B vengono associati la *matrice di base* A_B ed il punto $\bar{x} = A_B^{-1}b_B \in \mathbb{R}^n$, che è detto *soluzione primale di base*; nel caso sia ammissibile, $\bar{x}$ corrisponde ad un vertice del poliedro che individua la regione ammissibile di (P). Vogliamo costruire una soluzione duale che formi con $\bar{x}$ una coppia di soluzioni complementari: considerando anche l'insieme degli indici non in base $N = \{1, \ldots, m\} \setminus B$, possiamo associare alla base B la *soluzione duale di base*

$$\bar{y} = \left[\ \bar{y}_B\,,\, \bar{y}_N\ \right] = \left[\ cA_B^{-1}\,,\, 0\ \right]\ .$$

Tale soluzione soddisfa per costruzione la proprietà $\bar{y}A = c$ richiesta dalla Definizione 3.2; inoltre, segue facilmente da (3.22) che $\bar{y}$ è complementare a $\bar{x}$. Infatti, per $i \in B$ si ha $A_i\bar{x} = b_i$ (e $\bar{y}_i$ può essere diverso da zero), mentre per $i \in N$ si ha $\bar{y}_i = 0$ (e $A_i\bar{x}$ può essere diverso da b_i). Da ciò segue:

Proposizione 3.3. La coppia di soluzioni associate ad una base B soddisfa le condizioni degli scarti complementari (3.20).

Le soluzioni $\bar{x}$ e $\bar{y}$ associate ad una base vengono pertanto dette *soluzioni di base complementari*; qualora siano entrambe ammissibili, il teorema degli scarti complementari garantisce la loro ottimalità. In particolare, $\bar{x}$ e $\bar{y}$ risultano essere soluzioni di base *ammissibili*, *non ammissibili*, *degeneri* o *non degeneri* se verificano le condizioni riportate nella seguente tabella:

	$\bar{x}$	$\bar{y}$
ammissibile	$A_N\bar{x} \le b_N$	$\bar{y}_B \ge 0$
non ammissibile	$\exists\, i \in N\ :\ A_i\bar{x} > b_i$	$\exists\, i \in B\ :\ \bar{y}_i < 0$
degenere	$\exists\, i \in N\ :\ A_i\bar{x} = b_i$	$\exists\, i \in B\ :\ \bar{y}_i = 0$
non degenere	$\forall\, i \in N\ :\ A_i\bar{x} \ne b_i$	$\forall\, i \in B\ :\ \bar{y}_i \ne 0$

Una base B è detta primale o duale ammissibile, non ammissibile, degenere o non degenere a seconda che lo sia la soluzione di base primale o duale ad essa associata. Si può dimostrare che anche $\bar{y}$ è un vertice della regione ammissibile di (D) nel caso in cui sia ammissibile.

Esempio 3.25. Soluzioni di base complementari
Si consideri la coppia di problemi duali rappresentata nella tabella e nel grafico di Figura 3.9.

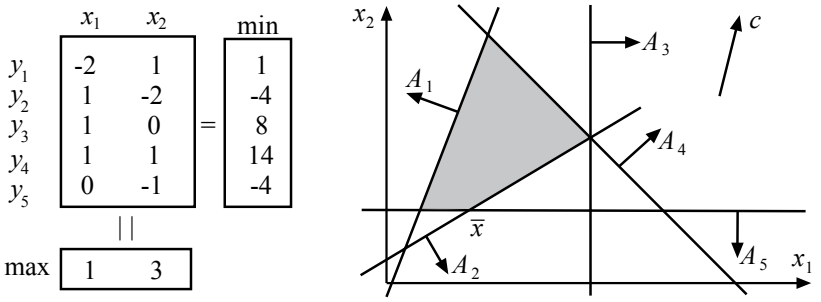


Figura 3.9: Dati per l'Esempio 3.25

Per $B = \{ 2 , 5 \}$ si ha

$$A_B = A_B^{-1} = \begin{bmatrix} 1 & -2 \\ 0 & -1 \end{bmatrix} \quad \Longrightarrow \quad \bar{x} = A_B^{-1} b_B = \begin{bmatrix} 1 & -2 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} -4 \\ -4 \end{bmatrix} = \begin{bmatrix} 4 \\ 4 \end{bmatrix} .$$

Si ha quindi $A_1\bar{x} = -4 < 1$, $A_3\bar{x} = 4 < 8$ e $A_4\bar{x} = 8 < 14$, cioè $\bar{x}$ è una soluzione di base primale ammissibile; la corrispondente $\bar{y}$ è data da $\bar{y} = [\, \bar{y}_B \,, 0 \,, 0 \,]$ dove $\bar{y}_B = cA_B^{-1} = [\, 1 \,, -5 \,]$ e quindi non è ammissibile ($\bar{y}_5 = -5 < 0$). Si noti che $\bar{x}$ e $\bar{y}$ sono soluzioni non degeneri.

Consideriamo adesso il problema di stabilire se una data soluzione primale $\bar{x}$ sia una soluzione di base: dato l'insieme I degli indici dei vincoli attivi, si ha che $\bar{x}$ è una soluzione di base se e solo se $rango(\, A_I \,) = rango(\, A \,) = n$. Infatti, $rango(\, A_I \,) = n$ implica che esista $B \subseteq I$ tale che $|\, B \,| = n$ e $det(\, A_B \,) \neq 0$, e quindi $\bar{x} = A_B^{-1} b_B$. Se $|\, I \,| = n$, allora $B = I$ e $\bar{x}$ è una soluzione di base *non degenere*. In questo caso, esiste un solo vettore $\bar{y}$ tale che $(\, \bar{x} \,, \bar{y} \,)$ sia una coppia di soluzioni di base complementari; infatti la matrice di base A_B associata a $\bar{x}$ è univocamente determinata, e di conseguenza è univocamente determinato il vettore $\bar{y}$. Viceversa, se $|\, I \,| > n$, allora $\bar{x}$ è una soluzione *degenere*: in questo caso più matrici di base possono corrispondere a $\bar{x}$, e conseguentemente più soluzioni di base di (D) possono costituire con $\bar{x}$ una coppia di soluzioni complementari.

Esempio 3.26. Soluzioni primali degeneri
Nel problema dell'Esempio 3.25 si consideri $\bar{x} = [\, 8 \,, 6 \,]$ con $c\bar{x} = 26$: si ha

$$\begin{array}{rcrcr} A_1\bar{x} & = & -10 & < & 1 \\ A_2\bar{x} & = & -4 & = & -4 \\ A_3\bar{x} & = & 8 & = & 8 \\ A_4\bar{x} & = & 14 & = & 14 \\ A_5\bar{x} & = & -6 & < & -4 \end{array}$$

e quindi $\bar{x}$ è ammissibile e soddisfa come equazione i vincoli 2, 3 e 4, cioè $I = \{ 2 \,, 3 \,, 4 \}$. $\bar{x}$ è quindi una soluzione di base degenere, cui corrispondono le basi $B' = \{ 2 \,, 3 \}$, $B'' = \{ 2 \,, 4 \}$ e $B''' = \{ 3 \,, 4 \}$, come è facilmente possibile verificare. Le soluzioni duali complementari corrispondenti a $\bar{x}$ sono

$$\bar{y}' = [\ 0 \,, -3/2 \,, 5/2 \,, 0 \,, 0\ ] \qquad , \qquad \bar{y}'' = [\ 0 \,, -2/3 \,, 0 \,, 5/3 \,, 0\ ] \qquad , \qquad \bar{y}''' = [\ 0 \,, 0 \,, -2 \,, 3 \,, 0\ ]$$

nessuna delle quali è ammissibile per (D); si ha comunque $c\bar{x} = \bar{y}'b = \bar{y}''b = \bar{y}'''b = 26$.

Consideriamo adesso il problema di stabilire se una data soluzione duale $\bar{y}$, tale che $\bar{y}A = c$, sia una soluzione di base: dato $J = J(\, \bar{y} \,) = \{\ j \ : \ \bar{y}_j \neq 0 \ \}$, si ha che $\bar{y}$ è una soluzione di base se e solo se tutte le righe di A_J sono linearmente indipendenti. Infatti, se $|\, J \,| = n$ allora la matrice di base corrispondente a $\bar{y}$ è $A_B = A_J$. In questo caso esiste un solo vettore $\bar{x}$ tale $(\, \bar{x} \,, \bar{y} \,)$ sia una coppia di soluzioni di base complementari; infatti la matrice di base A_B associata a $\bar{y}$ è univocamente determinata e di conseguenza è univocamente determinato il vettore $\bar{x}$. Se $|\, J \,| < n$, allora $\bar{y}$ è una soluzione degenere: a tale soluzione corrispondono più matrici di base, ottenute aggiungendo $n - |\, J \,|$ righe di A alla matrice A_J in modo che la matrice A_B risultante abbia determinante non nullo (tali righe esistono per l'ipotesi che $rango(\, A \,) = n$). Conseguentemente, più soluzioni di base $\bar{x}$ di (P) possono costituire con $\bar{y}$ una coppia di soluzioni complementari.

Esempio 3.27. Soluzioni duali degeneri
Si consideri ancora il problema dell'Esempio 3.25 con la diversa funzione obiettivo $c = [\, 1 \,, 1 \,]$, e sia $\bar{y} = [\, 0 \,, 0 \,, 0 \,, 1 \,, 0 \,]$, cosicché $\bar{y}b = 14$. $\bar{y}$ è una soluzione ammissibile di base degenere per (D): infatti risulta $J = \{ 4 \}$. Si può facil-

mente verificare che le basi per $\bar{y}$ sono $B' = \{1, 4\}$, $B'' = \{2, 4\}$, $B''' = \{3, 4\}$ e $B'''' = \{5, 4\}$, infatti risulta $cA_{B'}^{-1} = cA_{B''}^{-1} = cA_{B'''}^{-1} = cA_{B''''}^{-1} = [0, 1]$. Le soluzioni primali di base complementari a $\bar{y}$ sono riportate nella seguente tabella:

$\bar{x}' = [13/3\ , 29/3]$	$\bar{x}'' = \bar{x}''' = [8, 6]$	$\bar{x}'''' = [10\ , 4]$
(ammissibile)	(ammissibile degenere)	(non ammissibile)

Poiché $\bar{x}'$ e $\bar{x}'' = \bar{x}'''$ sono ammissibili, allora $\bar{y}$ è ottima: infatti $c\bar{x}' = c\bar{x}'' = c\bar{x}''' = 14 = \bar{y}b$. L'esempio mostra quindi come in presenza di degenerazione della soluzione duale ottima è quindi possibile avere soluzioni primali ottime diverse.

In assenza di degenerazione, ad una soluzione di base è pertanto associata una sola base ed una sola soluzione di base complementare. In questo caso è possibile verificare se la soluzione sia ottima o meno tramite l'ammissibilità dell'unica soluzione complementare.

Teorema 3.15. Siano $\bar{x}$ una soluzione di base primale ammissibile non degenere e B la corrispondente base. Allora, $\bar{x}$ è ottima se e solo se la soluzione duale complementare $\bar{y} = [\, cA_B^{-1}\, , 0\,]$ è ammissibile.

Dimostrazione La parte "se" del teorema segue immediatamente dalla Proposizione 3.3 e dal teorema degli scarti complementari. Consideriamo allora il caso in cui $\bar{x}$ sia una soluzione ottima. Per la Proposizione 3.1 esiste una soluzione duale ammissibile $\bar{y}$ complementare a $\bar{x}$ e quindi per cui valgono le condizioni degli scarti complementari (3.20). Poiché $\bar{x}$ è non degenere e quindi $I = B$, risulta $\bar{y}_{\bar{I}} = \bar{y}_N = 0$. Dall'ammissibilità di $\bar{y}$ segue che $\bar{y}_B A_B = c$, ovvero $\bar{y}_B = cA_B^{-1}$. Quindi, $\bar{y}$ è la soluzione duale di base associata a B. ◇

È possibile dimostrare l'analoga caratterizzazione dell'ottimalità di una soluzione di base duale non degenere.

Teorema 3.16. Siano $\bar{y}$ una soluzione di base duale ammissibile non degenere e B la corrispondente base. Allora, $\bar{y}$ è ottima se e solo se la soluzione primale complementare $\bar{x} = A_B^{-1} b_B$ è ammissibile.

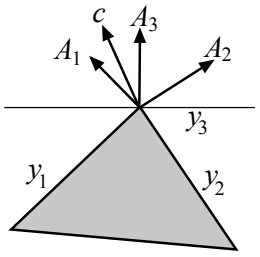
Nel caso di soluzioni di base degeneri, invece, l'ammissibilità della soluzione di base complementare fornisce soltanto una condizione sufficiente di ottimalità, come mostra il seguente esempio.

Esempio 3.28. Condizioni di ottimo in presenza di degenerazione

Si considerino, per il problema in figura qui accanto, la soluzione di base $\bar{x}$ tale che $I = \{1, 2, 3\}$ e le basi $B' = \{1, 2\}$, $B'' = \{1, 3\}$ e $B''' = \{2, 3\}$. Si ha che:

$$c \in cono(\{A_1, A_2\}) \Longrightarrow \bar{y}_{B'} \geq 0$$
$$c \in cono(\{A_1, A_3\}) \Longrightarrow \bar{y}_{B''} \geq 0 \quad ;$$
$$c \notin cono(\{A_2, A_3\}) \Longrightarrow \bar{y}_{B'''} \ngeq 0$$

quindi, $A_{B'''}$ non soddisfa la condizione di ottimo, pur essendo $\bar{x}$ una soluzione ottima.



In presenza di degenerazione, ad una soluzione di base corrispondono più soluzioni complementari, una per ciascuna base. Si possono però dimostrare gli analoghi dei Teoremi 3.15 e 3.16:

Teorema 3.17. Sia $\bar{x}$ una soluzione di base primale ammissibile. Allora, $\bar{x}$ è ottima se e solo se esiste una base B associata a $\bar{x}$ tale che la soluzione duale complementare $\bar{y} = [\, cA_B^{-1}\, , 0\,]$ sia ammissibile.

Teorema 3.18. Sia $\bar{y}$ una soluzione di base duale ammissibile. Allora, $\bar{y}$ è ottima se e solo se esiste una base B associata a $\bar{y}$ tale che la soluzione primale complementare $\bar{x} = A_B^{-1} b_B$ sia ammissibile.

Evitiamo di discutere le dimostrazioni perché possono essere ricavate da risultati relativi agli algoritmi che presenteremo nel seguito.

Esercizio 3.9. Costruire esempi di problemi (P), (D) rappresentabili in $\mathbb{R}^2$ aventi soluzione ottima degenere, individuando una matrice di base che non soddisfi le condizioni di ottimo.

Esercizio 3.10. Costruire, se esiste, una coppia di problemi (P), (D) tale che le soluzioni ottime $\bar{x}$ e $\bar{y}$ siano entrambe degeneri.

Esercizio 3.11. Dimostrare il seguente teorema: Se (P) $[(D)]$ ammette più soluzioni ottime di base, allora la soluzione ottima di base di (D) $[(P)]$ è degenere.

3.3 Algoritmi del Simplesso

La teoria della dualità sviluppata nel paragrafo precedente fornisce gli strumenti per la costruzione di un algoritmo per la soluzione dei problemi di *PL*, che chiameremo genericamente *algoritmo del simplesso*. Questo algoritmo costituisce il primo approccio computazionalmente efficiente per la soluzione di problemi di *PL*; originalmente proposto da G.B. Dantzig [1951] a partire da un idea di J. Von Neumman, il metodo del simplesso è stato sviluppato in diverse versioni e sta alla base dei più diffusi codici di *PL*. Quelle che presenteremo sono solamente alcune delle diverse varianti del metodo sviluppate a partire dall'algoritmo originale di Dantzig. In particolare, vedremo inizialmente la versione "naturale" per problemi nella forma (P) della coppia asimmetrica, che chiameremo di conseguenza *algoritmo del simplesso primale*. Poiché la teoria del paragrafo precedente indica chiaramente che risolvere (P) risolve anche (D) e viceversa, presenteremo poi una versione dell'algoritmo che "risolve (D) per risolvere (P)", e che quindi chiameremo *algoritmo del simplesso duale*. Vedremo però che i due sono nei fatti sostanzialmente lo stesso algoritmo.

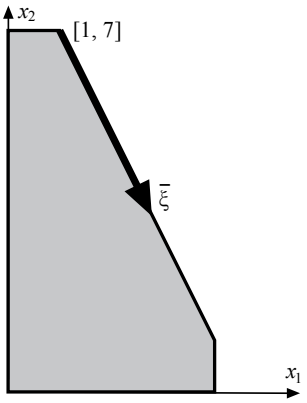
3.3.1 L'algoritmo del Simplesso Primale

L'algoritmo del simplesso considera primale l'insieme dei vertici del problema primale e, ad ogni passo, cerca un vertice che migliori il valore della funzione obiettivo. L'individuazione di un tale vertice richiede la determinazione di una direzione ammissibile di crescita, come evidenziato nel §3.2.3. Introduciamo i concetti fondamentali per mezzo di un esempio.

Esempio 3.29. Esecuzione "informale" del Simplesso Primale

Riprendiamo il problema della Pintel, e consideriamo il vertice $\bar{x} = [\,1\,,\,7\,]$, che già sappiamo non essere una soluzione ottima; tale vertice è una soluzione primale di base non degenere in quanto $B = I(\,\bar{x}\,) = \{\,2\,,\,3\,\}$. Poiché $\bar{x}$ non è una soluzione ottima per il problema, dovrà necessariamente esistere una direzione ξ che sia di crescita, cioè tale per cui $c\xi > 0$, e che sia ammissibile, cioè tale che valga $A_B\xi \leq 0$. Cerchiamo allora di determinare tale direzione considerando la coppia di sistemi studiata dal Lemma di Farkas:

$$\begin{array}{rcrcl} & & \xi_2 & \leq & 0 \\ 2\xi_1 & + & \xi_2 & \leq & 0 \\ 500\xi_1 & + & 200\xi_2 & > & 0 \end{array} \quad , \quad \begin{array}{rcrcl} & & 2y_2 & = & 500 \\ y_1 & + & y_2 & = & 200 \\ y_1 & , & y_2 & \geq & 0 \end{array} \; .$$



Per verificare quale tra i due sistemi ammetta soluzione, calcoliamo la sola soluzione candidata a risolvere il sistema duale, $\bar{y}_B = cA_B^{-1} = [\,-50\,,\,250\,]$. Essendo $\bar{y}_2 < 0$, tale soluzione non è ammissibile: sappiamo pertanto che esiste una soluzione per il sistema primale, vale a dire una direzione ammissibile di crescita per $\bar{x}$. Infatti $\bar{\xi} = [\,0.5\,,\,-1\,]$ è ammissibile per il sistema primale. Come evidenziato nella figura qui accanto, $\bar{\xi}$ individua la direzione di spostamento che va dal vertice $[\,1\,,\,7\,]$ verso il vertice ad esso *adiacente* $[\,4\,,\,1\,]$; in particolare, la direzione è uno dei (due) generatori del cono delle direzioni ammissibili per $[\,1\,,\,7\,]$, espresso come cono finitamente generato. Lungo tale direzione ci si sposta rimanendo sulla faccia individuata dal vincolo $2x_1 + x_2 \leq 9$, mentre ci si allontana dalla faccia individuata dal vincolo $x_2 \leq 7$. Possiamo allora migliorare il valore della funzione obiettivo spostandoci lungo $\bar{\xi}$ il più possibile, purché si rimanga all'interno della regione ammissibile: un tale spostamento ci porta al vertice $[\,4\,,\,1\,]$, che come già sappiamo è il punto ottimo. Per verificare tale ottimalità iteriamo il ragionamento, e cerchiamo di determinare se esistono direzioni ammissibili di crescita rispetto a $[\,4\,,\,1\,]$: poiché il sistema duale

$$\begin{array}{rcrcl} y_1 & + & 2y_3 & = & 500 \\ & & y_3 & = & 200 \\ y_1 & , & y_3 & \geq & 0 \end{array}$$

ammette la soluzione non negativa $[\,100\,,\,200\,]$, si può concludere che $[\,4\,,\,1\,]$ è una soluzione ottima del problema.

Formalizziamo ora le idee dell'esempio precedente, specificando, in particolare, le modalità di individuazione di una direzione ammissibile di crescita, se essa esiste. Per questo ci serviremo dei risultati sviluppati nel §3.1.1, ed in particolare di (3.15). Assumiamo di avere a disposizione una base ammissibile B e la corrispondente soluzione di base $\bar{x} = A_B^{-1}b_B$; supponiamo inoltre, per il momento, che la base sia non degenere, ossia $B = I(\,\bar{x}\,)$. Vogliamo determinare se esiste una direzione ammissibile di crescita rispetto a $\bar{x}$, ossia $\xi \in C(\,\bar{x}\,)$ tale che $c\xi > 0$. Sfruttiamo adesso la caratterizzazione (3.15) di $C(\,\bar{x}\,)$: i generatori del cono sono gli opposti delle colonne dell'inversa della matrice di base. Come

vedremo, ciascuna di queste direzioni è "naturalmente associata" ad uno specifico vincolo in base; per sottolineare questo indicizziamo le direzioni con gli indici $i \in B$ mediante

$$\xi_i = -A_B^{-1} u_{B(i)} \qquad\qquad i = 1, \ldots, n$$

dove $B(i)$ indica la posizione dell'indice i in B (la riga di A_B in cui si trova A_i), mentre $u_j \in \mathbb{R}^n$ indica il vettore che ha tutte le componenti nulle tranne la j-esima, che vale 1 (si tratta cioè del j-esimo vettore della base canonica di $\mathbb{R}^n$). Usando il Lemma 3.1 possiamo quindi affermare che *esiste* $\xi \in C(\bar{x})$ *tale che* $c\xi > 0$ *se e solo se* $c\xi_h > 0$ *per un qualche* $h \in B$.

Nel caso di una base non degenere disponiamo quindi di un meccanismo in grado di determinare una direzione ammissibile di crescita, se ce n'è una oppure dimostrare che $\bar{x}$ è una soluzione ottima. Infatti, $c\xi_i \leq 0$ per ogni $i \in B$ equivale a $\bar{y}_i = cA_B^{-1}u_{B(i)} = -c\xi_i \geq 0$ per ogni $i \in B$, ossia $\bar{y} = [\, \bar{y}_B \,,\, 0\,] \geq 0$: pertanto, $\bar{x}$ e $\bar{y}$ sono entrambe ammissibili (rispettivamente per il primale ed il duale), e siccome la Proposizione 3.3 garantisce che sono anche complementari, il Teorema degli Scarti Complementari ci garantisce di aver risolto entrambi i problemi.

Nel caso in cui $\bar{x}$ sia una soluzione di base degenere, però, il risultato vale in una sola direzione. In particolare, se $\bar{y}_B \geq 0$ allora possiamo comunque affermare di aver determinato una soluzione ottima. Se invece $c\xi_h > 0$ per un qualche $h \in B$, non possiamo per questo affermare che $\bar{x}$ non sia ottima, perché in effetti non abbiamo alcuna garanzia che ξ_h sia una direzione ammissibile per $\bar{x}$. Infatti, ξ_h è un generatore del cono $C_B = \{\, \xi \in \mathbb{R}^n \;:\; A_B\xi \leq 0 \,\}$, che peró non coincide col cono delle direzioni ammissibili $C(\bar{x})$ in quanto $I(\bar{x}) \supset B$; si ha quindi $C(\bar{x}) \subset C_B$, e può accadere che $\xi_h \notin C(\bar{x})$, ossia esiste un indice $i \in I \setminus B$ tale che $A_i\xi_h > 0$. Si può infatti dimostrare che $C(\bar{x})$ coincide con l'intersezione di tutti i coni C_B al variare di $B \subseteq I(\bar{x})$, e che i generatori di $C(\bar{x})$ sono tutti e soli i vettori che sono generatori di uno dei coni C_B e che appartengono a $C(\bar{x})$ (si ricordi il Teorema 3.7). Questo equivale al fatto che la direzione ξ_h è una soluzione non del "vero" Primale Ristretto, ma di un suo rilassamento in cui sono stati arbitrariamente ignorati i vincoli in $I \setminus B$: se il sistema così approssimato non ha soluzioni allora non ne ha neanche quello originario e deve avere soluzione il Duale Ristretto (infatti si può affermare che $\bar{x}$ è ottima), ma determinare una soluzione del sistema approssimato non garantisce di averne determinata una del Primale Ristretto originario.

Esempio 3.30. Direzioni ξ_h non ammissibili

Si consideri il seguente coppia di problemi duali

$$(P)\quad \begin{array}{lrcrcl} \max & x_1 & + & x_2 & & \\ & & - & x_2 & \leq & 1 \\ & -x_1 & - & x_2 & \leq & 1 \\ & x_1 & & & \leq & 0 \\ & x_1 & + & x_2 & \leq & 2 \end{array} \qquad , \qquad (D)\quad \begin{array}{lrcrcrcrcl} \min & y_1 & + & y_2 & & & + & 2y_4 & & \\ & & - & y_2 & + & y_3 & + & y_4 & = & 1 \\ & -y_1 & - & y_2 & & & + & y_4 & = & 1 \\ & y_1 & , & y_2 & , & y_3 & , & y_4 & \geq & 0 \end{array}$$

e la base $B = \{\, 1\,,\, 2\,\}$. Le corrispondenti matrice A_B e soluzioni di base $\bar{x} = A_B^{-1}b_B$ e $\bar{y}_B = cA_B^{-1}$ sono

$$A_B = \begin{bmatrix} 0 & -1 \\ -1 & -1 \end{bmatrix},\ \bar{x} = \begin{bmatrix} 1 & -1 \\ -1 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \end{bmatrix},\ \bar{y}_B = \begin{bmatrix}\, 1\,,\, 1 \,\end{bmatrix}\begin{bmatrix} 1 & -1 \\ -1 & 0 \end{bmatrix} = \begin{bmatrix}\, 0\,,\, -1 \,\end{bmatrix}$$

ed è immediato verificare che la soluzione $\bar{x}$ è ammissibile. L'insieme degli indici dei vincoli attivi è $I(\bar{x}) = \{\, 1\,,\, 2\,,\, 3\,\}$ e quindi $\bar{x}$ è degenere. Infatti, si ha

$$C_B = \left\{ \begin{array}{rcrcl} & - & \xi_2 & \leq & 0 \\ -\xi_1 & - & \xi_2 & \leq & 0 \end{array} \right. \quad \supset \quad C(\bar{x}) = \left\{ \begin{array}{rcrcl} & - & \xi_2 & \leq & 0 \\ -\xi_1 & - & \xi_2 & \leq & 0 \\ \xi_1 & & & \leq & 0 \end{array} \right. .$$

I generatori di C_B sono gli opposti delle colonne di A_B^{-1}; il fatto che $\bar{y}_2 = -1$ corrisponde al fatto che

$$c\xi_2 = \begin{bmatrix}\, 1\,,\, 1 \,\end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = 1 = -\bar{y}_2 > 0$$

e quindi ξ_2 è una direzione di crescita. Poiché $A_3\xi_2 = 1 > 0$, come è facile verificare, si ha però che $\xi_2 \notin C(\bar{x})$, ossia la direzione ξ_2 non è ammissibile per $\bar{x}$.

Il fatto che una direzione di crescita ξ si riveli non essere ammissibile non la rende per ció stesso inutilizzabile; come vedremo, uno spostamento "degenere" lungo una direzione di questo tipo causa

comunque un *cambiamento di base*, il che permette di proseguire con l'algoritmo. Scegliamo quindi per il momento di ignorare il problema, e di utilizzare la direzione come se fosse ammissibile. Per questo calcoliamo il massimo passo di spostamento che è possibile effettuare lungo ξ mantenendo l'ammissibilità, ossia il massimo valore di λ per cui la soluzione $x(\lambda) = \bar{x} + \lambda\xi$ risulta essere ammissibile; sappiamo che questo valore sarà non negativo, in quanto $x(0) = \bar{x}$ è ammissibile. Per $i \in B$, risulta

$$A_i x(\lambda) = A_i\bar{x} + \lambda A_i\xi \leq A_i\bar{x} \leq b_i$$

per ogni $\lambda \geq 0$, in quanto $A_i\xi \leq 0$. Analogamente, se $i \in N$ è tale che $A_i\xi \leq 0$ la soluzione $x(\lambda)$ soddisfa il vincolo i-esimo per ogni valore non negativo di λ. Se invece $A_i\xi > 0$, abbiamo

$$A_i x(\lambda) = A_i\bar{x} + \lambda A_i\xi \leq b_i \quad \Longleftrightarrow \quad \lambda \leq (b_i - A_i\bar{x})/(A_i\xi) \ .$$

Pertanto, il massimo passo che può essere effettuato lungo ξ a partire da $\bar{x}$ senza violare il vincolo i-esimo risulta essere

$$\lambda_i = \begin{cases} (b_i - A_i\bar{x})/(A_i\xi) & \text{se } A_i\xi > 0 \\ +\infty & \text{altrimenti} \end{cases} \ . \tag{3.27}$$

Scegliendo il più piccolo di questi valori, ovvero

$$\bar{\lambda} = \min\{\lambda_i \ : \ i \in N\} \ , \tag{3.28}$$

si ottiene il massimo passo di spostamento consentito. In particolare, se $A_N\xi \leq 0$, abbiamo $\bar{\lambda} = +\infty$ e $x(\lambda)$ è ammissibile per ogni valore positivo di λ: poiché ξ è una direzione di crescita, il valore della funzione obiettivo cresce indefinitamente al crescere di λ, e siamo nel caso in cui il problema primale (P) è illimitato ed il problema duale (D) è vuoto. Se invece esiste un indice $i \in N$ tale che $A_i\xi > 0$, abbiamo $\bar{\lambda} < +\infty$: $x(\lambda)$ è ammissibile per ogni $\lambda \in [0, \bar{\lambda}]$ ma non ammissibile per $\lambda > \bar{\lambda}$. Nel caso in cui $\bar{x}$ sia una soluzione di base non degenere, abbiamo $\lambda_i > 0$ per ogni $i \in N$ e quindi $\bar{\lambda} > 0$: come già verificato in precedenza, la direzione ξ è ammissibile. Possiamo quindi effettuare il massimo spostamento consentito $\bar{\lambda}$, spostandoci sul punto $x(\bar{\lambda})$ e migliorando strettamente il valore della funzione obiettivo ($cx(\bar{\lambda}) > c\bar{x}$). Abbiamo definito in tal modo la tipica iterazione dell'algoritmo del simplesso. Per poter ripetere il procedimento, tuttavia, dobbiamo essere certi che $x(\bar{\lambda})$ sia un vertice del poliedro come lo era $\bar{x}$: è possibile dimostrare che questo è in effetti il caso. Questo risultato si basa su una proprietà peculiare delle direzioni ξ_h: $A_i\xi_h = 0$ per ogni $i \in B \setminus \{h\}$, mentre $A_h\xi_h = -1$, come è immediato verificare dalla definizione di ξ_h. Geometricamente, questo significa che ξ_h è perpendicolare a tutti gli A_i in base (tranne A_h), e che quindi "punta all'interno della frontiera" degli iperpiani corrispondenti; in altri termini, per ogni possibile valore di λ

$$A_i x(\lambda) = A_i(\bar{x} + \lambda\xi_h) = A_i\bar{x} = b_i \qquad i \in B \setminus \{h\} \ , \tag{3.29}$$

ossia i vincoli in base (tranne l'h-esimo), che sono per definizione attivi in $\bar{x}$, sono anche attivi in $x(\lambda)$. Invece il fatto che $A_h\xi_h = -1$ significa che A_h non è attivo in $x(\lambda)$ per $\lambda > 0$; infatti, al crescere di λ il punto "si allontana dalla frontiera dell'h-esimo iperpiano", verso l'interno del semispazio corrispondente. Da questo segue:

Lemma 3.5. Data una base ammissibile B, la corrispondente soluzione di base $\bar{x}$, una direzione $\xi = \xi_h$ per $h \in B$, ed un indice $k \in N$ tale che $\bar{\lambda} = \bar{\lambda}_k$, $B' = B \cup \{k\} \setminus \{h\}$ è una base e $x(\bar{\lambda})$ è la corrispondente soluzione primale di base.

Dimostrazione È facile mostrare che A_k non è linearmente dipendente dalle righe di $A_{B\setminus\{h\}}$. Infatti, ricordiamo che $A_i\xi = 0$ per ogni $i \in B \setminus \{h\}$, mentre $A_k\xi > 0$ dato che $\bar{\lambda}_k < +\infty$ (cf. (3.27)). Pertanto, se esistesse un vettore $v \in \mathbb{R}^{n-1}$ per cui $vA_{B\setminus\{h\}} = A_k$, moltiplicando entrambe i termini di questa relazione per ξ si otterrebbe $0 < A_k\xi = vA_{B\setminus\{h\}}\xi = v0 = 0$, una contraddizione. Da questo e dal fatto che, per ipotesi, le righe di $A_{B\setminus\{h\}}$ sono linearmente indipendenti, segue che B' è una base.

Per dimostrare che $x(\bar{\lambda})$ è la soluzione di base associata a B' basta verificare che $B' \subseteq I(x(\bar{\lambda}))$. Questo per $i \in B \setminus \{h\}$ segue da (3.29), mentre per $i = k$ segue da (3.28): il passo scelto è precisamente quello che rende attivo il vincolo k-esimo, come è immediato verificare. ◇

Nonostante l'analisi precedente sia stata sviluppata per il caso non degenere, è facile verificare che essa si estende senza alcun problema a quello degenere, in cui possiamo avere sia $\bar{\lambda} > 0$ che $\bar{\lambda} = 0$.

Nel primo caso si effettua lo spostamento analogamente al caso non degenere. Se invece il massimo spostamento consentito è nullo, la direzione $\xi = \xi_h$ individuata non è ammissibile in quanto esiste un indice $k \in I \setminus B$ tale che $A_k\xi > 0$, e quindi $\xi \notin C(\bar{x})$. Poiché $\bar{\lambda} = \bar{\lambda}_k = 0$, $x(\bar{\lambda}) = \bar{x}$ e quindi non abbiamo cambiato vertice; abbiamo comunque individuato una *diversa base* $B \cup \{k\} \setminus \{h\}$ che individua lo stesso vertice $\bar{x}$. Parliamo in questo caso di un *cambiamento di base degenere*.

Un cambiamento di base non degenere, ovvero tale che $\bar{\lambda} > 0$, consente di spostarsi da un vertice della regione ammissibile ad un altro; i due vertici sono collegati da uno spigolo e vengono detti *adiacenti*. Inoltre, il nuovo vertice migliora il valore della funzione obiettivo rispetto al vertice precedente. Un cambiamento di base degenere, ovvero tale che $\bar{\lambda} = 0$, non consente di cambiare vertice ma fornisce una nuova base relativa allo stesso vertice. Utilizzando una regola che impedisca di individuare basi già analizzate precedentemente (si veda il Teorema 3.19) nelle successive iterazioni degeneri, nel caso pessimo si analizzano tutte le basi relative al vertice: quindi, in un numero finito di passi o si determina una direzione di crescita ammissibile, oppure si certifica l'ottimalità del vertice.

Esempio 3.31. Cambiamenti di base degeneri

Per esemplificare lo sviluppo precedente di consideri la seguente coppia di problemi duali

$$(P)\quad \begin{array}{rrrcr} \max & & x_2 & & \\ & - & x_2 & \leq & 1 \\ -x_1 & - & x_2 & \leq & 1 \\ -x_1 & & & \leq & 0 \\ -x_1 & + & x_2 & \leq & -1 \\ x_1 & & & \leq & 2 \end{array} \qquad (D)\quad \begin{array}{rrrrrrrrrrr} \min & y_1 & + & y_2 & & & - & y_4 & + & 2y_5 & \\ & & - & y_2 & - & y_3 & - & y_4 & + & y_5 & = 0 \\ & -y_1 & - & y_2 & & & + & y_4 & & & = 1 \\ & y_1 & , & y_2 & , & y_3 & , & y_4 & , & y_5 & \geq 0 \end{array}$$

e la base $B^1 = \{1, 2\}$. Le corrispondenti matrice e soluzione primale di base sono

$$A_{B^1} = \begin{bmatrix} 0 & -1 \\ -1 & -1 \end{bmatrix} \quad , \quad \bar{x} = A_{B^1}^{-1} b_{B^1} = \begin{bmatrix} 1 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \end{bmatrix} .$$

La soluzione $\bar{x}$ è ammissibile in quanto

$$A_{N^1}\bar{x} = \begin{bmatrix} -1 & 0 \\ -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix} \leq b_{N^1} = \begin{bmatrix} 0 \\ -1 \\ 2 \end{bmatrix} .$$

L'insieme degli indici dei vincoli attivi è $I = \{1, 2, 3, 4\}$, e quindi $\bar{x}$ è degenere. Verificare l'ottimalità di $\bar{x}$, o individuare una sua direzione ammissibile di crescita, corrisponde a verificare quale dei due sistemi ristretti

$$(P_R)\quad \begin{cases} & & \xi_2 & > & 0 \\ & - & \xi_2 & \leq & 0 \\ -\xi_1 & - & \xi_2 & \leq & 0 \\ -\xi_1 & & & \leq & 0 \\ -\xi_1 & + & \xi_2 & \leq & 0 \end{cases} \qquad (D_R)\quad \begin{cases} & - & y_2 & - & y_3 & - & y_4 & = 0 \\ -y_1 & - & y_2 & & & + & y_4 & = 1 \\ y_1 & , & y_2 & , & y_3 & , & y_4 & \geq 0 \end{cases}$$

ha soluzione. Poiché (D_R) è sottodeterminato (almeno per quanto riguarda le eguaglianze), una possibile strategia è quella di fissare a zero le variabili 3 e 4 (quelle fuori base) e risolvere il sistema ristretto a quelle in base; ciò corrisponde ad ignorare temporaneamente i vincoli attivi 3 e 4. In altri termini, la soluzione duale di base corrispondente a B^1 è

$$y^1_{B^1} = cA^{-1}_{B^1} = \begin{bmatrix} 0, 1 \end{bmatrix} \begin{bmatrix} 1 & -1 \\ -1 & 0 \end{bmatrix} = \begin{bmatrix} -1, 0 \end{bmatrix} .$$

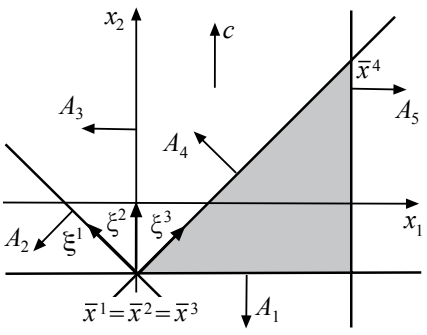
Poiché $y^1_1 < 0$, selezionando $h = 1$ ($B^1(h) = 1$) la direzione

$$\xi^1 = -A^{-1}_{B^1} u_{B^1(h)} = -\begin{bmatrix} 1 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix} ,$$

è di crescita per. Poiché però

$$A_3\xi^1 = \begin{bmatrix} -1, 0 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} = 1 > 0$$

$$A_4\xi^1 = \begin{bmatrix} -1, 1 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} = 2 > 0$$



questa direzione non è ammissibile per $\bar{x}$, come si può facilmente verificare dalla figura qui sopra. Quindi, la scelta (arbitraria) di fissare $y_3 = y_4 = 0$ in (D_R) non si è rivelata vincente: non si è ottenuta né una soluzione duale ammissibile, né una direzione ammissibile di crescita. Possiamo però effettuare un cambiamento di base degenere, scegliendo $k_1 = 3$. Per la nuova base $B^2 = B^1 \setminus \{ h \} \cup \{ k \} = \{ 3 , 2 \}$ risulta

$$A_{B^2} = \begin{bmatrix} -1 & 0 \\ -1 & -1 \end{bmatrix} \quad , \quad A_{B^2}^{-1} = \begin{bmatrix} -1 & 0 \\ 1 & -1 \end{bmatrix} \quad , \quad y^2_{B^2} = \begin{bmatrix} 0 , 1 \end{bmatrix} \begin{bmatrix} -1 & 0 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 , -1 \end{bmatrix} \ ,$$

da cui la scelta $h = 4$ ($B^2(h) = 2$) consente di individuare la direzione di crescita

$$\xi^2 = -A_{B^2}^{-1} u_{B^2(h)} = \begin{bmatrix} -1 & 0 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \ ,$$

che però ancora non è ammissibile per $\bar{x}$ in quanto

$$A_4 \xi^2 = \begin{bmatrix} -1 , 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = 1 > 0 \ .$$

Possiamo comunque effettuare un nuovo cambiamento di base degenere sostituendo $k = 4$ a h: otteniamo così la base $B^3 = \{ 3 , 4 \}$ per cui abbiamo

$$A_{B^3} = \begin{bmatrix} -1 & 0 \\ -1 & 1 \end{bmatrix} \quad , \quad A_{B^3}^{-1} = \begin{bmatrix} -1 & 0 \\ -1 & 1 \end{bmatrix} \quad , \quad y^3_{B^3} = \begin{bmatrix} 0 , 1 \end{bmatrix} \begin{bmatrix} -1 & 0 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} -1 , 1 \end{bmatrix} \ .$$

La scelta $h = 3$ ($B^3(h) = 1$) consente allora di individuare la direzione di crescita

$$\xi^3 = -A_{B^3}^{-1} u_{B^3(h)} = \begin{bmatrix} -1 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \ ,$$

che è ammissibile per $\bar{x}$ in quanto

$$A_1 \xi^3 = \begin{bmatrix} 0 , -1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = -1 < 0 \quad , \quad A_2 \xi^3 = \begin{bmatrix} -1 , -1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = -2 < 0 \ .$$

Pertanto, ξ^3 risolve (P_R) e quindi $\bar{x}$ non è una soluzione ottima di (P). Infatti, il massimo spostamento consentito lungo ξ^3, calcolato mediante (3.28),

$$\bar{\lambda} = \lambda_5 = \frac{b_5 - A_5 \bar{x}}{A_5 \xi_3} = \frac{2 - 0}{1} = 2 \ ,$$

permette di individuare $k = 5$, la nuova base $B^4 = \{ 5 , 4 \}$ e la corrispondente soluzione di base

$$\bar{x}^4 = A_{B^4}^{-1} b_{B^4} = x(\bar{\lambda}) = \bar{x} + \bar{\lambda} \xi^3 = \begin{bmatrix} 0 \\ -1 \end{bmatrix} + 2 \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \ ,$$

per cui la funzione obiettivo vale $c\bar{x}^4 = 1 > -1 = c\bar{x}$.

Possiamo ora fornire una descrizione formale di un algoritmo per la soluzione dei problemi di *PL*, che verrà chiamato *Simplesso Primale*.

```
procedure ( B , x̄ , ȳ , stato ) = Simplesso_Primale( A , b , c , B ) {
  for( stato = "" ; ; ) {    // invariante: B primale ammissibile
    x̄ = A_B^{-1} b_B; ȳ = [ ȳ_B , ȳ_N ] = [ c A_B^{-1} , 0 ];
    if( ȳ_B ≥ 0 )  then { stato = "ottimo"; break; }
    h = min{ i ∈ B : ȳ_i < 0 }; ξ = −A_B^{-1} u_{B(h)};
    if( A_N ξ ≤ 0 ) then { stato = "P.illimitato"; break; }
    λ̄ = min{ λ_i = ( b_i − A_i x̄ ) / ( A_i ξ ) , A_i ξ > 0 , i ∈ N };
    k = min{ i ∈ N : λ_i = λ̄ }; B = B ∪ { k } \ { h };
    }
  }
```

Procedura 3.1: Simplesso Primale

L'algoritmo riceve in input una descrizione del problema ed un base B che si assume essere *primale ammissibile*, ed iterativamente esegue le seguenti operazioni:

1. Verifica l'ottimalità della soluzione primale di base $\bar{x}$, nel qual caso termina fornendo anche la corrispondente soluzione duale $\bar{y}$, oppure individua una direzione di crescita ξ.

2. Calcola il massimo passo di spostamento lungo la direzione ξ, che può essere nullo in caso di degenerazione primale: se il passo risulta essere $+\infty$, cioè se la direzione ξ consente una crescita illimitata della funzione obiettivo, allora l'algoritmo termina avendo provato che (P) è illimitato e quindi (D) è vuoto.
3. Viene aggiornata la base: $B' = B \cup \{ k \} \setminus \{ h \}$, con h e k selezionati come precedentemente definito; ciò corrisponde o ad un cambiamento di vertice, ed in questo caso il valore della funzione obiettivo cresce, oppure ad un cambiamento di base degenere.

L'algoritmo termina in un numero finito di passi fornendo una base ottima B e la corrispondente coppia di soluzioni ottima, se esiste, oppure fornendo la risposta che il problema è illimitato. Infatti, quando la direzione di crescita ξ è ammissibile, o si determina che il problema è illimitato (ed in tal caso l'algoritmo termina), oppure viene individuata una nuova soluzione di base avente un valore della funzione obiettivo maggiore di quello corrente. Essendo tale nuova soluzione un vertice, ed essendo il numero di vertici finito, la proprietà di generare una sequenza crescente di valori della funzione obiettivo di (P) garantisce che l'algoritmo esegua un numero finito di iterazioni in corrispondenza di vertici non degeneri. Quando invece l'algoritmo visita un vertice degenere, allora, come già evidenziato, esso può generare direzioni non ammissibili e non spostarsi dal vertice corrente. L'algoritmo proposto garantisce però la terminazione finita anche in presenza di basi primali degeneri utilizzando uno specifico *criterio di selezione degli indici entrante ed uscente* noto come *regola anticiclo di Bland*: nel caso in cui esistano più indici h candidati ad uscire dalla base corrente B e/o più indici k candidati ad entrare in base, l'algoritmo seleziona sempre l'indice minimo, cioè $h = \min\{ i \in B : \bar{y}_i < 0 \}$ e $k = \min\{ i \in N : \lambda_i = \bar{\lambda} \}$. Nel seguito (cf. Teorema 3.19) dimostreremo che questo criterio garantisce che una base non venga esaminata più di una volta e quindi, essendo il numero di basi finito, che venga eseguito un numero finito di iterazioni anche in corrispondenza dei vertici degeneri. Osserviamo che l'ordine delle righe e delle colonne è arbitrario: basta fissare un qualsiasi ordinamento degli indici e applicare il criterio di Bland rispetto a tale ordinamento.

È utile discutere brevemente alcuni elementi dell'algoritmo:

- Abbiamo assunto che una base primale ammissibile venga fornita in input, ma in generale la determinazione di una qualsiasi base primale ammissibile è non banale; per questo si può ricorrere ad una fase di inizializzazione che verrà descritta nel seguito.
- Anche se non evidenziato nello pseudo-codice, nel caso in cui il problema è superiormente illimitato può essere utile fornire in output anche la direzione ξ determinata. Questa, insieme a $\bar{x}$ costituisce infatti un *certificato di illimitatezza* del problema: qualsiasi punto nella semiretta $x(\lambda) = \bar{x} + \lambda\xi$ per $\lambda \geq 0$ è una soluzione ammissibile con un costo che cresce al crescere di λ. In effetti è possibile derivare da ξ anche un *certificato di inammissibilità* del problema duale. Questo si basa sul fatto che

$$c\xi > 0 \qquad \text{e} \qquad A\xi = \begin{bmatrix} A_B \\ A_N \end{bmatrix} \xi = \begin{bmatrix} -u_{B(h)} \\ z_N \end{bmatrix} \qquad \text{con} \qquad z_N = A_N\xi \leq 0 \ .$$

  Qualsiasi soluzione ammissibile del duale dovrebbe avere $yA = c$ e $y \geq 0$: moltiplicando entrambe i lati della prima uguaglianza per ξ si ottiene un uguaglianza che deve essere verificata per ogni soluzione y, ossia

$$c\xi + y_h = z_N y_N \ ,$$

  che però non può avere soluzioni in quanto il lato destro è ≤ 0 (tutte le y_i sono non negative e $z_N \leq 0$), mentre quello sinistro è positivo ($c\xi > 0$, $y_h > 0$). Quindi ξ fornisce un certificato "compatto" del fatto che i vincoli di non-negatività sulle y_N sono incompatibili con quello sulla y_h e con i vincoli strutturali $yA = c$.
- Ad ogni passo occorre calcolare l'inversa della matrice A_B, o equivalentemente fattorizzare A_B, e poi calcolare prodotti scalari o risolvere sistemi lineari per determinare $\bar{x}$, $\bar{y}_B$ e ξ: il primo passo ha costo computazionale $O(n^3)$ ed i successivi $O(n^2)$ per matrici dense non strutturate utilizzando metodi elementari. In pratica è però possibile determinare l'inversa di $A_{B'}$ ($B' =$

$B \setminus \{ h \} \cup \{ k \}$) a partire dalla conoscenza di A_B^{-1}, o aggiornare le fattorizzazioni, con costo computazionale circa $O(n^2)$. Inoltre, la matrice A nelle applicazioni reali è di solito molto sparsa, e questo può essere sfruttato per ridurre ulteriormente il costo computazionale.

- Sperimentalmente si prova che la scelta di h suggerita dalla regola di Bland produce un algoritmo abbastanza inefficiente. Poiché nei problemi reali l'algoritmo, anche in assenza di regole anticiclo, cicla raramente, si preferisce la scelta di direzioni che garantiscano (sperimentalmente) una migliore efficienza della procedura, quali ad esempio selezionare l'indice h a cui corrisponde il valore $\bar{y}_i$ minore (negativo con valore assoluto più grande), il che determina la direzione che fornisce il massimo incremento unitario della funzione obiettivo (dato che $cx(\lambda) = c\bar{x} - \lambda\bar{y}_i$). Metodi alternativi al criterio di Bland per evitare cicli sono le cosiddette tecniche di perturbazione, in cui i dati vengono leggermente modificati in modo da rendere tutte le soluzioni di base non degeneri, senza perdere l'ottimalità delle basi ottime nel problema originario.

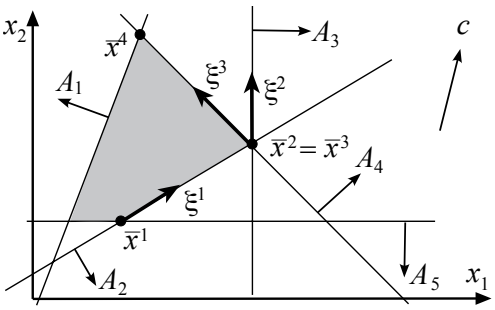
Esercizio 3.12. Determinare la complessità delle singole istruzioni dell'algoritmo *Simplesso_Primale*.

Esercizio 3.13. Determinare l'occupazione di memoria dell'algoritmo *Simplesso_Primale*.

Esempio 3.32. Esecuzione dell'algoritmo del simplesso primale

Consideriamo la coppia asimmetrica di problemi duali dell'Esempio 3.25 e applichiamo l'algoritmo del simplesso primale a partire dalla base primale ammissibile $B^1 = \{ 2, 5 \}$ già analizzata in quell'esempio, a cui corrisponde il vertice $\bar{x}^1 = [4, 4]$. Come abbiamo già visto, la soluzione duale di base è $\bar{y}^1 = [0, 1, 0, 0, -5]$: quindi l'indice uscente è $h = 5$, con $B^1(h) = 2$ e la direzione di crescita è

$$\xi^1 = -A_{B^1}^{-1} u_{B^1(h)} = \begin{bmatrix} -1 & 2 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix} .$$



La ricerca dell'indice uscente è effettuata analizzando i vincoli non in base:

$$A_{N^1}\xi^1 = \begin{bmatrix} -2 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} -3 \\ 2 \\ 3 \end{bmatrix} .$$

Essendo positive solo $A_3\xi^1$ ed $A_4\xi^1$, l'insieme $J^1 = \{ i \in N^1 : A_i\xi^1 > 0 \}$ degli indici candidati ad entrare in base è $\{ 3, 4 \}$; infatti, dalla Figura 3.9 si può osservare come, partendo da $\bar{x}^1$, e muovendosi nella direzione ξ^1 lungo l'iperpiano (retta) di supporto del secondo vincolo, si incontrano gli iperpiani (rette) di supporto de vincoli 3 e 4. Il valore negativo di $A_1\xi^1$ indica che ci si sta allontanando dall'iperpiano (retta) di supporto del vincolo 1.

Determiniamo ora il passo dello spostamento lungo ξ^1, dato da $\bar{\lambda} = \min\{ \lambda_i : i \in J^1 \}$, dove λ_i è data da (3.27):

$$\lambda_3 = (14 - 8)/3 = 2 \qquad , \qquad \lambda_4 = (8 - 4)/2 = 2 \ ;$$

essendo $\bar{\lambda} = \lambda_3 = \lambda_4 = 2$, per la regola anticiclo di Bland si ha $k = \min\{ 3, 4 \} = 3$. La nuova base è perciò $B^2 = B^1 \setminus \{ h \} \cup \{ k \} = \{ 2, 5 \} \setminus \{ 5 \} \cup \{ 3 \} = \{ 2, 3 \}$.

Effettuiamo la seconda iterazione:

$$A_{B^2} = \begin{bmatrix} 1 & -2 \\ 1 & 0 \end{bmatrix} \quad , \quad A_{B^2}^{-1} = \begin{bmatrix} 0 & 1 \\ -1/2 & 1/2 \end{bmatrix} \quad , \quad b_{B^2} = \begin{bmatrix} -4 \\ 8 \end{bmatrix} \ ;$$

pertanto la nuova soluzione di base primale è

$$\bar{x}^2 = A_{B^2}^{-1} b_{B^2} = \begin{bmatrix} 0 & 1 \\ -1/2 & 1/2 \end{bmatrix} \begin{bmatrix} -4 \\ 8 \end{bmatrix} = \begin{bmatrix} 8 \\ 6 \end{bmatrix}$$

mentre quella duale è

$$\bar{y}_{B^2} = cA_{B^2}^{-1} = \begin{bmatrix} 1, 3 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -1/2 & 1/2 \end{bmatrix} = \begin{bmatrix} -3/2, 5/2 \end{bmatrix} \quad , \quad \bar{y}_{N^2} = 0$$

e quindi $\bar{y}^2 = [\,0\,,\,-3/2\,,\,5/2\,,\,0\,,\,0\,]$. Essendoci solo $\bar{y}^2_2 < 0$, l'indice uscente è necessariamente $h = 2$. Calcoliamo la direzione di crescita e il passo lungo essa:

$$\xi^2 = \begin{bmatrix} 0 & -1 \\ 1/2 & -1/2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1/2 \end{bmatrix} \quad , \quad A_{N^2}\xi^2 = \begin{bmatrix} -2 & 1 \\ 1 & 1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1/2 \end{bmatrix} = \begin{bmatrix} 1/2 \\ 1/2 \\ -1/2 \end{bmatrix} \ ,$$

$$J^2 = \{\,1\,,\,4\,\} \quad , \quad \lambda_1 = 22 \quad , \quad \lambda_4 = 0 = \bar{\lambda} \quad , \quad k = 4 \ .$$

$\lambda_4 = 0$ è una conseguenza del fatto che $A_4\bar{x}^2 = 14 = b_4$, cioè che la soluzione di base è primale degenere, come avevamo già mostrato nell'Esempio 3.26. Pertanto, $\bar{\lambda} = 0$ indica che la direzione di crescita ξ^2 non è una direzione ammissibile. La nuova base è $B^3 = B^2 \setminus \{\,2\,\} \cup \{\,4\,\} = \{\,3\,,\,4\,\}$ e la corrispondente soluzione primale di base è

$$A_{B^3} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \quad , \quad A^{-1}_{B^3} = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \quad , \quad b_{B^3} = \begin{bmatrix} 8 \\ 14 \end{bmatrix} \quad , \quad \bar{x}^3 = \begin{bmatrix} 8 \\ 6 \end{bmatrix} \ .$$

Abbiamo effettuato un cambio di base degenere, in quanto la soluzione primale individuata coincide con la precedente; la nuova soluzione di base duale, come visto nell'Esempio 3.26, è diversa dalla precedente, infatti:

$$\bar{y}_{B^3} = \begin{bmatrix}\, 1\,,\,3\,\end{bmatrix} \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix}\, -2\,,\,3\,\end{bmatrix} \quad , \quad \bar{y}_{N^3} = 0 \quad , \quad \bar{y}^3 = \begin{bmatrix}\, 0\,,\,0\,,\,-2\,,\,3\,,\,0\,\end{bmatrix} \ .$$

L'indice uscente è quindi $h = 3$, e

$$\xi^3 = \begin{bmatrix} -1 & 0 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \quad , \quad A_{N^3}\xi^3 = \begin{bmatrix} -2 & 1 \\ 1 & -2 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ -3 \\ -1 \end{bmatrix} \ ;$$

si ha pertanto $J^3 = \{\,1\,\}$ e $\bar{\lambda} = \lambda_1 = 11/3$, l'indice entrante è $k = 1$ e la nuova base è $B^4 = B^3 \setminus \{\,3\,\} \cup \{\,1\,\} = \{\,1\,,\,4\,\}$. Poiché

$$A_{B^4} = \begin{bmatrix} -2 & 1 \\ 1 & 1 \end{bmatrix} \quad , \quad A^{-1}_{B^4} = \begin{bmatrix} -1/3 & 1/3 \\ 1/3 & 2/3 \end{bmatrix} \quad , \quad b_{B^4} = \begin{bmatrix} 1 \\ 14 \end{bmatrix} \quad , \quad \bar{x}^4 = \begin{bmatrix} 13/3 \\ 29/3 \end{bmatrix}$$

$$\bar{y}_{B^4} = \begin{bmatrix}\, 2/3\,,\,7/3\,\end{bmatrix} \quad , \quad \bar{y}_{N^4} = 0 \quad , \quad \bar{y}^4 = \begin{bmatrix}\, 2/3\,,\,0\,,\,0\,,\,7/3\,,\,0\,\end{bmatrix} \geq 0$$

la soluzione duale è ammissibile e quindi l'algoritmo termina: $\bar{x}^4$, $\bar{y}^4$ costituiscono la coppia di soluzioni ottime associate alla base ottima $B = \{\,1\,,\,4\,\}$. La sequenza di soluzioni primali e di direzioni di crescita è mostrata nella figura in alto.

Come sappiamo, la determinazione di una soluzione ottima è solamente uno dei due casi in cui l'algoritmo può terminare; il prossimo esempio illustra l'altro.

Esempio 3.33. Simplesso primale (algebrico), caso illimitato

Si vuole risolva il problema di *PL* dato applicando l'algoritmo del simplesso primale a partire dalla base $B = \{\,3\,,\,4\,\}$. Per ogni iterazione indicheremo la base, la matrice di base e la sua inversa, la coppia di soluzioni di base (discutendo la loro eventuale degenerazione), l'indice uscente, la direzione di crescita, il passo di spostamento e l'indice entrante, giustificando brevemente le risposte.

$$\begin{array}{lrcrcrl} \max & -4x_1 & + & 2x_2 & & & \\ & x_1 & + & x_2 & \leq & 12 & \\ & -x_1 & + & x_2 & \leq & 2 & \\ & x_1 & - & x_2 & \leq & 2 & \\ & x_1 & & & \leq & 6 & \\ & & & x_2 & \leq & 4 & \end{array}$$

it.1) $B = \{\,3\,,\,4\,\}$, $A_B = \begin{bmatrix} 1 & -1 \\ 1 & 0 \end{bmatrix}$, $A_B^{-1} = \begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix}$, $\bar{x} = A_B^{-1}b_B = \begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 6 \end{bmatrix} = \begin{bmatrix} 6 \\ 4 \end{bmatrix}$

$\bar{y}_B = cA_B^{-1} = \begin{bmatrix}\, -4\,,\,2\,\end{bmatrix} \begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix}\, -2\,,\,-2\,\end{bmatrix}$, $\bar{y}_N = 0$, $\bar{y} = \begin{bmatrix}\, 0\,,\,0\,,\,-2\,,\,-2\,,\,0\,\end{bmatrix}$

La base è primale degenere in quanto esiste almeno un indice $i \in N$ tale che $A_i\bar{x} = b_i$ (in particolare, $i = 5$) ma duale non degenere in quanto $\bar{y}_i \neq 0$ per ogni $i \in B$. $h = \min\{\, i \in B : \bar{y}_i < 0 \,\} = \min\{\,3\,,\,4\,\} = 3$, $B(h) = 1$

$\xi = -A_B^{-1}u_{B(h)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, $A_N\xi = \begin{bmatrix} 1 & 1 \\ -1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$, $J = \{\, i \in N : A_i\xi > 0 \,\} = \{\,1\,,\,2\,,\,5\,\}$

$\lambda_i = (\,b_i - A_i\bar{x}\,) \,/\, A_i\xi$, $\lambda_1 = 2$, $\lambda_2 = 4$, $\lambda_5 = 0$, $\bar{\lambda} = \min\{\,\lambda_i : \ i \in J\,\} = 0$

$k = \min\{\, i \in J : \lambda_i = \bar{\lambda} \,\} = 5$ [cambio di base degenere]

it.2) $B = \{\,4\,,\,5\,\}$, $A_B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $A_B^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, $\bar{x} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 6 \\ 4 \end{bmatrix} = \begin{bmatrix} 6 \\ 4 \end{bmatrix}$

$\bar{y}_B = \begin{bmatrix}\, -4\,,\,2\,\end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix}\, -4\,,\,2\,\end{bmatrix}$, $\bar{y} = \begin{bmatrix}\, 0\,,\,0\,,\,0\,,\,-4\,,\,2\,\end{bmatrix}$

[base ovviamente ancora primale degenere e duale non degenere] $h = 4$, $B(h) = 1$

$$\xi = \begin{bmatrix} -1 \\ 0 \end{bmatrix} \ , \ A_N\xi = \begin{bmatrix} 1 & 1 \\ -1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix} \ J = \{\, 2 \,\} \ , \ \bar{\lambda} = \lambda_2 = 4 \ , \ k = 2$$

it.3) $B = \{\, 2 \,, 5 \,\}$, $A_B = \begin{bmatrix} -1 & 1 \\ 0 & 1 \end{bmatrix}$, $A_B^{-1} = \begin{bmatrix} -1 & 1 \\ 0 & 1 \end{bmatrix}$, $\bar{x} = \begin{bmatrix} -1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$

$$\bar{y}_B = \begin{bmatrix} -4 \,, 2 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 4 \,, -2 \end{bmatrix} \ , \ \bar{y} = \begin{bmatrix} 0 \,, 4 \,, 0 \,, 0 \,, -2 \end{bmatrix}$$

[base primale non degenere e duale non degenere] $h = 5$, $B(h) = 2$

$$\xi = \begin{bmatrix} -1 \\ -1 \end{bmatrix} \ , \ A_N\xi = \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \end{bmatrix} = \begin{bmatrix} -2 \\ 0 \\ -1 \end{bmatrix}$$

Poiché $A_N\xi \leq 0$, ossia $J = \emptyset$, alla terza iterazione ξ è una direzione di crescita illimitata. Ciò significa che tutti i punti della forma

$$x(\,\lambda\,) = \bar{x} + \lambda\xi = \begin{bmatrix} 2 \\ 4 \end{bmatrix} + \lambda \begin{bmatrix} -1 \\ -1 \end{bmatrix} = \begin{bmatrix} 2 - \lambda \\ 4 - \lambda \end{bmatrix}$$

sono ammissibili per qualsiasi valore di $\lambda \geq 0$. Inoltre, il loro valore della funzione obiettivo

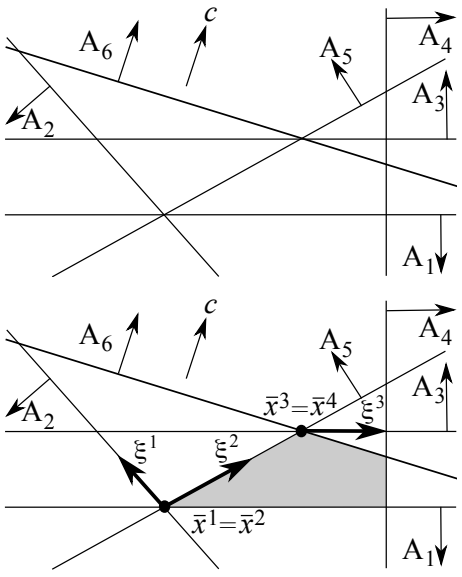
$$cx(\,\lambda\,) = \bar{x} + \lambda\xi = \begin{bmatrix} -4 \,, 2 \end{bmatrix} \begin{bmatrix} 2 - \lambda \\ 4 - \lambda \end{bmatrix} = 2\lambda$$

è chiaramente crescente in λ. Pertanto il problema primale è superiormente illimitato e di conseguenza il problema duale è vuoto.

Negli esempi precedenti abbiamo proceduto algebricamente, come farebbe effettivamente un computer. Grazie a ciò che sappiamo della geometria della *PL*, però, è possibile eseguire l'algoritmo del simplesso (per $n = 2$, o al massimo $n = 3$) avendo solo una rappresentazione grafica del problema, e non la sua effettiva espressione algebrica (i dati A, b e c). Questa esecuzione *geometrica* (o *grafica*) dell'algoritmo è utile per confermare l'intuizione rispetto alle principali proprietà dell'algoritmo, come mostra l'esempio seguente.

Esempio 3.34. Simplesso primale geometrico

Si vuole risolvare per mezzo dell'algoritmo del simplesso primale il problema di *PL* utilizzando come dati esclusivamente la figura qui accanto in cui è rappresentato(si noti che A_4 ed A_6 sono collineari) e la base di partenza $B = \{\, 1 \,, 2 \,\}$. Si noti che A_3 ed A_1 sono collineari, ma di verso opposto, e che separatamente A_6 e c sono collineari. Per ogni iterazione si forniranno la base, la soluzione primale di base x, la direzione di spostamento ξ, il segno delle variabili duali in base, gli indici uscente ed entrante, giustificando le risposte solamente con argomentazioni geometriche. Allo stesso modi si discuterà la degenerazione primale e duale delle soluzioni, ed al termine si discuterà l'unicità della soluzione ottima sia del primale che del duale determinate.



Per descrivere il funzionamento dell'algoritmo faremo principalmente riferimento alla figura qui accanto, nella quale è evidenziata la regione ammissibile e ripoteremo sia le soluzioni di base che le direzioni di crescita individuate nelle varie iterazioni, in quanto possono essere disegnate sul grafico originale. Faremo inoltre riferimento alla Figura 3.10 ove mostreremo il dettaglio dei vettori A_i e c rilevanti in quella iterazione al fine di determinare il segno delle variabili duali e quindi l'indice entrante.

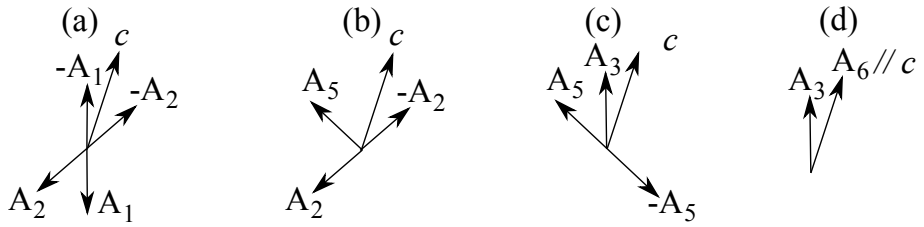


Figura 3.10: Ingrandimento dei dettagli rilevanti nell'esecuzione geometrica

it. 1) $B = \{ 1 , 2 \}$. La soluzione $\bar{x}^1$ è quindi l'unico punto all'intersezione delle rette frontiera dei vincoli 1 e 2, come mostrato in figura. $\bar{y}_1 < 0$ e $\bar{y}_2 < 0$ poiché c appartiene al cono generato da $-A_1$ e $-A_2$, come mostrato in Figura 3.10.(a): quindi, $h = \min\{ i \in B : \bar{y}_i < 0 \} = \min\{ 1 , 2 \} = 1$ [regola anticiclo di Bland]. La direzione ξ^1 è quindi quella che rimane interna alla frontiera del vincolo 2 (che resta in base) allontanandosi, dal lato ammissibile, dalla frontiera del vincolo 1 (che esce di base), come mostra ξ^1 in figura. La base è primale degenere, in quanto $I(\bar{x}^1) = \{ 1 , 2 , 5 \} \supsetneq B$, ed è duale non degenere in quanto c è interno al cono generato da $-A_1$ e $-A_2$, ossia non coincide con nessuno dei due generatori. Il massimo passo lungo la direzione ξ^1 si ottiene in corrispondenza del vincolo 5, attivo ma non in base: quindi, $k = 5$ e si compie un passo degenere.

it. 2) $B = \{ 2 , 5 \}$, $\bar{y}_2 < 0$ e $\bar{y}_5 > 0$ poiché c appartiene al cono generato da $-A_2$ ed A_5, come mostrato in Figura 3.10.(b); quindi, $h = 2$. La base è quindi ancora sia primale degenere (ovviamente, in quanto la soluzione primale di base è la stessa dell'iterazione precedente) che duale non degenere. Il massimo passo lungo la direzione ξ^2 (mostrata in figura) si ottiene in corrispondenza dei due vincoli 3 e 6: quindi, $k = \min\{ 3 , 6 \} = 3$ per la regola anticiclo di Bland (questa volta l'iterazione è non degenere).

it. 3) $B = \{ 3 , 5 \}$, quindi come previsto $\bar{x}^3 \neq \bar{x}^2 = \bar{x}^1$, come mostrato in figura. $\bar{y}_3 > 0$ e $\bar{y}_5 < 0$ poiché c appartiene al cono generato da A_3 e $-A_5$, come mostrato in Figura 3.10.(c); quindi, $h = 5$. La base è di nuovo primale degenere in quanto $I(\bar{x}^3) = \{ 3 , 5 , 6 \} \supsetneq B$, e duale non degenere. Il massimo passo lungo la direzione ξ^3 si ottiene in corrispondenza del solo vincolo 6, attivo ma non in base: si compie quindi di nuovo un passo degenere.

it. 4) $B = \{ 3 , 6 \}$, $\bar{y}_3 > 0$ e $\bar{y}_6 = 0$ poiché c appartiene al cono generato da A_3 ed A_6 "in modo degenere", ossia è collineare ad A_3, come mostrato in Figura 3.10.(d). La base è quindi ancora primale degenere (ovviamente, dato che $\bar{x}^3 = \bar{x}^4$, come mostrato in figura, avendo fatto un passo degenere) ma adesso è anche duale degenere. In ogni caso, la base è sia primale che duale ammissibile, e l'algoritmo termina avendo determinato una soluzione ottima per entrambi i problemi.

Per discutere l'unicità delle soluzioni consideriamo la degenerazione della base ottima ed utilizziamo il teorema degli scarti complementari. Poiché la base è duale degenere, la soluzione ottima del primale non è necessariamente unica. Infatti, tutto il segmento avente per estremi $\bar{x}^4$ ed il vertice (adiacente) corrispondente alla base $B' = \{ 4 , 6 \}$ è ammissibile e perpendicolare a c, quindi tutti i suoi punti hanno il valore ottimo $c\bar{x}^4$ e sono soluzioni ottime alternative. Per via della degenerazione primale della base ottima anche la soluzione duale potrebbe non essere unica, ma in effetti questo non è il caso. Infatti, $c \notin cono(\{ A_3 , A_5 \})$ (si veda ancora la Figura 3.10.(c)), e pertanto l'unico modo per esprimere c come combinazione lineare non negativa di A_3, A_5 ed A_6 è quello di porre $y_3 = y_5 = 0$. Detto in altri termini, consideriamo una qualsiasi soluzione ottima x^* non di base (che sappiamo esistere): in x^* è attivo il solo vincolo A_6, e siccome qualsiasi soluzione ottima duale deve rispettare le condizioni degli scarti complementari con qualsiasi soluzione ottima primale, e quindi in particolare con x^*, deve necessariamente risultare $y_i = 0$ per $i \neq 6$.

3.3.1.1 Individuazione di una base primale ammissibile

La procedura *Simplesso_Primale* non può mai determinare che il problema (P) sia vuoto, in quanto abbiamo supposto di avere in input una soluzione di base primale ammissibile. In generale, mentre determinare una qualsiasi base è semplice, non è altrettanto semplice determinarne una che sia anche primale ammissibile. Per questo si può però utilizzare la stessa procedura *Simplesso_Primale*, applicandola ad un *problema ausiliario* opportunamente costruito. Possiamo innanzitutto assumere di avere a disposizione una base B; questa esiste per l'assunzione che $rango(A) = n$, e può essere facilmente determinata.

Esercizio 3.14. Proporre un algoritmo per determinare una base B.

Siano quindi date B e la corrispondente soluzione di base $\bar{x} = A_B^{-1} b_B$. Siano poi $H = \{ i : A_i \bar{x} \leq b_i \}$ e $J = \{ i : A_i \bar{x} > b_i \}$ rispettivamente l'insieme degli indici dei vincoli rispettati e di quelli violati da $\bar{x}$: è ovvio che $B \subseteq H$ mentre $J \cap B = \emptyset$. Chiaramente, se $J = \emptyset$ allora $\bar{x}$ è una soluzione ammissibile

di base per (P), altrimenti si può costruire il problema ausiliario

$$(PA)\qquad \begin{array}{lrcl} \max & & - u\nu & \\ & A_H x & & \leq b_H \\ & A_J x & - \nu & \leq b_J \\ & & - \nu & \leq 0 \end{array}$$

dove u è il vettore che ha tutte le componenti uguali ad 1. Tale problema ha $x = \bar{x}$, $\bar{\nu} = A_j\bar{x} - b_j$ come soluzione ammissibile di base; infatti, il problema ha $n + |\,J\,|$ variabili ed il vettore $[\,\bar{x}\,,\,\bar{\nu}\,]$ è soluzione del sistema di $n + |\,J\,|$ equazioni in $n + |\,J\,|$ incognite

$$\begin{bmatrix} A_B & 0 \\ A_J & -I \end{bmatrix} \begin{bmatrix} x \\ \nu \end{bmatrix} = \begin{bmatrix} b_B \\ b_J \end{bmatrix} \qquad \text{dove} \qquad \mathcal{A}_{\mathcal{B}} = \begin{bmatrix} A_B & 0 \\ A_J & -I \end{bmatrix}$$

è la *matrice di base ampliata* rispetto alla *base ampliata* $\mathcal{B} = B \cup J$, che è non singolare in quanto A_B è non singolare. Quindi, la procedura *Simplesso_Primale* può essere utilizzata a partire da $\mathcal{B}$ per determinare una soluzione ottima di base $[\,x^*\,,\,\nu^*\,]$ di (PA); si osservi che il problema non può essere illimitato, poiché $-u\nu \leq 0$ per ogni $\nu \geq 0$. Si hanno due casi possibili:

- $\nu^* \neq 0$: in questo caso (P) è vuoto, infatti ad una qualsiasi soluzione ammissibile x di (P) corrisponderebbe la soluzione $[\,x\,,\,0\,]$ di (PA) con valore nullo della funzione obiettivo, e di conseguenza migliore di $[\,x^*\,,\,\nu^*\,]$ che ha un valore negativo della funzione obiettivo;
- $\nu^* = 0$: in questo caso x^* è una soluzione di base ammissibile per (P).

Esempio 3.35. Individuazione di una base primale ammissibile
Si consideri il seguente problema di *PL*

$$(P)\qquad \begin{array}{lrcrcl} \max & x_1 & + & 3x_2 & & \\ & x_1 & + & 2x_2 & \leq & 5 \\ & x_1 & - & x_2 & \leq & 2 \\ & x_1 & + & 3x_2 & \leq & 7 \\ & & - & x_2 & \leq & 0 \end{array}$$

e la base $B = \{\,1\,,\,4\,\}$ non primale ammissibile, in quanto

$$A_B = A_B^{-1} = \begin{bmatrix} 1 & 2 \\ 0 & -1 \end{bmatrix} \quad , \quad \bar{x} = A_B^{-1} b_B = \begin{bmatrix} 1 & 2 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 5 \\ 0 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$$

$$A_N \bar{x} = \begin{bmatrix} 1 & -1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} 5 \\ 0 \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix} \not\leq \begin{bmatrix} 2 \\ 7 \end{bmatrix} = b_N \ .$$

A partire da B possiamo trovare, se esiste, una base primale ammissibile risolvendo il problema primale ausiliario (PA) ed il suo duale (DA); poiché $J = \{\,2\,\}$ e $H = \{\,3\,\}$, essi sono

$$(PA)\qquad \begin{array}{lrcrcrcl} \max & & & & - & \nu & & \\ & x_1 & + & 2x_2 & & & \leq & 5 \\ & x_1 & - & x_2 & - & \nu & \leq & 2 \\ & x_1 & + & 3x_2 & & & \leq & 7 \\ & & - & x_2 & & & \leq & 0 \\ & & & & - & \nu & \leq & 0 \end{array} \qquad (DA)\qquad \begin{array}{lrcrcrcrcrcl} \min & 5y_1 & + & 2y_2 & + & 7y_3 & & & & & \\ & y_1 & + & y_2 & + & y_3 & & & & & = & 0 \\ & 2y_1 & - & y_2 & + & 3y_3 & - & y_4 & & & = & 0 \\ & & & y_2 & & & & & + & y_5 & = & 1 \\ & y_1 & , & y_2 & , & y_3 & , & y_4 & , & y_5 & \geq & 0 \end{array}$$

La base $\mathcal{B}^1 = B \cup J = \{\,1\,,\,2\,,\,4\,\}$ è primale ammissibile per costruzione, possiamo quindi applicare l'algoritmo del simplesso primale a partire da essa.
it. 1) $\mathcal{B}^1 = \{\,1\,,\,2\,,\,4\,\}$, $\mathcal{N}^1 = \{\,3\,,\,5\,\}$:

$$\mathcal{A}_{\mathcal{B}^1} = \begin{bmatrix} 1 & 2 & 0 \\ 1 & -1 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad , \quad \mathcal{A}_{\mathcal{B}^1}^{-1} = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 0 & -1 \\ 1 & -1 & 3 \end{bmatrix} \quad , \quad \begin{bmatrix} \bar{x}^1 \\ \bar{\nu}^1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 0 & -1 \\ 1 & -1 & 3 \end{bmatrix} \begin{bmatrix} 5 \\ 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \\ 3 \end{bmatrix}$$

$$\mathcal{A}_{\mathcal{N}^1} \begin{bmatrix} \bar{x}^1 \\ \bar{\nu}^1 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 5 \\ 0 \\ 3 \end{bmatrix} = \begin{bmatrix} 5 \\ -3 \end{bmatrix} \leq \begin{bmatrix} 7 \\ 0 \end{bmatrix} = b_{\mathcal{N}^1}$$

$$\bar{y}_{\mathcal{B}^1} = \begin{bmatrix} 0, 0, -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 2 \\ 0 & 0 & -1 \\ 1 & -1 & 3 \end{bmatrix} = \begin{bmatrix} -1, 1, -3 \end{bmatrix} \quad , \quad \bar{y}^1 = \begin{bmatrix} -1, 1, 0, -3, 0 \end{bmatrix} \not\geq 0 \ .$$

Pertanto, per la regola anticiclo di Bland $h = 1$ con $\mathcal{B}^1(h) = 1$. Per individuare l'indice entrante calcoliamo

$$\xi^1 = \begin{bmatrix} -1 \\ 0 \\ -1 \end{bmatrix} \quad , \quad \mathcal{A}_{\mathcal{N}^1}\xi^1 = \begin{bmatrix} 1 & 3 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \quad , \quad \bar{\lambda} = \lambda_5 = 3 \quad , \quad k = 5 \ .$$

it. 2) $\mathcal{B}^2 = \{ 5, 2, 4 \}$, $\mathcal{N}^2 = \{ 1, 3 \}$:

$$\mathcal{A}_{\mathcal{B}^2} = \begin{bmatrix} 0 & 0 & -1 \\ 1 & -1 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad , \quad \mathcal{A}_{\mathcal{B}^2}^{-1} = \begin{bmatrix} -1 & 1 & -1 \\ 0 & 0 & -1 \\ -1 & 0 & 0 \end{bmatrix} \quad , \quad \begin{bmatrix} \bar{x}^2 \\ \bar{\nu}^2 \end{bmatrix} = \begin{bmatrix} -1 & 1 & -1 \\ 0 & 0 & -1 \\ -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix}$$

$$\mathcal{A}_{\mathcal{N}^2} \begin{bmatrix} \bar{x}^2 \\ \bar{\nu}^2 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 0 \\ 1 & 3 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix} \leq \begin{bmatrix} 5 \\ 7 \end{bmatrix} = b_{\mathcal{N}^2}$$

$$\bar{y}_{\mathcal{B}^2} = \begin{bmatrix} 0, 0, -1 \end{bmatrix} \begin{bmatrix} -1 & 1 & -1 \\ 0 & 0 & -1 \\ -1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1, 0, 0 \end{bmatrix} \quad , \quad \bar{y}^2 = \begin{bmatrix} 0, 0, 0, 0, 1 \end{bmatrix} \geq 0 \ .$$

Poiché $\bar{y}^2$ è ammissibile per (DA), l'algoritmo termina con le soluzioni ottime $[\, \bar{x}^2, \bar{\nu}^2 \,]$ e $\bar{y}^2$. Si osservi che $\bar{\nu}^2 = 0$, e quindi $\bar{x}^2$ è una soluzione di base primale ammissibile per (P), avente $B = \mathcal{B}^2 \setminus \{ 5 \} = \{ 2, 4 \}$ come base; infatti:

$$A_B = A_B^{-1} = \begin{bmatrix} 1 & -1 \\ 0 & -1 \end{bmatrix} \quad , \quad \bar{x} = A_B^{-1} b_B = \begin{bmatrix} 1 & -1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \end{bmatrix} = \bar{x}^2$$

da cui

$$A_N \bar{x} = \begin{bmatrix} 1 & 2 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix} \leq \begin{bmatrix} 5 \\ 7 \end{bmatrix} = b_N \ .$$

Esercizio 3.15. Risolvere il problema (P) dell'esempio precedente utilizzando l'algoritmo del simplesso primale a partire dalla base $B = \{ 2, 4 \}$.

3.3.1.2 La regola anticiclo di Bland

Riportiamo adesso una dimostrazione formale del fatto che la regola anticiclo di Bland garantisce che una base non possa venir esaminata più di una volta, e quindi la terminazione finita dell'algoritmo del simplesso. Primale.

Teorema 3.19. La procedura *Simplesso_Primale*, che implementa la regola anticiclo di Bland, non esamina la stessa base più di una volta.

Dimostrazione Supponiamo per assurdo che una base B venga visitata due o più volte. Chiamando $B(i)$, $h(i)$ e $k(i)$ rispettivamente la base, l'indice entrante e l'indice uscente all'i-esima iterazione dell'algoritmo, questa ipotesi afferma che esistono due iterazioni v ed ℓ, con $v < \ell$, per cui $B(v) = B(\ell) = B$ e $B(i) \neq B$ per ogni i tale che $v < i < \ell$. Poiché ogni volta che l'algoritmo visita un nuovo vertice il valore della funzione obiettivo cresce, tutte le basi visitate nelle iterazioni del ciclo corrispondono allo stesso vertice $\bar{x}$, e quindi tutti gli indici che entrano ed escono di base appartengono a $I(\bar{x})$. Definiamo

$$r = \max \{ h(i) : v \leq i \leq \ell \} \quad = \quad \max \{ k(i) : v \leq i \leq \ell \} \ ,$$

il massimo degli indici entranti ed uscenti dalla base in tutte le iterazioni comprese tra v ed ℓ, estremi inclusi. Il fatto che il massimo degli indici entranti sia uguale al massimo degli indici uscenti segue immediatamente dal fatto che $B(v) = B(\ell)$: qualsiasi indice che entri in base dopo l'iterazione v deve uscirne prima dell'iterazione l, ed analogamente per gli indici entranti.

Sia adesso p una qualsiasi iterazione $v \leq p \leq \ell$ in cui r è l'indice uscente, ossia $r = h(p)$, e sia q una qualsiasi iterazione $v \leq q \leq \ell$ in cui r è l'indice entrante, ossia $r = k(p)$. Sia adesso $\bar{y} = [\, \bar{y}_{B(p)}, 0 \,]$ la soluzione duale all'iterazione p, in cui r esce di base, e sia $\xi = -A_{B(q)}^{-1} u_{B(h(q))}$ la direzione all'iterazione q, in cui r entra. Per costruzione vale $c\xi > 0$ e $\bar{y}_{B(p)} A_{B(p)} = c$, e quindi

$$c\xi = \bar{y}_{B(p)} A_{B(p)} \xi = \textstyle\sum_{i \in B(p)} \bar{y}_i A_i \xi > 0 \ .$$

Da questo possiamo ottenere una contraddizione mostrando che tutti i termini nella sommatoria della relazione precedente sono minori o uguali a zero. Dobbiamo quindi distinguere tre casi $i = r$, $i > r$ ed $i < r$.

- $i = r$ Poiché r è l'indice uscente all'iterazione p si ha $\bar{y}_r < 0$; dal fatto che r è l'indice entrante all'iterazione q si ha invece che $A_r \xi > 0$, da cui $\bar{y}_r A_r \xi < 0$.
- $i > r$ Per definizione di r, tutti gli indici $i > r$ che stanno in $B(v)$ (se esistono) appartengono a *tutte* le basi in tutte le iterazioni comprese tra v ed ℓ; quindi, in particolare appartengono sia a $B(p)$ che a $B(q)$. Per definizione, nessuno

di questi indici può essere $h(q)$, l'indice uscente all'iterazione q: di conseguenza si ha per ogni indice $i > r$ in $B(p)$ (se ne esistono) che $i \in B(q)$ e $i \neq h(q)$. Per costruzione si ha allora che $A_i\xi = 0$, e quindi $\bar{y}_i A_i\xi = 0$ per ogni $i > r$.

$i < r$ Dalla definizione di r segue immediatamente che

$$r = h(p) = \min\{\, j \in B(p) \,:\, \bar{y}_j < 0 \,\} = k(q) = \min\{\, j \in I(\,\bar{x}\,) \setminus B(q) \,:\, A_j\xi > 0 \,\} \ .$$

Dalla prima relazione discende quindi che $\bar{y}_i \geq 0$ per ogni $i < r$. Siccome qualsiasi indice i in $B(p)$ è l'indice di un vincolo attivo in $\bar{x}$, allora si ha certamente $A_i\xi \leq 0$ per $i < r$: infatti, se $i \in B(q)$ si ha $A_i\xi \leq 0$ per costruzione, mentre per $i \notin B(q)$ si ha che r è il minimo indice j per cui $A_j\xi > 0$, e quindi $A_i\xi \leq 0$ per $i < r$. Da tutto questo segue che $\bar{y}_i A_i\xi \leq 0$ per $i < r$.

Abbiamo quindi ottenuto una contraddizione, il che dimostra che l'algoritmo visita ciascuna base al più una volta. $\diamond$

Oltre a mostrare formalmente la correttezza dell'algoritmo del simplesso primale, il Teorema 3.19 può essere sfruttato per dimostrare un certo numero di risultati teorici.

Esercizio 3.16. Dati $c \in \mathbb{R}^n$ e $A \in \mathbb{R}^{m\times n}$ con $rango(\,A\,) = n$, il che implica $m \geq n$, si dimostri il Lemma di Farkas utilizzando il Teorema 3.19 (suggerimento: si applichi l'algoritmo del Simplesso Primale alle coppia di problemi

$$(P) \quad \max\{\, cx \,:\, Ax \leq 0 \,\} \qquad (D) \quad \min\{\, y0 \,:\, yA = c \,,\ y \geq 0 \,\}$$

notando che (P) non è vuoto, e che se esiste una soluzione $\bar{x}$ tale che $c\bar{x} > 0$ allora (P) è illimitato).

Esercizio 3.17. Dati $c \in \mathbb{R}^n$ e $A \in \mathbb{R}^{m\times n}$ con $rango(\,A\,) < n$, il che accade sicuramente ad esempio se $m < n$, si dimostri il Lemma di Farkas utilizzando il Teorema 3.19 (suggerimento: si sfrutti l'esercizio precedente ed il Teorema 3.5).

Esercizio 3.18. Si dimostrino i Teoremi 3.17 e 3.18.

3.3.2 L'algoritmo del Simplesso Duale

L'algoritmo del simplesso primale, descritto nel precedente paragrafo, genera una sequenza di coppie di soluzioni di base primali ammissibili e duali non ammissibili (tranne al più l'ultima): se la soluzione di base duale risulta ammissibile allora l'algoritmo termina, avendo determinato un vertice ottimo. Risulta naturale pensare ad un algoritmo che generi coppie di soluzioni di base complementari, in cui la soluzione duale sia sempre ammissibile, il quale cerchi invece di raggiungere l'ammissibilità primale. Un algoritmo di questo tipo è l'algoritmo *simplesso duale*, presentato nel seguito.

Il simplesso duale altro non è che il simplesso primale applicato al problema duale della coppia asimmetrica, riscritto in forma primale; esaminando nel dettaglio le operazioni compiute dall'algoritmo se ne ottiene poi una versione semplificata in cui si fa riferimento alle stesse entità (basi e soluzioni di base) che appaiono nel Simplesso Primale. Pur senza entrare nei dettagli, mostriamo nel seguito i principali passaggi che portano alla definizione dell'algoritmo. Il duale della coppia asimmetrica, scritto in forma primale diviene

$$(D') \quad \max\{\, c'y \,:\, A'y \leq b' \,\} \qquad \text{dove} \qquad c' = -b^T \quad , \quad A' = \begin{bmatrix} A^T \\ -A^T \\ -I \end{bmatrix} \quad , \quad b' = \begin{bmatrix} c \\ -c \\ 0 \end{bmatrix} \quad . \tag{3.30}$$

Poiché per ogni $\bar{y}$ ammissibile deve risultare $\bar{y}A^i = c_i$ per ogni $i = 1, \ldots, n$, almeno uno dei due vincoli $A^i y \leq c_i$ e $-A^i y \leq -c_i$ deve appartenere a qualsiasi base "primale" ammissibile B' per il problema; dato che i due vincoli sono linearmente dipendenti, al più uno di essi potrà appartenere ad una base. Quindi, degli m vincoli in una base "primale" ammissibile per (D'), n devono essere scelti in modo da contenere l'intera matrice A^T (eventualmente con un sottoinsieme delle righe cambiate di segno) mentre gli altri $m - n$ devono essere scelti tra gli m vincoli $-Iy \leq 0$. Detto $N \subset \{\, 1 \,, \ldots \,, \, m \,\}$ con $|\, N \,| \, = \, m - n$ l'insieme degli indici che caratterizzano quest'ultimo blocco di vincoli e $B = \{\, 1 \,, \, \ldots \, , \, m \,\} \setminus N$ il suo complementare, ne segue che (modulo un riordinamento di righe e colonne) si può sempre assumere che la matrice di base $A'_{B'}$ corrispondente ad una base B' "primale" ammissibile

per (D') ed il corrispondente vettore $b'_{B'}$ abbiano la forma

$$A'_{B'} = \begin{bmatrix} \pm A_B^T & \pm A_N^T \\ 0 & -I_N \end{bmatrix} \qquad b'_{B'} = \begin{bmatrix} \pm c \\ 0 \end{bmatrix}$$

dove "$\pm$" indica che alcune delle righe potrebbero essere l'opposto della corrispondente colonna di A, e quindi anche il corrispondente elemento di $b'_{B'}$ è l'opposto del coefficiente dei costi. È immediato verificare che $A'_{B'}$ è invertibile se e solo se A_B è invertibile, e che la corrispondente soluzione "primale" di base $\bar{y}$ è esattamente $[\,\bar{y}_B\,,\,\bar{y}_N\,] = [\,cA_B^{-1}\,,\,0\,]$.

Esercizio 3.19. Dimostrare le affermazioni precedenti.

Si noti che la conoscenza della "struttura" della matrice di base permette di calcolarla invertendo solamente A_B, una matrice $n \times n$, invece che l'intera $A'_{B'}$ che è $m \times m$; ciò permette un notevole incremento delle prestazioni nel caso in cui $m \gg n$. Analoghe considerazioni possono essere fatte per gli altri passi dell'algoritmo del Simplesso Primale, che si semplificano e trasformano quando l'algoritmo sia applicato a (D'). Ad esempio, gli unici indici che possono realmente uscire dalla base B' sono quelli relativi ai vincoli $-Iy \leq 0$, ossia quelli in N; se esce il vincolo $A^i y \leq c_i$ deve necessariamente entrare $-A^i y \leq -c_i$, e viceversa, il che corrisponde ad una particolare iterazione degenere che può sempre essere trattata a parte. Pertanto, l'indice uscente da B' esce da N, e quindi entra in B. Infatti, l'algoritmo che presenteremo seleziona prima l'indice entrante in B e poi quello uscente, (apparentemente) al contrario di ciò che fa il simplesso primale.

Esercizio 3.20. Si continui l'analisi dell'applicazione del simplesso primale a (D') scrivendo il duale (P'), calcolando la soluzione "duale" di base $\bar{x}'$ corrispondente ad una base B', verificando cosa significhi che una variabile duale sia negativa, calcolando la corrispondente direzione di crescita ξ ed il massimo passo lungo di essa.

Forniamo adesso una descrizione formale dell'algoritmo.

```
procedure ( B , x̄ , ȳ , stato ) = Simplesso_Duale( A , b , c , B ) {
  for( stato = "" ; ; ) {    // invariante: B duale ammissibile
    x̄ = A_B^{-1} b_B; ȳ = [ ȳ_B , ȳ_N ] = [ cA_B^{-1} , 0 ];
    if( A_N x̄ ≤ b_N ) then { stato = "ottimo"; break; }
    k = min{ i ∈ N : A_i x̄ > b_i }; η_B = A_k A_B^{-1};
    if( η_B ≤ 0 )     then { stato = "P.vuoto"; break; }
    θ̄ = min{ θ_i = ȳ_i/η_i : η_i > 0 , i ∈ B };
    h = min{ i ∈ B : θ_i = θ̄ }; B = B ∪ { k } \ { h };
    }
  }
```

Procedura 3.2: Simplesso Duale

L'algoritmo del simplesso duale riceve in input una base duale ammissibile B, e calcola la corrispondente coppia di soluzioni di base $\bar{x}$ e $\bar{y}$ (con $\bar{y} \geq 0$). Se anche $\bar{x}$ è ammissibile, cioè se $A_N\bar{x} \leq b_N$, l'algoritmo termina, avendo individuato una coppia di soluzioni ottime. Altrimenti, cioè se esiste un indice $k \in N$ tale che $A_k\bar{x} > b_k$, l'algoritmo determina una direzione d di decrescita per $\bar{y}$, in taluni casi ammissibile, definita nel seguente modo:

$$d_i = \begin{cases} -\eta_i & \text{se } i \in B \\ 1 & \text{se } i = k \\ 0 & \text{altrimenti} \end{cases} \tag{3.31}$$

dove $\eta_B = A_k A_B^{-1}$. Per verificare che d sia effettivamente una direzione di decrescita, consideriamo la soluzione duale parametrica $y(\,\theta\,) = \bar{y} + \theta d$ che si ottiene spostandosi da $\bar{y}$ lungo d di un passo $\theta \geq 0$. d è una direzione di decrescita, infatti per ogni $\theta > 0$ si ha

$$y(\theta)b = (\,\bar{y}_B - \theta\eta_B\,)b_B + \theta b_k = \bar{y}_B b_B + \theta(b_k - A_k A_B^{-1} b_B) = \bar{y}b + \theta(\,b_k - A_k\bar{x}\,) < \bar{y}b$$

essendo $A_k\bar{x} > b_k$. Per quanto riguarda l'ammissibilità, è facile verificare che $y(\,\theta\,)A = c$ per qualsiasi scelta di θ, in quanto

$$y(\,\theta\,)A = (\,\bar{y}_B - \theta\eta_B\,)A_B + \theta A_k = \bar{y}_B A_B + \theta(\,A_k - A_k\,) = \bar{y}_B A_B = c \ .$$

Per garantire che d sia ammissibile, va quindi verificato che sia $y(\,\theta\,) = \bar{y} + \theta d \geq 0$ per un opportuno passo di spostamento $\theta > 0$. Osserviamo che gli indici critici sono quelli in B, dovendo essere $(\,\bar{y}_B - \theta\eta_B\,) \geq 0$; infatti, $\bar{y}_k + \theta \geq 0$ per ogni $\theta \geq 0$. In particolare, se $\eta_B \leq 0$, allora l'ammissibilità di $y(\,\theta\,)$ è assicurata per qualsiasi $\theta > 0$; essendo d una direzione di decrescita, segue che (D) è illimitato, e conseguentemente (P) è vuoto. Se invece esiste almeno un indice $i \in B$ per cui $\eta_i > 0$, il massimo passo $\bar{\theta}$ che può essere compiuto lungo d senza perdere l'ammissibilità duale è

$$\bar{\theta} = \min\{\,\bar{y}_i \,/\, \eta_i \ : \ i \in B\,,\, \eta_i > 0\,\} \ .$$

Se risulta $\bar{\theta} > 0$ allora d è effettivamente una direzione di decrescita, altrimenti siamo in un caso, del tutto analogo a quello visto nel primale, di cambiamento di base degenere: infatti ciò può accadere solamente se esiste un indice $i \in B$ tale che $\bar{y}_i = 0$, ossia se B è duale degenere. In ogni caso l'algoritmo fa uscire da B un indice h che determina il passo $\bar{\theta}$ lungo d, ossia l'indice di una componente di $y(\,\theta\,)$ che diverrebbe negativa se fosse effettuato un passo più lungo di $\bar{\theta}$; se $\bar{\theta} > 0$ la nuova base individua un diverso vertice del poliedro delle soluzioni duali, altrimenti la soluzione duale di base non cambia. Analogamente a quanto visto per l'algoritmo del simplesso primale, per evitare cicli si può applicare la regola anticiclo di Bland; nel caso dell'algoritmo del simplesso duale, tale regola si traduce nella selezione del minimo tra gli indici $k \in N$ tali che $A_k\bar{x} > b_k$ e del minimo tra gli indici h che determina il passo di spostamento $\bar{\theta}$ lungo d.

Si noti che, analogamente al caso primale, se il simplesso duale termina dichiarando che il duale è inferiormente illimitato allora ha fornito un *certificato di inammissibilità* del primale, sotto forma del vettore $\eta_B \leq 0$. Infatti, per la definizione di $\bar{x}$ ed η_B

$$\eta_B b_B = A_k A_B^{-1} b_B = A_k\bar{x} > b_k \ .$$

Qualsiasi soluzione ammissibile x del primale deve soddisfare in particolare $A_B x \leq b_B$; moltiplicando entrambi i lati della disequazione per $\eta_B \leq 0$ e ricordando la definizione di η_B si ottiene

$$(\,\eta_B A_B x = A_k x\,) \geq \eta_B b_B \qquad \text{che è incompatibile con} \qquad A_k x \leq b_k$$

in quanto $\eta_B b_B > b_k$. In altri termini, il sistema di disequazioni $A_B x \leq b_B$ è *incompatibile* con $A_k x \leq b_k$: non può esistere nessun x che soddisfa contemporaneamente tutti quei vincoli. L'insieme $B \cup \{\,k\,\}$ caratterizza quindi un *insieme inconsistente* (IS), dal quale si può poi cercare di determinare un insieme inconsistente *irriducibile* (IIS), ossia tale che la rimozione di qualsiasi vincolo lo rende consistente. Si noti che $B \cup \{\,k\,\}$ può non essere irriducibile; ciò ad esempio accade, come è facile verificare, se $\eta_i = 0$ per qualche $i \in B$. Gli IIS sono utili ad esempio per "debuggare" i modelli: se ci si aspetta che il modello abbia soluzione, perché la realtà modellata notoriamente ne ha, e questo non risulta vero, l'IIS indica "di quali vincoli è la colpa" e quindi può aiutare a determinare dove sia il problema. Gli (I)IS possono comunque avere anche altri usi.

Illustriamo adesso il funzionamento dell'algoritmo mediante alcuni esempi.

Esempio 3.36. Applicazione del simplesso duale per via algebrica
Si consideri la coppia asimmetrica di problemi duali

$$(P) \quad \begin{array}{lrcrcl} \max & x_1 & & & & \\ & x_1 & + & 2x_2 & \leq & 6 \\ & x_1 & - & 2x_2 & \leq & 6 \\ & 2x_1 & + & x_2 & \leq & 4 \\ & 2x_1 & - & x_2 & \leq & 4 \\ & -x_1 & & & \leq & 0 \end{array} \qquad (D) \quad \begin{array}{lrcrcrcrcrcl} \min & 6y_1 & + & 6y_2 & + & 4y_3 & + & 4y_4 & & & \\ & y_1 & + & y_2 & + & 2y_3 & + & 2y_4 & - & y_5 & = 1 \\ & 2y_1 & - & 2y_2 & + & y_3 & - & y_4 & & & = 0 \\ & y_1 & , & y_2 & , & y_3 & , & y_4 & & y_5 & \geq 0 \end{array}$$

e la base $B^1 = \{\,1\,,\,2\,\}$. Essendo

$$A_{B^1} = \begin{bmatrix} 1 & 2 \\ 1 & -2 \end{bmatrix} \quad , \quad A_{B^1}^{-1} = \begin{bmatrix} 1/2 & 1/2 \\ 1/4 & -1/4 \end{bmatrix} \quad , \quad b_{B^1} = \begin{bmatrix} 6 \\ 6 \end{bmatrix}$$

la corrispondente soluzione di base primale è

$$\bar{x}^1 = A_{B_1}^{-1} b_{B_1} = \begin{bmatrix} 1/2 & 1/2 \\ 1/4 & -1/4 \end{bmatrix} \begin{bmatrix} 6 \\ 6 \end{bmatrix} = \begin{bmatrix} 6 \\ 0 \end{bmatrix} \ ,$$

che risulta essere non ammissibile; infatti

$$A_{N^1} \bar{x}^1 = \begin{bmatrix} 2 & 1 \\ 2 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 6 \\ 6 \end{bmatrix} = \begin{bmatrix} 12 \\ 12 \\ -6 \end{bmatrix} \not\leq \begin{bmatrix} 4 \\ 4 \\ 0 \end{bmatrix} = b_{N^1} \ .$$

Viceversa, c può essere espresso come combinazione lineare a coefficienti non negativi di A_1 ed A_2, e quindi la base B^1 è duale ammissibile; infatti

$$\bar{y}_{B^1} = c A_{B^1}^{-1} = \begin{bmatrix} 1\,,\,0 \end{bmatrix} \begin{bmatrix} 1/2 & 1/2 \\ 1/4 & -1/4 \end{bmatrix} = \begin{bmatrix} 1/2\,,\,1/2 \end{bmatrix} \quad , \quad \bar{y}^1 = \begin{bmatrix} 1/2\,,\,1/2\,,\,0\,,\,0\,,\,0 \end{bmatrix} \geq 0 \ .$$

Possiamo quindi applicare l'algoritmo del simplesso duale a partire dalla base B^1. L'indice entrante è

$$k = \min \{\, i \in N^1 \,:\, A_i \bar{x}^1 > b_i \,\} = \min \{\,3\,,\,4\,\} = 3$$

e quindi

$$\eta_{B^1} = A_3 A_{B^1}^{-1} = \begin{bmatrix} 2\,,\,1 \end{bmatrix} \begin{bmatrix} 1/2 & 1/2 \\ 1/4 & -1/4 \end{bmatrix} = \begin{bmatrix} 5/4\,,\,3/4 \end{bmatrix}$$

$$\bar{\theta} = \min \left\{ \frac{\bar{y}_1^1}{\eta_1^1} \,,\, \frac{\bar{y}_2^1}{\eta_2^1} \right\} = \min \left\{ \frac{1/2}{5/4} \,,\, \frac{1/2}{3/4} \right\} = \min \left\{ \frac{2}{5} \,,\, \frac{2}{3} \right\} = \frac{2}{5} \ ;$$

pertanto l'indice uscente è $h = 1$. La nuova base è $B^2 = B^1 \setminus \{\,h\,\} \cup \{\,k\,\} = \{\,1\,,\,2\,\} \setminus \{\,1\,\} \cup \{\,3\,\} = \{\,3\,,\,2\,\}$. Effettuiamo quindi la seconda iterazione, dove

$$A_{B^2} = \begin{bmatrix} 2 & 1 \\ 1 & -2 \end{bmatrix} \quad , \quad A_{B^2}^{-1} = \begin{bmatrix} 2/5 & 1/5 \\ 1/5 & -2/5 \end{bmatrix} \quad , \quad b_{B^2} = \begin{bmatrix} 4 \\ 6 \end{bmatrix} \quad , \quad \bar{x}^2 = \begin{bmatrix} 2/5 & 1/5 \\ 1/5 & -2/5 \end{bmatrix} \begin{bmatrix} 4 \\ 6 \end{bmatrix} = \begin{bmatrix} 14/5 \\ -8/5 \end{bmatrix}$$

$$\bar{y}_{B^2} = \begin{bmatrix} 1\,,\,0 \end{bmatrix} \begin{bmatrix} 2/5 & 1/5 \\ 1/5 & -2/5 \end{bmatrix} = \begin{bmatrix} 2/5\,,\,1/5 \end{bmatrix} \quad , \quad \bar{y}^2 = \begin{bmatrix} 0\,,\,1/5\,,\,2/5\,,\,0\,,\,0 \end{bmatrix} \ .$$

La soluzione primale $\bar{x}^2$ non è ammissibile in quanto

$$A_{N^2} \bar{x}^2 = \begin{bmatrix} 2/5 & 1/5 \\ 1/5 & -2/5 \end{bmatrix} \begin{bmatrix} 14/5 \\ -8/5 \end{bmatrix} = \begin{bmatrix} -2/5 \\ 36/5 \\ -14/5 \end{bmatrix} \not\leq \begin{bmatrix} 6 \\ 4 \\ 0 \end{bmatrix} = b_{N^2}$$

e quindi l'indice entrante è $k = 4$ (quello dell'unico vincolo violato); pertanto

$$\eta_{B^2} = \begin{bmatrix} 2\,,\,-1 \end{bmatrix} \begin{bmatrix} 2/5 & 1/5 \\ 1/5 & -2/5 \end{bmatrix} = \begin{bmatrix} 3/5\,,\,4/5 \end{bmatrix}$$

$$\bar{\theta} = \min \left\{ \frac{\bar{y}_3^2}{\eta_3^2} \,,\, \frac{\bar{y}_2^2}{\eta_2^2} \right\} = \min \left\{ \frac{2/5}{3/5} \,,\, \frac{1/5}{4/5} \right\} = \min \left\{ \frac{2}{3} \,,\, \frac{1}{4} \right\} = \frac{1}{4} \ ,$$

da cui l'indice uscente è $h = 2$. La nuova base è perciò $B_3 = \{\,3\,,\,2\,\} \setminus \{\,2\,\} \cup \{\,4\,\} = \{\,3\,,\,4\,\}$. Nella terza iterazione

$$A_{B^3} = \begin{bmatrix} 2 & 1 \\ 2 & -1 \end{bmatrix} \quad , \quad A_{B^3}^{-1} = \begin{bmatrix} 1/4 & 1/4 \\ 1/2 & -1/2 \end{bmatrix} \quad , \quad b_{B^3} = \begin{bmatrix} 4 \\ 4 \end{bmatrix} \quad , \quad \bar{x}^3 = \begin{bmatrix} 1/4 & 1/4 \\ 1/2 & -1/2 \end{bmatrix} \begin{bmatrix} 4 \\ 4 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$$

$$\bar{y}_{B^3} = \begin{bmatrix} 1\,,\,0 \end{bmatrix} \begin{bmatrix} 1/4 & 1/4 \\ 1/2 & -1/2 \end{bmatrix} = \begin{bmatrix} 1/4\,,\,1/4 \end{bmatrix} \quad , \quad \bar{y}^3 = \begin{bmatrix} 0\,,\,0\,,\,1/4\,,\,1/4\,,\,0 \end{bmatrix} \ .$$

Stavolta la soluzione primale $\bar{x}^3$ è ammissibile, in quanto

$$A_{N^3} \bar{x}^3 = \begin{bmatrix} 1 & 2 \\ 1 & -2 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \\ -2 \end{bmatrix} \leq \begin{bmatrix} 6 \\ 6 \\ 0 \end{bmatrix} = b_{N^3}$$

e quindi l'algoritmo termina: $(\,\bar{x}^3\,,\,\bar{y}^3\,)$ è la coppia di soluzioni ottime associate alla base ottima $B = \{\,3\,,\,4\,\}$.

In modo simmetrico al simplesso primale, il simplesso duale non può mai dimostrare che il problema

primale è illimitato (il duale dovrebbe essere vuoto, ma l'algoritmo necessita di una base ammissibile) ma può dimostrare che è vuoto (il duale è illimitato).

Esempio 3.37. Simplesso duale, il caso primale vuoto

Si vuole risolvere il problema di *PL* dato applicando l'algoritmo del simplesso duale a partire dalla base $B = \{ 1 , 4 \}$. Per ogni iterazione si indicheranno: la base, la matrice di base e la sua inversa, la coppia di soluzioni di base, l'indice entrante, il vettore η_B, il passo di spostamento e l'indice uscente. Solo nella prima iterazione si riporteranno le formule utilizzate per calcolarle.

$$\begin{array}{lrcrcrcr} \max & x_1 & + & 2x_2 & & & \\ & & & x_2 & \leq & 4 \\ & x_1 & - & 2x_2 & \leq & 1 \\ & 2x_1 & + & x_2 & \leq & 4 \\ & x_1 & & & \leq & 2 \\ & -x_1 & + & x_2 & \leq & -2 \end{array}$$

it. 1) $B = \{ 1 , 4 \}$: $A_B = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$, $A_B^{-1} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ $\bar{x} = A_B^{-1} b_B = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 4 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$

$\bar{y}_B = c A_B^{-1} = \begin{bmatrix} 1 , 2 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 2 , 1 \end{bmatrix}$, $\bar{y}_N = 0$, $\bar{y} = \begin{bmatrix} 2 , 0 , 0 , 1 , 0 \end{bmatrix}$

$A_N \bar{x} = \begin{bmatrix} 1 & -2 \\ 2 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \end{bmatrix} = \begin{bmatrix} -6 \\ 8 \\ 2 \end{bmatrix} \not\leq b_N = \begin{bmatrix} 1 \\ 4 \\ -2 \end{bmatrix}$

$k = \min \{ i \in N : A_i \bar{x} > b_i \} = \min \{ 3 , 5 \} = 3$ [regola anticiclo di Bland]

$\eta_B = A_k A_B^{-1} = \begin{bmatrix} 2 , 1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 , 2 \end{bmatrix}$

$\bar{\theta} = \min \{ \bar{y}_i / \eta_i : i \in B , \eta_i > 0 \} = \min \{ \bar{y}_1 / \eta_1 , \bar{y}_4 / \eta_4 \} = \min \{ 2 / 1 , 1 / 2 \} = 1/2$

$h = \min \{ i \in B : \eta_i > 0 , \bar{\theta} = \bar{y}_i / \eta_i \} = \min \{ 4 \} = 4$

$B = B \setminus \{ u \} \cup \{ k \} = \{ 1 , 4 \} \setminus \{ 4 \} \cup \{ 3 \} = \{ 1 , 3 \}$

it. 2) $B = \{ 1 , 3 \}$: $A_B = \begin{bmatrix} 0 & 1 \\ 2 & 1 \end{bmatrix}$, $A_B^{-1} = \begin{bmatrix} -1/2 & 1/2 \\ 1 & 0 \end{bmatrix}$, $\bar{x} = \begin{bmatrix} -1/2 & 1/2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 4 \\ 4 \end{bmatrix} = \begin{bmatrix} 0 \\ 4 \end{bmatrix}$

$\bar{y}_B = \begin{bmatrix} 1 , 2 \end{bmatrix} \begin{bmatrix} -1/2 & 1/2 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 3/2 , 1/2 \end{bmatrix}$, $\bar{y}_N = 0$, $\bar{y} = \begin{bmatrix} 3/2 , 0 , 1/2 , 0 , 0 \end{bmatrix}$

$A_N \bar{x} = \begin{bmatrix} 1 & -2 \\ 1 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 4 \end{bmatrix} = \begin{bmatrix} -8 \\ 0 \\ 4 \end{bmatrix} \not\leq b_N = \begin{bmatrix} 1 \\ 2 \\ -2 \end{bmatrix}$, $k = 5$ (unico vincolo violato)

$\eta_B = \begin{bmatrix} -1 , 1 \end{bmatrix} \begin{bmatrix} -1/2 & 1/2 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 3/2 , -1/2 \end{bmatrix}$, $h = 1$ in quanto η_1 è l'unico > 0

$\bar{\theta} = \bar{y}_1 / \eta_1 = (3/2) / (3/2) = 1$ (ma $\bar{\theta}$ è nei fatti ininfluente)

it. 3) $B = \{ 3 , 5 \}$: $A_B = \begin{bmatrix} 2 & 1 \\ -1 & 1 \end{bmatrix}$, $A_B^{-1} = \begin{bmatrix} 1/3 & -1/3 \\ 1/3 & 2/3 \end{bmatrix}$, $\bar{x} = \begin{bmatrix} 1/3 & -1/3 \\ 1/3 & 2/3 \end{bmatrix} \begin{bmatrix} 4 \\ -2 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$

$\bar{y}_B = \begin{bmatrix} 1 , 2 \end{bmatrix} \begin{bmatrix} 1/3 & -1/3 \\ 1/3 & 2/3 \end{bmatrix} = \begin{bmatrix} 1 , 1 \end{bmatrix}$, $\bar{y}_N = 0$, $\bar{y} = \begin{bmatrix} 0 , 0 , 1 , 0 , 1 \end{bmatrix}$

$A_N \bar{x} = \begin{bmatrix} 0 & 1 \\ 1 & -2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ 2 \end{bmatrix}] \not\leq b_N = \begin{bmatrix} 4 \\ 1 \\ 2 \end{bmatrix}$, $k = 2$ (unico vincolo violato)

$\eta_B = \begin{bmatrix} 1 , -2 \end{bmatrix} \begin{bmatrix} 1/3 & -1/3 \\ 1/3 & 2/3 \end{bmatrix} = \begin{bmatrix} -1/3 , -5/3 \end{bmatrix} \leq 0$

Poiché $\eta_B \leq 0$ si ha che il problema duale è inferiormente illimitato, e di conseguenza il problema primale è vuoto. Ciò può essere mostrato in diversi modi. Il fatto che il duale sia inferiormente illimitato discende dal fatto che la direzione

$$d = \begin{bmatrix} -\eta_B , 1 , 0 \end{bmatrix} = \begin{bmatrix} 0 , 1 , 1/3 , 0 , 5/3 \end{bmatrix}$$

è una direzione ammissibile illimitata per il problema duale, ossia rispetta la condizioni $dA = 0$ (come è possibile verificare) e $d \geq 0$ (come è ovvio), ed in più è di decrescita in quanto $db = 1 + 4/3 - 10/3 = -1 < 0$. Ciò significa che tutte le soluzioni delle forma

$$y(\theta) = \bar{y} + \theta d = \begin{bmatrix} 0 , 0 , 1 , 0 , 1 \end{bmatrix} + \theta \begin{bmatrix} 0 , 1 , 1/3 , 0 , 5/3 \end{bmatrix} = \begin{bmatrix} 0 , \theta , 1 + \theta/3 , 0 , 1 + 5\theta/3 \end{bmatrix}$$

sono ammissibili per qualsiasi $\theta \geq 0$, ed il loro costo

$$y(\theta) b = \theta + 4(1 + \theta/3) - 2(1 + 5\theta/3) = 2 - \theta$$

è decrescente in θ, pertanto il duale è inferiormente illimitato ed il primale è vuoto.

Alternativamente, la base dell'ultima iterazione $B = \{ 3 , 5 \}$ più l'indice $k = 2$ identificato rappresentano un *Inconsistent Subsystem* del problema primale: infatti, utilizzando i moltiplicatori η_B si ottiene

$$\eta_B A_B x = A_k A_B^{-1} A_B x = A_k x = x_1 - 2x_2 \geq \eta_B b_B = \begin{bmatrix} -1/3 , -5/3 \end{bmatrix} \begin{bmatrix} 4 \\ -2 \end{bmatrix} = 2$$

(il "$\geq$" al penultimo passaggio deriva dal fatto che $\eta_B \leq 0$), che è chiaramente incompatibile con il secondo vincolo.

In modo analogo al simplesso primale è possibile uno svolgimento geometrico, o grafico, del simplesso duale per problemi di piccole dimensioni. Questo è leggermente meno ovvia che nel caso del simplesso primale, però, perché tipicamente si riesce a visualizzare lo spazio del primale, ma non quello del duale che è quello "naturale" per l'algoritmo. Ciò necessita di alcuni accorimenti leggermente più complessi. In particolare, data una base $B = \{ i , j \}$ ed un indice entrante k, è necessario stimare i rapporti $\bar{y}_i / \eta_i$ e $\bar{y}_j / \eta_j$ per determinare quale di essi sia il minore e quindi quale sia l'indice uscente tra i e j. Questo non è sempre necessario: infatti, se uno solo tra η_i ed η_j è positivo sicuramente l'indice corrispondente uscirà di base, mentre se sono entrambi non positivi l'algoritmo termina.

Il "caso difficile" è quindi quello in cui sono entrambi positivi. Come mostrato in Figura 3.11, geometricamente sarebbe possibile stimare le proiezioni di c ed A_k su A_i ed A_j, che corrispondono rispettivamente ai $\bar{y}$ ed agli η, e stimare quale dei due rapporti sia minore. Questo non è agevole, pertanto si possono utilizzare "scorciatoie". La prima è il caso in cui c ed A_k siano collineari (con lo stesso verso), ossia $c = \alpha A_k$ per $\alpha > 0$: in questo caso ovviamente $\bar{y}_i = \alpha \eta_i$ e $\bar{y}_j = \alpha \eta_j$, pertanto $\bar{y}_i / \eta_i = \bar{y}_j / \eta_j = \alpha$, e quindi per la regola anticiclo di Bland uscirà di base quello di indice minore tra i e j. Se ciò non capita, allora c può appartenere ad uno solo tra $cono(\{ A_i , A_k \})$ e $cono(\{ A_j , A_k \})$ (si veda ancoa la Figura 3.11): poiché il simplesso duale visita solamente duali ammissibili, deve necessariamente uscire di base l'indice $p \in \{ i , j \}$ tale che $c \notin cono(\{ A_p , A_k \})$.

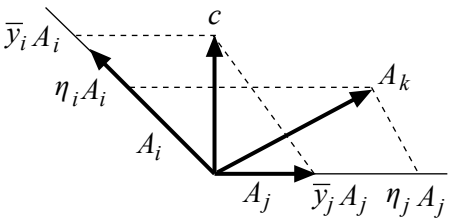
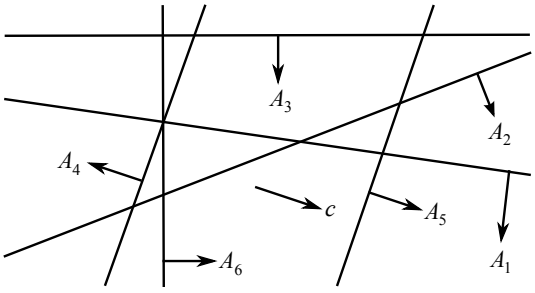


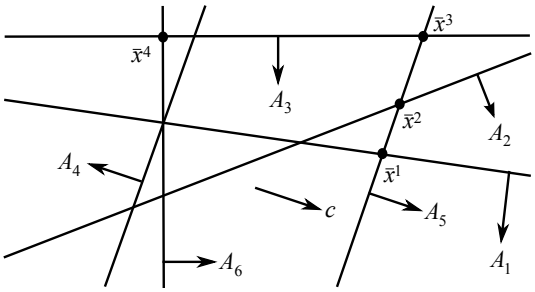
Figura 3.11: Il "caso difficile" del calcolo di $\bar{\theta}$

Esempio 3.38. Esecuzione grafica del simplesso duale

Si vuole risolvere graficamente il problema di *PL* in figura qui accanto, utilizzando l'algoritmo del simplesso duale a partire dalla base $B = \{ 1 , 5 \}$; si noti che c, A_4 ed A_5 sono collineari. Per ogni iterazione si indicheranno: la base, la soluzione primale di base, l'indice entrante k, i segni delle componenti dei vettori $\bar{y}_B$ e η_B e l'indice uscente h. Solo alla prima iterazione si giustificheranno in modo dettagliato le risposte. Si discuterà inoltre l'eventuale degenerazione primale e duale delle soluzioni di base determinate.



Per descrivere il funzionamento dell'algoritmo faremo riferimento alla figura qui accanto, nella quale *non* è evidenziata la regione ammissibile: sappiamo che è vuota, ed in ogni caso ha rilevanza per il simplesso duale solo all'ultima iterazione. Infatti, in questo caso ripoteremo in figura solamente le soluzioni di base primali: quelle duali, e le direzioni di crescita individuate nelle varie iterazioni vivono nello spazio duale $\mathbb{R}^6$ che non può essere facilmente rappresentato. Faremo inoltre riferimento alla Figura 3.12 ove mostreremo il dettaglio dei vettori A_i rilevanti in quella iterazione (quelli in B, ed A_k) e di c al fine di determinare il segno delle variabili duali e quindi l'indice entrante.



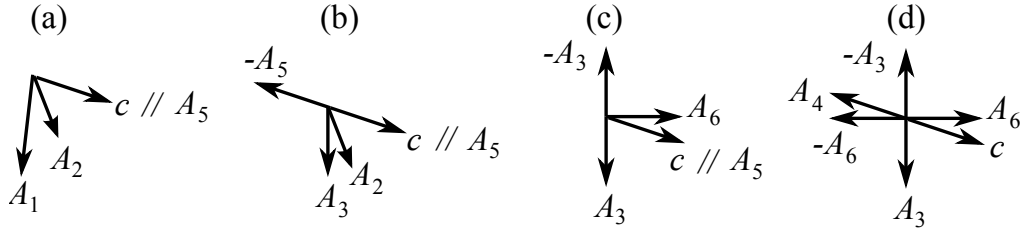


Figura 3.12: Ingrandimento dei dettagli rilevanti nell'esecuzione geometrica

it. 1): $B = \{1, 5\}$. La soluzione primale di base $\bar{x}^1$ è l'unico punto nell'intersezione tra le frontiere (rette) dei vincoli 1 e 5, come mostrato in figura: quindi viola i vincoli 2, 3 e 6, e pertanto $k = \min\{2, 3, 6\} = 2$ (minimo indice di un vincolo violato) per la regola anticiclo di Bland. $\bar{y}_1 = 0$ e $\bar{y}_5 > 0$ in quanto c è collineare con A_5; quindi la base è duale degenere, ma è primale non degenere poiché $B = I(\bar{x}^1)$. Poiché $A_2 \in cono(\{A_1, A_5\})$, come mostrato in Figura 3.12.(a), risultano $\eta_1 > 0$ e $\eta_5 > 0$: siamo quindi nel "caso difficile" entrambi gli η sono positivi, e quindi dobbiamo trovare il modo per determinare l'indice uscente. Questo può essere ottenuto con diversi ragionamenti. Ad esempio, poichè $c \in cono(\{A_2, A_5\})$ ma $c \notin cono(\{A_1, A_2\})$, deve necessariamente risultare $h = 1$. Un modo diverso di ottenere lo stesso risultato è notare che, poiché $\bar{y}_1 = 0$, $\bar{y}_1/\eta_1 = 0$ mentre $\bar{y}_5/\eta_5 > 0$.

it. 2): $B = \{2, 5\}$. La soluzione primale di base $\bar{x}^2$ viola i vincoli 3 e 6, pertanto $k = \min\{3, 6\} = 3$ per la regola anticiclo di Bland. $\bar{y}_2 = 0$ e $\bar{y}_5 > 0$ in quanto c è collineare con A_5; quindi la base è ancora duale degenere, e rimane primale non degenere poiché $B = I(\bar{x}^2)$. Poiché $A_3 \in cono(\{A_2, -A_5\})$, come mostrato in Figura 3.12.(b), risultano $\eta_2 > 0$ e $\eta_5 < 0$; pertanto $h = 2$ perché possono uscire di base solamente indici $i \in B$ con $\eta_i > 0$.

it. 3): $B = \{3, 5\}$. La soluzione primale di base $\bar{x}^3$ viola il solo vincolo 6, pertanto $k = 1$. $\bar{y}_3 > 0$ e $\bar{y}_5 > 0$ in quanto c è interno al $cono(\{A_3, A_5\})$; quindi la base è duale non degenere, ed è anche primale non degenere poiché $B = I(\bar{x}^3)$. Poiché $A_6 \in cono(\{-A_3, A_5\})$, come mostrato in Figura 3.12.(c), risultano $\eta_3 < 0$, $\eta_5 > 0$; pertanto $h = 5$.

it. 4): $B = \{3, 6\}$. La soluzione primale di base $\bar{x}^3$ viola il solo vincolo 4, pertanto $k = 4$. $\bar{y}_3 > 0$ e $\bar{y}_6 > 0$ in quanto c è interno al cono generato da A_3 e A_6; quindi la base è ancora duale e primale non degenere, poiché ancora $B = I(\bar{x}^4)$. Poiché $A_4 \in cono(\{-A_3, -A_6\})$, come mostrato in Figura 3.12.(d), risultano $\eta_3 < 0$, $\eta_6 < 0$; pertanto l'algoritmo termina avendo determinato che il duale è inferiormente illimitato ed il primale è vuoto.

È facile constatare che il risultato dell'algoritmo è corretto. Sarebbe possibile, ma complesso, mostrare graficamente che il duale è illimitato se si conoscessero i dati A, b e c del problema: si potrebbe rappresentare il sottospazio di $\mathbb{R}^6$ corrispondente a fissare le variabili y_1, y_2 ed y_5 a zero, in quanto esse sono nulle in tutti i punti della forma $y(\theta) = \bar{y} + \theta d$ (cf. (3.31)), e mostrare nel rimanente spazio $\mathbb{R}^3$ corrispondente alle variabili y_3, y_4 ed y_6 che il poliedro è illimitato e contiene una direzione di crescita. È molto più agevole mostrare invece che il poliedro del primale è vuoto. Infatti, i vincoli 1, 4 e 6 da soli definirebbero come regione ammissibile l'unico punto in cui le loro frontiere si incontrano, ma tale punto non soddisfa il vincolo 3. Anche considerando solamente i vincoli 4 e 6, la regione ammissibile deve contenere lo stretto triangolo che essi delimitano, ma tutti i punti di quel triangolo sono inammissibili per il vincolo 3. Quindi, $B \cup \{k\} = \{3, 6\} \cup \{4\} = \{3, 4, 6\}$ è un *Inconsistent Subsystem* (IS) per il problema, ed è anche *Irreducible* (IIS) in quanto ovviamente eliminando uno qualsiasi dei tre vincoli si ottiene una regione ammissibile non vuota.

Esercizio 3.21. Si risolva geometricamente il problema dell'Esempio 3.36 mediante l'algoritmo simplesso duale, partendo dalla base $B = \{1, 2\}$; si confrontino i risultati ottenuti con quelli riportati nell'esempio.

3.3.2.1 Individuazione di una base duale ammissibile

Per completare lo studio del Simplesso Duale resta da considerare il caso in cui non è nota una soluzione di base duale ammissibile da cui partire. Nel problema duale (D) possiamo supporre senza perdita di generalità che sia $c \geq 0$: infatti, essendo i vincoli del problema in forma di uguaglianza, è possibile moltiplicare per -1 ogni vincolo per cui sia $c_i < 0$ (ciò corrisponde alla sostituzione di variabile $x_i = -x_i$ nel primale). Introduciamo quindi il *duale ausiliario* (ed il suo duale)

$$(DA)\ \min\{yb + wM : yA + w = c\ ,\ y \geq 0\ ,\ w \geq 0\} \qquad (PA)\ \max\{cx : Ax \leq b\ ,\ x \leq M\}$$

dove M è un vettore con componenti positive ed "opportunamente grandi" (il significato esatto di tale espressione risulterà chiaro nel seguito). Per (DA) è nota la base "artificiale" $B = \{m+1, \ldots, m+n\}$ contenente gli indici relativi alle variabili "artificiali" w, ossia le variabili duali dei vincoli "artificiali" $x \leq M$, con la corrispondente matrice di base $A_B = I$: è immediato verificare che la soluzione duale di base è $[\bar{y}, \bar{w}] = [0, c]$, che quindi è ammissibile ($c \geq 0$). Da tale base si può quindi partire con l'algoritmo del simplesso duale. Si può dimostrare che, scegliendo le componenti di M sufficientemente grandi, (DA) è equivalente a D; più precisamente, se (D) è inferiormente illimitato allora anche (DA)

lo è, indipendentemente dal valore scelto per le componenti di M, mentre se (D) non è inferiormente illimitato allora esiste un valore per tali componenti tale che, data una qualsiasi soluzione ottima di base $[\,y^*\,,\,w^*\,]$ per (DA), se $w^* = 0$ allora y^* è una soluzione ottima per (D), altrimenti (cioè se $w^* \neq 0$) allora (D) non possiede soluzioni ammissibili. È possibile però che (DA) risulti inferiormente non limitato anche nel caso in cui (D) sia vuoto; in tal caso almeno una variabile ausiliaria avrà valore positivo. Poiché determinare a priori un valore opportuno di M non è in generale possibile, si procede usualmente fissando un certo valore e risolvendo il problema per quel fissato valore; una volta determinata una base ottima, oppure che il problema è inferiormente illimitato, si usano le tecniche di analisi parametrica rispetto ai cambiamenti del vettore b (che in questo caso contiene anche M), descritte nel §3.3.4, per determinare se il valore di M è corretto, oppure deve essere aumentato, oppure a nessun valore finito di M corrisponde una soluzione duale ammissibile (il che dimostra che (D) è vuoto). È in effetti possibile implementare l'algoritmo del simplesso duale in modo tale che risolva (DA) per un opportuno valore delle componenti di M, senza che esso venga scelto a priori; questi interessanti ma non ovvi dettagli sono lasciati per esercizio al lettore interessato..

Esempio 3.39. Individuazione di una base duale ammissibile

Si consideri la coppia di problemi duali:

$$(P)\quad \begin{array}{lrcrcl} \max & 4x_1 & + & 2x_2 & & \\ & -x_1 & + & 4x_2 & \leq & 2 \\ & x_1 & - & 2x_2 & \leq & -3 \\ & -x_1 & + & x_2 & \leq & -1 \end{array} \qquad (D)\quad \begin{array}{lrcrcrcl} \min & 2y_1 & - & 3y_2 & - & y_3 & & \\ & -y_1 & + & y_2 & - & y_3 & = & 4 \\ & 4y_1 & - & 2y_2 & + & y_3 & = & 2 \\ & y_1 & , & y_2 & , & y_3 & \geq & 0 \end{array}$$

Non avendo né una base primale né una base duale ammissibile di partenza, risolviamo il problema Duale Ausiliario (DA) ed il suo duale (PA), ponendo $M = [\,20\,,\,20\,]$:

$$(DA)\quad \begin{array}{lrcrcrcrcrcl} \min & 2y_1 & - & 3y_2 & - & y_3 & + & 20w_1 & + & 20w_2 & & \\ & -y_1 & + & y_2 & - & y_3 & + & w_1 & & & = & 4 \\ & 4y_1 & - & 2y_2 & + & y_3 & & & + & w_2 & = & 2 \\ & y_1 & , & y_2 & , & y_3 & , & w_1 & , & w_2 & \geq & 0 \end{array} \qquad (PA)\quad \begin{array}{lrcrcl} \max & 4x_1 & + & 2x_2 & & \\ & -x_1 & + & 4x_2 & \leq & 2 \\ & x_1 & - & 2x_2 & \leq & -3 \\ & -x_1 & + & x_2 & \leq & -1 \\ & x_1 & & & \leq & 20 \\ & & & x_2 & \leq & 20 \end{array}$$

La base di partenza "artificiale" $B^1 = \{\,4\,,\,5\,\}$ è quella relativa alle variabili ausiliarie w_1 e w_2.

it. 1): $B^1 = \{\,4\,,\,5\,\}$, $A_{B^1} = A_{B^1}^{-1} = I$, $\bar{x}^1 = A_{B^1}^{-1} b_{B^1} = b_{B^1} = [\,20\,,\,20\,]$ $\bar{y}^1 = [\,0\,,\,0\,,\,0\,]$, $\bar{w}^1 = cA_{B^1}^{-1} = [\,4\,,\,2\,]$

$$A_{N^1}\bar{x}^1 = \begin{bmatrix} -1 & 4 \\ 1 & -2 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 20 \\ 20 \end{bmatrix} = \begin{bmatrix} 60 \\ -20 \\ 0 \end{bmatrix} \not\leq \begin{bmatrix} 2 \\ -3 \\ -1 \end{bmatrix}$$

$k = \min\{\, i \in N \,:\, A_i\bar{x} > b_i \,\} = \min\{\,1\,,\,3\,\} = 1$

$\eta_{B^1} = A_k A_{B^1}^{-1} = A_1 = [\ -1\,,\,4\ ]$, $\eta_4 = -1 < 0$ e $\eta_5 = 4 > 0$ quindi l'indice uscente è $h = B^1(2) = 5$

$B^2 = B^1 \setminus \{\,5\,\} \cup \{\,1\,\} = \{\,4\,,\,1\,\}$.

it. 2): $B^2 = \{\,4\,,\,1\,\}$, $A_{B^2} = \begin{bmatrix} 1 & 0 \\ -1 & 4 \end{bmatrix}$, $A_{B^2}^{-1} = \begin{bmatrix} 1 & 0 \\ 1/4 & 1/4 \end{bmatrix}$, $\bar{x}^2 = \begin{bmatrix} 20 \\ 11/2 \end{bmatrix}$

$$[\,\bar{y}^2\,,\,\bar{w}^2\,]_{B^2} = cA_{B^2}^{-1} = [\,4\,,\,2\,] \begin{bmatrix} 1 & 0 \\ 1/4 & 1/4 \end{bmatrix} = [\,9/2\,,\,1/2\,] \ , \ [\,\bar{y}^2\,,\,\bar{w}^2\,] = [\,1/2\,,\,0\,,\,0\,,\,9/2\,,\,0\,]$$

$$A_{N^2}\bar{x}^2 = \begin{bmatrix} 9 \\ -29/2 \\ 11/2 \end{bmatrix} \not\leq \begin{bmatrix} -3 \\ -1 \\ 20 \end{bmatrix} \implies k = 2 \implies \eta_{B^2} = [\,1/2\,,\,-1/2\,] \implies h = 4$$

it. 3): $B^3 = \{\,2\,,\,1\,\}$, $A_{B^3} = \begin{bmatrix} 1 & -2 \\ -1 & 4 \end{bmatrix}$, $A_{B^3}^{-1} = \begin{bmatrix} 2 & 1 \\ 1/2 & 1/2 \end{bmatrix}$, $\bar{x}^3 = \begin{bmatrix} -4 \\ -1/2 \end{bmatrix}$

$[\,\bar{y}^3\,,\,\bar{w}^3\,]_{B^3} = [\,9\,,\,5\,]$, $[\,\bar{y}^3\,,\,\bar{w}^3\,] = [\,5\,,\,9\,,\,0\,,\,0\,,\,0\,]$

$$A_{N^3}\bar{x}^3 = \begin{bmatrix} 7/2 \\ -4 \\ -1/2 \end{bmatrix} \not\leq \begin{bmatrix} -1 \\ 20 \\ 20 \end{bmatrix} \implies k = 3 \implies \eta_{B^3} = [\,-3/2\,,\,-1/2\,]$$

Poiché alla terza iterazione $\eta_{B^3} \leq 0$, (DA) è inferiormente illimitato e di conseguenza anche (D) lo è, quindi (P) è vuoto.

Esercizio 3.22. Si studi il problema (P) dell'esempio precedente, rappresentandolo geometricamente,

e si verifichi che esso è vuoto. Inoltre, considerando il problema (PA), si interpretino geometricamente le iterazioni svolte nell'esempio.

Esercizio 3.23. Si usi l'informazione disponibile al termine dell'algoritmo per determinare un sistema inconsistente per il problema (P) dell'esempio precedente.

3.3.3 Versioni specializzate degli algoritmi del simplesso

Ancorché qualsiasi problema di *PL* possa essere portato in una qualsiasi delle forme normali che abbiamo introdotto, le istanze che derivano da modelli di problemi reali spesso hanno caratteristiche specifiche che possono essere sfruttate per sviluppare versioni specializzate degli algoritmi del simplesso che risultino più efficienti in pratica. Abbiamo in effetti già visto un esempio di questo processo nel §3.3.2, in cui il simplesso duale viene sviluppato come la versione del simplesso primale che sfrutta la speciale struttura dei problemi di *PL* che derivano dalla riscrittura in forma primale del duale della coppia asimmetrica (cf. (3.30)).

Lo stesso principio può essere applicato in molti altri casi, rendendo molto più efficienti le implementazioni dell'algoritmo. In queste dispense non abbiamo modo di entrare nella descrizione di dettaglio di queste varianti, ma accenniamo almeno a due casi rilevanti:

- In quasi tutti i modelli reali, la maggior parte delle variabili (se non tutte) hanno esplicitamente limiti superiori ed inferiori finiti, ossia *vincoli di scatola* della forma $l \leq x \leq u$. Tali vincoli, analogamente a quanto visto nel §3.3.2, inducono una forte struttura nei sistemi primali e duali di base che portano ad essere in grado di calcolare il valore di molte delle variabili primali o duali in modo efficiente, dovendo poi risolvere sistemi lineari generali solamente con il (piccolo) sottoinsieme delle variabili rimanenti. La varianti degli algoritmi del simplesso che sfruttano questa struttura vengono dette *simplesso rivisto* (*revised simplex*); in pratica, tutte le implementazioni efficienti degli algoritmi del simplesso fanno uso di tale tecnica.
- Una delle strutture che si incontrano frequentemente nei modelli di problemi reali è quella di flusso o cammino à introdotta nel §1.2.6 ed alla quale è dedicato (quasi) tutto il Capitolo 2. Il generico problema di Flusso di Costo Minimo (MCF) è un problema di *PL*, ed è quindi possibile sviluppare varianti specializzate degli algoritmi del simplesso per risolverlo. In effetti, si può verificare che la tecnica delle *basi di cicli* descritta nel §2.5.2 per implementare l'algoritmo *Cancella-Cicli* altro non sia che l'aggiornamento della base A_B, e la soluzione dei corrispondenti sistemi primali e duali di base, per l'algoritmo del simplesso (duale) applicato alla soluzione di (MCF). Per semplicità in quella trattazione si è evitato di discutere esplicitamente il ruolo dei vincoli di capacità superiore sugli archi ($x_{ij} \leq u_{ij}$), che possono essere trattati in modo efficiente utilizzando le tecniche del simplesso rivisto accennate al punto precedente.

Per ulteriori dettagli su questo punto si rimanda alla letteratura citata.

3.3.4 Analisi post-ottimale

Data la coppia asimmetrica (P)–(D) di problemi duali, sia A_B una base ottima con $\bar{x} = A_B^{-1} b_B$ e $\bar{y} = [\, \bar{y}_B \,,\, 0\,] = [\, cA_B^{-1} \,,\, 0\,]$ la corrispondente coppia di soluzioni complementari ottime: vogliamo studiare come varia la coppia di soluzioni ottime al variare dei dati del problema.

Si tratta di un problema di notevole interesse pratico, per diversi motivi. Innanzitutto, va ricordato che per costruire il modello di *PL* sono state fatte approssimazioni (spesso notevoli), ad esempio perché sono stati assunti come lineari fenomeni che non lo sono, oppure perché alcuni parametri non sono noti con precisione. È quindi utile conoscere quanto sia "stabile" la soluzione ottima ottenuta risolvendo il problema di *PL*, cioè quanto essa sia sensibile a piccole variazioni dei dati; in questo caso si parla di *analisi di sensitività*. Altre volte si considerano alcuni dei dati come funzione di uno o più parametri e ci si pone il problema di determinare il valore ottimo della funzione obiettivo come funzione dei parametri stessi; in questo caso si parla di *analisi parametrica*. Infine, un altro caso di notevole interesse pratico è quello in cui, a partire dalla soluzione trovata, si voglia risolvere un nuovo problema che differisca dal precedente solo in alcuni dei dati (ad esempio, un nuovo vincolo, una diversa funzione obiettivo o un diverso vettore di risorse); in questo caso si parla di *riottimizzazione*.

Tratteremo nel seguito, separatamente, il caso in cui le variazioni riguardano i vettori c e b ed i casi in cui viene aggiunto al problema, o da esso eliminato, un vincolo o una variabile. Questi sono i casi per cui è possibile reagire in modo efficiente alle variazioni. I cambiamenti arbitrari di elementi della matrice A possono, ad esempio, rendere non più invertibile A_B e sono quindi più difficili da gestire in modo efficiente.

3.3.4.1 Variazione del vettore c

Supponiamo innanzitutto che il vettore dei costi c venga sostituito da un nuovo vettore c': ovviamente, la matrice di base A_B continua ad essere primale ammissibile, e $\bar{x}$ continua ad essere la soluzione primale ad essa corrispondente. La soluzione duale diventa $\bar{y}' = [\,\bar{y}'_B\,,\,\bar{y}'_N\,] = [\,c'A_B^{-1}\,,\,0\,]$: se $\bar{y}'_B \geq 0$, allora A_B rimane ottima, altrimenti si può applicare, a partire da A_B, il simplesso primale per determinare una nuova base ottima.

Consideriamo ora il caso in cui il vettore dei costi sia una funzione lineare di un parametro λ, ossia $c(\,\lambda\,) = c + \lambda\nu$ con ν un qualsiasi vettore di $\mathbb{R}^n$, e si voglia determinare quale è l'intervallo in cui può variare λ senza che la base A_B perda l'ottimalità. Per questo, è sufficiente che risulti $y(\,\lambda\,)_B = (\,c + \lambda\nu\,)A_B^{-1} \geq 0$, cioè

$$(\,c + \lambda\nu\,)A_B^{-1}u_{B(i)} \geq 0 \qquad i \in B \ ,$$

dove, ricordiamo, $A_B^{-1}u_{B(i)}$ è la colonna dell'inversa della matrice di base relativa alla variabile di base $\bar{y}_i = cA_B^{-1}u_{B(i)}$. Esplicitando queste condizioni, risulta che, per ogni $i \in B$, deve essere

$$\lambda \geq \frac{-cA_B^{-1}u_{B(i)}}{\nu A_B^{-1}u_{B(i)}} = -\frac{\bar{y}_i}{\nu A_B^{-1}u_{B(i)}} \qquad \text{se } \nu A_B^{-1}u_{B(i)} > 0 \ ,$$

$$\lambda \leq \frac{-cA_B^{-1}u_{B(i)}}{\nu A_B^{-1}u_{B(i)}} = -\frac{\bar{y}_i}{\nu A_B^{-1}u_{B(i)}} \qquad \text{se } \nu A_B^{-1}u_{B(i)} < 0 \ ;$$

queste disequazioni definiscono l'intervallo desiderato (si noti che tale intervallo deve contenere lo 0, in quanto abbiamo supposto che B sia duale ammissibile). Può essere interessante studiare l'andamento del valore ottimo della funzione obiettivo in funzione del parametro λ: indicando con $z(\lambda)$ tale funzione, è facile vedere che si tratta di una funzione convessa lineare a tratti.

Esempio 3.40. Variazione del vettore c

Si consideri il problema dell'Esempio 3.36; al termine dell'applicazione del simplesso duale si è ottenuta la base ottima $B = \{\,3\,,\,4\,\}$, a cui corrispondono le soluzioni ottime $\bar{x} = [\,2\,,\,0\,]$ e $\bar{y} = [\,0\,,\,0\,,\,1/4\,,\,1/4\,,\,0\,]$. Si consideri ora il gradiente del costo in forma parametrica: $c(\,\lambda\,) = c + \lambda\nu$, con $\nu = [\,1\,,\,1\,]$; si vuole conoscere per quali valori di $c(\,\lambda\,)$ la base B resta ottima. Dalle formule sopra esposte si ha che

$$\nu A_B^{-1} = \begin{bmatrix} 1\,,\,1 \end{bmatrix} \begin{bmatrix} 1/4 & 1/4 \\ 1/2 & -1/2 \end{bmatrix} = \begin{bmatrix} 3/4\,,\,-1/4 \end{bmatrix} \ ,$$

pertanto

a) $\nu A_B^{-1}u_{B(1)} = 3/4 > 0 \quad\Longrightarrow\quad \lambda \geq -\frac{\bar{y}_3}{3/4} = -\frac{1/4}{3/4} = -1/3$

b) $\nu A_B^{-1}u_{B(2)} = -1/4 < 0 \quad\Longrightarrow\quad \lambda \leq \frac{\bar{y}_4}{-(-1/4)} = \frac{1/4}{1/4} = 1$

da cui si ottiene che $B = \{\,3\,,\,4\,\}$ resta ottima per $-1/3 \leq \lambda \leq 1$. Per studiare la variazione del valore ottimo della funzione obiettivo al variare di λ si ricorda che $z(\,\lambda\,) = y(\,\lambda\,)_B b_B = (\,c + \lambda\nu\,)\bar{x}$, e quindi

$$z(\,\lambda\,) = \begin{bmatrix} 1 + \lambda\,,\,\lambda \end{bmatrix} \begin{bmatrix} 2 \\ 0 \end{bmatrix} = 2 + 2\lambda \ :$$

quando λ varia nell'intervallo $[\,-1/3\,,\,1\,]$, il valore ottimo della funzione obiettivo varia linearmente nell'intervallo $[\,4/3\,,\,4\,]$.

Esercizio 3.24. Si mostri geometricamente che, per il problema dell'esempio precedente, il vettore parametrico $c(\,\lambda\,)$ descrive il cono finitamente generato da A_3 e A_4, al variare di λ nell'intervallo $[\,-1/3\,,\,1\,]$.

Esercizio 3.25. Studiare le soluzioni ottime che si ottengono per $\lambda < -1/3$ e per $\lambda > 1$. Definire inoltre la funzione $z(\,\lambda\,)$ per qualsiasi valore di λ.

Presentiamo adesso un esempio che riassume i concetti appena visti.

Esempio 3.41. Calcolo della funzione valore

È dato il seguente problema di *PL* parametrico in ε, di cui è noto che la base ottima per $\varepsilon = 1$ è $B = \{1, 2\}$, con le corrispondenti soluzioni primali e duali di vase ottime $\bar{x} = [2, 2]$ e $\bar{y} = [0, 1, 0, 0, 0]$. Si vuole caratterizzare la *funzione valore* del problema, ossia il suo valore ottimo $z(\varepsilon)$ come funzione di ε.

$$\begin{array}{lrcrcl} \max & \varepsilon x_1 & + & x_2 & & \\ & & & x_2 & \leq & 2 \\ & x_1 & + & x_2 & \leq & 4 \\ & -x_1 & + & 2x_2 & \leq & 2 \\ & -x_1 & + & x_2 & \leq & 1 \\ & -x_1 & & & \leq & 1 \end{array}$$

Dalla soluzione ottima, nota, per $\varepsilon = 1$ otteniamo immediatamente che $z(1) = 4$. Applichiamo adesso l'analisi di sensitività alla base $B = \{1, 2\}$ per determinare per quali valori di ε essa resti ottima. Poiché ε influisce sulla funzione obiettivo e non sui vincoli, è ovvio che la base rimane primale ammissibile per qualsiasi valore di ε; occorre quindi verificare solamente la duale ammissibilità, ossia per quali valori di ε si abbia

$$\bar{y}_B(\varepsilon) = \begin{bmatrix} \varepsilon, 1 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1-\varepsilon, \varepsilon \end{bmatrix} \geq 0 \ .$$

Quindi, per $\varepsilon \in [0, 1]$ la base $B = \{1, 2\}$ resta ottima, la soluzione primale di base $\bar{x} = [2, 2]$ resta ottima, e pertanto $z(\varepsilon) = 2\varepsilon + 2$. Occorre adesso esaminare cosa accada al di fuori di tale intervallo; per questo utilizziamo tecniche di riottimizzazione.

Per $\varepsilon > 1$ si ha $\bar{y}_1 = 1 - \varepsilon < 0$, quindi $h = 1$ con $B(h) = 1$; si ottiene quindi

$$\xi = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \ A_N\xi = \begin{bmatrix} -1 & 2 \\ -1 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} -3 \\ -2 \\ -1 \end{bmatrix} < 0$$

e pertanto il problema è superiormente illimitato; in altri termini, $z(\varepsilon) = +\infty$ per $\varepsilon > 1$.

Per $\varepsilon < 0$ si ha $\bar{y}_2 = \varepsilon < 0$, quindi $h = 2$ con $B(h) = 2$; si ottiene quindi

$$\xi = \begin{bmatrix} -1 \\ 0 \end{bmatrix} \ , \ A_N\xi = \begin{bmatrix} -1 & 2 \\ -1 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

e quindi $k = 3$ perché, il vincolo 3 è attivo in $\bar{x} = [2, 2]$. In corrispondenza alla base $B = \{1, 3\}$ si ha

$$\bar{y}_B = \begin{bmatrix} \varepsilon & 1 \end{bmatrix} \begin{bmatrix} 2 & -1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 2\varepsilon + 1 & -\varepsilon \end{bmatrix}$$

e quindi la base è ottima per $\varepsilon \in [-1/2, 0]$; ne consegue che $z(\varepsilon) = 2\varepsilon + 2$ anche per $\varepsilon \in [-1/2, 0]$.

Per $\varepsilon < -1/2$ si ha $\bar{y}_1 = 2\varepsilon + 1 < 0$, quindi $h = 1$ con $B(h) = 1$; si ottiene quindi $\xi = [-2, -1]$. Il vertice che si incontra corrisponde alla base $B = \{3, 4\}$. In effetti, poiché per tale base risulta

$$\bar{y}_B = \begin{bmatrix} \varepsilon, 1 \end{bmatrix} \begin{bmatrix} -2 & 1 \\ -1 & = \end{bmatrix} \begin{bmatrix} \varepsilon + 1, -2\varepsilon - 1 \end{bmatrix} \ ,$$

si ha che $\bar{y}_B \geq 0$ per $\varepsilon \in [-1, -1/2]$. Per questo intervallo, la corrispondente soluzione ottima di base $\bar{x} = [0, 1]$ è ottima, e quindi $z(\varepsilon) = 1$.

Per $\varepsilon < -1$ si ha $\bar{y}_3 = \varepsilon + 1 < 0$, quindi $h = 3$ con $B(h) = 1$; si ottiene quindi $\xi = [-1, -1]$. Il vertice che si incontra corrisponde alla base $B = \{4, 5\}$, per la quale risulta

$$\bar{y}_B = \begin{bmatrix} \varepsilon, 1 \end{bmatrix} \begin{bmatrix} 0 & -1 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 1, -\varepsilon - 1 \end{bmatrix}$$

Pertanto, $\bar{y}_B \geq 0$ per $\varepsilon \leq -1$; in questo intervallo, la corrispondente soluzione ottima di base $\bar{x} = [-1, 0]$ è ottima, e quindi $z(\varepsilon) = -\varepsilon$.

Riassumendo, si ha:

$$z(\varepsilon) = \begin{cases} -\varepsilon & \text{per } \varepsilon \leq -1 \\ 1 & \text{per } -1 \leq \varepsilon \leq -1/2 \\ 2\varepsilon + 2 & \text{per } -1/2 \leq \varepsilon \leq 1 \\ +\infty & \text{per } \varepsilon > 1 \end{cases}$$

Tale funzione è convessa, e questo è vero in generale per qualsiasi funzione valore costruita in questo modo; dimostrarlo può essere un utile esercizio.

3.3.4.2 Variazione del vettore b

Consideriamo il caso della variazione di una componente del vettore b (l'estensione al caso più generale è immediata), e sia b'_k il nuovo valore assunto da b_k; la soluzione duale non cambia, mentre per la

soluzione primale si devono distinguere due casi:

$k \in N$: se risulta $A_k \bar{x} \leq b'_k$ allora A_B rimane ottima, altrimenti si può applicare il simplesso duale per determinare la nuova base ottima, conoscendo già l'indice k entrante in base;

$k \in B$: si calcola $\bar{x}' = A_B^{-1} b'_B$: se risulta $A_N \bar{x}' \leq b_N$ allora B rimane ottima, altrimenti si può applicare il simplesso duale per determinare la nuova base ottima.

Esempio 3.42. Variazione del vettore b
Si consideri la seguente coppia di problemi di PL

$$(P)\quad \begin{array}{lrcrcl} \max & 3x_1 & + & x_2 & & \\ & -2x_1 & + & x_2 & \leq & 1 \\ & x_1 & - & 2x_2 & \leq & -4 \\ & x_1 & + & x_2 & \leq & 14 \\ & x_1 & & & \leq & 8 \\ & & - & x_2 & \leq & -4 \end{array} \qquad (D)\quad \begin{array}{lrcrcrcrcrcl} \min & y_1 & - & 4y_2 & + & 14y_3 & + & 8y_4 & - & 4y_5 & \\ & -2y_1 & + & y_2 & + & y_3 & + & y_4 & & & = 3 \\ & y_1 & - & 2y_2 & + & y_3 & & & - & y_5 & = 1 \\ & y_1 & , & y_2 & , & y_3 & , & y_4 & , & y_5 & \geq 0 \end{array}$$

È facile verificare che la base ottima è $B^1 = \{\, 2\,,\, 3\,\}$ e le soluzioni ottime sono $\bar{x}^1 = [\, 8\,,\, 6\,]$ e $\bar{y}^1 = [\, 0\,,\, 2/3\,,\, 7/3\,,\, 0\,,\, 0\,]$. Sia ora $b'_4 = 7$ il nuovo valore del termine noto del quarto vincolo: siccome $4 \in N^1$, le soluzioni di base non cambiano. Si deve controllare se, dopo la perturbazione, il quarto vincolo è ancora soddisfatto:

$$A_4 \bar{x}_1 = \left[\begin{array}{cc} 1\,, & 0 \end{array}\right] \left[\begin{array}{c} 8 \\ 6 \end{array}\right] = 8 > 7 \ ;$$

poiché ciò non accade, la base B^1 non è primale ammissibile. Si applica allora il simplesso duale partendo da B^1:

$$A_{B^1} = \left[\begin{array}{cc} 1 & -2 \\ 1 & 1 \end{array}\right] \quad , \quad A_{B^1}^{-1} = \left[\begin{array}{cc} 1/3 & 2/3 \\ -1/3 & 1/3 \end{array}\right]$$

e l'indice entrante è ovviamente $k = 4$; si ha quindi

$$\eta_{B^1} = A_k A_{B^1}^{-1} = \left[\begin{array}{cc} 1\,, & 0 \end{array}\right] \left[\begin{array}{cc} 1/3 & 2/3 \\ -1/3 & 1/3 \end{array}\right] = \left[\begin{array}{cc} 1/3\,, & 2/3 \end{array}\right] \ .$$

Siccome entrambe le componenti di η_{B^1} sono positive, il passo di spostamento e l'indice uscente dalla base si ottengono mediante il criterio del minimo rapporto

$$\bar{\theta} = \min\left\{ \frac{\bar{y}^1_2}{\eta^1_2}\,,\, \frac{\bar{y}^1_3}{\eta^1_3} \right\} = \min\left\{ \frac{2/3}{1/3}\,,\, \frac{7/3}{2/3} \right\} = \min\left\{ 2\,,\, \frac{7}{2} \right\} = 2 \ ;$$

quindi $h = 2$ e si ottiene la nuova base $B^2 = \{\, 4\,,\, 3\,\}$. Alla successiva iterazione si ha quindi

$$A_{B^2} = \left[\begin{array}{cc} 1 & 0 \\ 1 & 1 \end{array}\right] \quad , \quad A_{B^2}^{-1} = \left[\begin{array}{cc} 1 & 0 \\ -1 & 1 \end{array}\right] \quad , \quad \bar{x}^2 = \left[\begin{array}{c} 7 \\ 7 \end{array}\right] \quad , \quad \bar{y}^2_{B^2} = \left[\begin{array}{cc} 2\,, & 1 \end{array}\right] \quad , \quad \bar{y}^2 = \left[\begin{array}{ccccc} 0\,, & 0\,, & 1\,, & 2\,, & 0 \end{array}\right]$$

$$A_{N^2} \bar{x}^2 = \left[\begin{array}{cc} -2 & 1 \\ 1 & -2 \\ 0 & -1 \end{array}\right] \left[\begin{array}{c} 7 \\ 7 \end{array}\right] = \left[\begin{array}{c} -7 \\ -7 \\ -7 \end{array}\right] < \left[\begin{array}{c} 1 \\ -4 \\ -4 \end{array}\right] = b_{N^2} \ .$$

Poiché $\bar{x}^2$ è ammissibile, B^2 è ottima; $\bar{x}^2$ e $\bar{y}^2$ sono, rispettivamente, le soluzioni ottime dei problemi (P') e (D') ottenuti a seguito della variazione del vettore dei termini noti b.

Esercizio 3.26. Si analizzino geometricamente i passi svolti nell'esempio precedente.

In analogia a quanto fatto per il vettore dei costi, assumiamo che il vettore delle risorse sia una funzione lineare di un parametro λ, cioè del tipo $b(\,\lambda\,) = b + \lambda\nu$ con ν un qualsiasi vettore di $\mathbb{R}^m$, e si voglia determinare quale è l'intervallo in cui può variare λ senza che la base A_B perda l'ottimalità: per questo, basta notare che la soluzione primale di base corrispondente a B per un dato valore di λ è

$$x(\,\lambda\,) = A_B^{-1}(\, b_B + \lambda\nu_B\,) = \bar{x} + \lambda A_B^{-1}\nu_B \ ,$$

per cui deve risultare

$$A_N x(\,\lambda\,) = A_N \bar{x} + \lambda A_N A_B^{-1}\nu_B \leq b_N + \lambda\nu_N \ ,$$

e, a partire da tali relazioni, è immediato determinare l'intervallo in cui può variare λ.

Esercizio 3.27. Determinare, per la coppia di problemi dell'Esempio 3.42, l'intervallo in cui può variare λ senza che la base A_{B^1} perda l'ottimalità a seguito della variazione parametrica del vettore dei termini noti $b(\lambda) = b + \lambda\nu$ con $\nu = [\,0\,,\,1\,,\,1\,,\,0\,,\,0\,]$.

Esercizio 3.28. Si utilizzino le idee appena esposte per sviluppare i dettagli relativi alla determinazione di una base ammissibile per il simplesso duale. In particolare, sia B la base ottima di (DA) e si assuma che almeno una delle variabili "artificiali" w_i sia in base; si discuta come determinare se esiste oppure no un valore sufficientemente grande di M per cui la base non sia più ottima (w_i esca di base). Si estenda poi la discussione al caso in cui (DA) sia inferiormente illimitato ma almeno una variabile artificiale appartenga a $B \cup \{\,k\,\}$ al momento in cui l'algoritmo termina.

3.3.4.3 Aggiunta o rimozione di un vincolo in (P) (di una variabile in (D))

Sia $A_{m+1}x \leq b_{m+1}$ il vincolo da aggiungere: se $A_{m+1}\bar{x} \leq b_{m+1}$, allora A_B rimane ottima, altrimenti si può applicare il simplesso duale per determinare la nuova base ottima, conoscendo già l'indice $(m+1)$ della variabile che dovrà entrare in base. Questo caso è equivalente a quello in cui si debba inserire una nuova variabile in (D).

La rimozione del vincolo i-esimo corrisponde a porre $b_i = +\infty$, e quindi può essere affrontata con i meccanismi descritti nel paragrafo precedente.

3.3.4.4 Aggiunta o rimozione di una variabile in (P) (di un vincolo in (D))

Sia x_{n+1} la nuova variabile da aggiungere, con corrispondente colonna A^{n+1} e coefficiente di costo c_{n+1}. Supponiamo che $x_{n+1} \geq 0$; in questo caso, oltre alla nuova colonna si aggiunge anche la riga corrispondente al vincolo di non negatività della nuova variabile. La nuova matrice è allora

$$A' = \begin{bmatrix} A & A^{n+1} \\ 0 & -1 \end{bmatrix} \qquad \text{e la sottomatrice} \qquad A'_{B'} = \begin{bmatrix} A_B & A_B^{n+1} \\ 0 & -1 \end{bmatrix}$$

è quadrata e non singolare, quindi è una matrice di base associata alla base $B' = B \cup \{m+1\}$, con $N' = N$. Alla nuova base corrispondono le seguenti soluzioni di base:

- $x' = [\,\bar{x}\,,\,x'_{n+1}\,] = [\,\bar{x}\,,\,0\,]$, che risulta essere ovviamente ammissibile;
- $y' = [\,y'_{B'}\,,\,y'_N\,]$, dove $y'_N = 0$ e $y'_{B'} = [\,\bar{y}_B\,,\,y'_{m+1}\,]$, con $y'_{m+1} = \bar{y}_B A_B^{n+1} - c_{n+1}$.

La base B' risulta duale ammissibile se e solo se $y'_{m+1} \geq 0$; nel caso B' non sia duale ammissibile e quindi non ottima, si ha comunque una base di partenza per il simplesso primale, per cercare l'ottimo del problema trasformato partendo dalla base e dalle soluzioni ottime del problema originario.

Esercizio 3.29. Si dimostri che le soluzioni x' e y', sopra indicate, sono le soluzioni di base associate alla base $B' = B \cup \{\,m+1\,\}$; si dimostrino le asserzioni fatte sull'ammissibilità primale e duale.

Supponiamo ora che la nuova variabile x_{n+1} non sia vincolata in segno. In tal caso si possono applicare le trasformazioni (3.3), sostituendo la nuova variabile con la differenza di due variabili non negative, ed ottendo un problema equivalente la cui matrice dei vincoli è

$$A'' = \begin{bmatrix} A & A^{n+1} & -A^{n+1} \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \text{e la cui sottomatrice} \qquad A''_{B''} = \begin{bmatrix} A_B & A_B^{n+1} & -A_B^{n+1} \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

è la matrice di base associata alla base $B'' = B \cup \{\,m+1\,,\,m+2\,\}$, con $N'' = N$. A B'' sono associate le seguenti soluzioni di base:

- $x'' = [\,\bar{x}\,,\,x^+_{n+1}\,,\,x^-_{n+1}\,] = [\,\bar{x}\,,\,0\,,\,0\,]$, che risulta essere ammissibile;
- $y'' = [\,y''_{B''}\,,\,y''_N\,]$, dove $y''_N = 0$ e $y''_{B''} = [\,\bar{y}_B\,,\,y''_{m+1}\,,\,y''_{m+2}\,]$, con $y''_{m+1} = y''_{m+2} = \bar{y}_B A_B^{n+1} - c_{n+1}$.

Anche in questo caso la base B'' risulta essere duale ammissibile, e quindi ottima, se e solo se $y''_{m+1} \geq 0$ (e quindi $y''_{m+2} \geq 0$). Se ciò non si verifica, B'' costituisce una base di partenza per il Simplesso Primale.

Esercizio 3.30. Si dimostri che le soluzioni x'' e y'' sopra indicate sono le soluzioni di base associate alla base $B'' = B \cup \{m+1, m+2\}$; si dimostrino le asserzioni fatte sull'ammissibilità primale e duale.

Esempio 3.43. Aggiunta di una variabile

Si consideri la coppia di problemi di PL dell'Esempio 3.42, a cui viene aggiunta una nuova variabile x_3, la relativa colonna A^3 e la componente c_3 riportati nella seguente coppia di problemi trasformati

$$
\begin{array}{llrcrcrcl}
\max & & 3x_1 & + & x_2 & - & 2x_3 & & \\
 & & -2x_1 & + & x_2 & + & x_3 & \leq & 1 \\
 & & x_1 & - & 2x_2 & - & 3x_3 & \leq & -4 \\
(P') & & x_1 & + & x_2 & & & \leq & 14 \\
 & & x_1 & & & + & x_3 & \leq & 8 \\
 & & & - & x_2 & + & 2x_3 & \leq & -4 \\
 & & & & & - & x_3 & \leq & 0
\end{array}
\qquad
\begin{array}{lrcrcrcrcrcrcl}
\min & y_1 & - & 4y_2 & + & 14y_3 & + & 8y_4 & - & 4y_5 & & & \\
 & -2y_1 & + & y_2 & + & y_3 & + & y_4 & & & & = & 3 \\
(D') & y_1 & - & 2y_2 & + & y_3 & & & - & y_5 & & = & 1 \\
 & y_1 & - & 3y_2 & & & + & y_4 & + & 2y_5 & - & y_6 = & -2 \\
 & y_1 & , & y_2 & , & y_3 & , & y_4 & , & y_5 & , & y_6 \geq & 0
\end{array}
$$

La base ottima per (P) è $B = \{\,2\,,\,3\,\}$, e le corrispondenti soluzioni ottime sono $\bar{x} = [\,8\,,\,6\,]$ e $\bar{y} = [\,0\,,\,2/3\,,\,7/3\,,\,0\,,\,0\,]$. Inoltre, la matrice di base e la sua inversa sono:

$$A_B = \begin{bmatrix} 1 & -2 \\ 1 & 1 \end{bmatrix} \quad , \quad A_B^{-1} = \begin{bmatrix} 1/3 & 2/3 \\ -1/3 & 1/3 \end{bmatrix} \; .$$

La base per il problema trasformato (P') è $B' = B \cup \{\,6\,\} = \{\,2\,,\,3\,,\,6\,\}$ ($N' = N = \{\,1\,,\,4\,,\,5\,\}$), in cui il vincolo $-x_3 \leq 0$ è attivo. La nuova matrice di base e la sua inversa sono

$$A'_{B'} = \begin{bmatrix} A_B & A_B^{n+1} \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 & -2 & -3 \\ 1 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \quad , \quad {A'}_{B'}^{-1} = \begin{bmatrix} 1/3 & 2/3 & -1 \\ -1/3 & 1/3 & 1 \\ 0 & 0 & -1 \end{bmatrix}$$

e quindi è facile verificare che la corrispondente soluzione primale di base

$$\bar{x}' = {A'}_{B'}^{-1} b_{B'} = \begin{bmatrix} 1/3 & 2/3 & -1 \\ -1/3 & 1/3 & 1 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} -4 \\ 14 \\ 0 \end{bmatrix} = \begin{bmatrix} 8 \\ 6 \\ 0 \end{bmatrix}$$

è esattamente $\bar{x}' = [\,\bar{x}\,,\,0\,]$. La soluzione duale di base è data da

$$\bar{y}'_{B'} = c' {A'}_{B'}^{-1} = \begin{bmatrix} 3\,,\,1\,,\,-2 \end{bmatrix} \begin{bmatrix} 1/3 & 2/3 & -1 \\ -1/3 & 1/3 & 1 \\ 0 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 2/3\,,\,7/3\,,\,0 \end{bmatrix} \quad , \quad \bar{y}' = \begin{bmatrix} 0\,,\,2/3\,,\,7/3\,,\,0\,,\,0\,,\,0 \end{bmatrix} \; .$$

Si noti che si è ottenuto $\bar{y}'_6 = 0$; infatti, per verifica, applicando la formula fornita si ottiene:

$$\bar{y}'_6 = \bar{y}_B A_B^6 - c_6 = \begin{bmatrix} 2/3\,,\,7/3 \end{bmatrix} \begin{bmatrix} -3 \\ 0 \end{bmatrix} + 2 = -2 + 2 = 0 \; .$$

Si è ottenuta una soluzione duale ammissibile e degenere; pertanto la base B' è ottima, e le soluzioni di base ottime, rispettivamente per (P') e (D'), sono $\bar{x}'$ e $\bar{y}'$.

La rimozione della variabile x_j corrisponde all'inserimento nel problema del vincolo $x_j = 0$, o equivalentemente dei due vincoli $x_j \geq 0$ e $x_j \leq 0$, e quindi può essere affrontata con i meccanismi descritti nel paragrafo precedente.

Riferimenti Bibliografici

S. Boyd, L. Vandenberghe "Convex Optimization" Cambridge University Press, 2008
https://web.stanford.edu/~boyd/cvxbook

G.B. Dantzig, M.N. Thapa "Linear Programming: Theory and Extensions" Springer, 2003

J. Lee "A First Course in Linear Optimization v4.06" 2022
https://github.com/jon77lee/JLee_LinearOptimizationBook/blob/master/JLee.4.06.zip

K.G. Murty "Linear and combinatorial programming" Wiley, 1976

F. Schoen "Optimization Models" free e-book version, 2022
https://webgol.dinfo.unifi.it/OptimizationModels/contents.html

M. Padberg “Linear optimization and extensions” Springer-Verlag, 1995

M. Pappalardo, M. Passacantando “Ricerca Operativa” Edizioni Plus, 2013

Capitolo 4

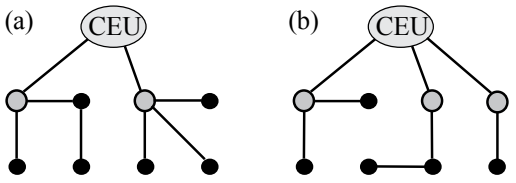
Ottimizzazione Combinatoria

4.1 Introduzione

L'*Ottimizzazione Combinatoria* (OC) studia i problemi di ottimizzazione in cui l'insieme ammissibile è definito in termini di strutture combinatorie, tra le quali svolgono sicuramente un ruolo di rilievo i grafi. La caratteristica fondamentale di tali problemi è quella di avere insiemi ammissibili *discreti*, a differenza ad esempio della PL, in cui l'insieme ammissibile è *continuo*. Le metodologie necessarie per risolvere problemi di OC sono pertanto spesso diverse da quelle utilizzate per risolvere problemi nel continuo.

Nei Capitoli 1 e 2 abbiamo già incontrato molti problemi di OC. In particolare, nel Capitolo 2 abbiamo descritto alcune importanti classi di problemi di OC che ammettono algoritmi risolutivi di complessità polinomiale. Moltissimi problemi di OC sono invece "difficili" ($\mathcal{NP}$-ardui, si veda l'Appendice A), ed è di questi problemi che ci occuperemo in questo capitolo e nei successivi. In effetti, i problemi di OC per i quali esistono algoritmi polinomiali hanno caratteristiche molto peculiari, e capita molto spesso che, non appena si introduca qualche variante, apparentemente trascurabile, ad un problema di OC "facile", il problema diventi subito difficile.

Esempio 4.1. Un problema di progetto di rete



Si consideri il problema della Banca Gatto & Volpe definito nell'Esempio 1.6. La banca sta pianificando un significativo aggiornamento del sistema informativo verso un paradigma "hybrid cloud", in cui parte delle applicazioni (quelle cruciali per la sicurezza delle transazioni) vengono ancora eseguite dal mainframe nel CEU, mentre altre, relative ad esempio a tecniche "data driven" per la profilazione degli utenti, vengono eseguite sui server di un provider "cloud"; per motivi di sicurezza però l'unica connessione con Internet, e quindi il provider, passa per il CEU. Chiaramente, questo tipo di cambiamento aumenta sensibilmente la quantità di informazione che viene inviata sulla rete. Data una soluzione, cioè un albero, tutto il traffico inviato al CEU da tutte le filiali appartenenti ad un certo sottoalbero deve passare per l'unico collegamento tra il nodo "radice", che rappresenta il CEU, e la filiale che funge da radice del sottoalbero. Quindi, sarebbero proprio questi collegamenti "critici" ad essere saturati per primi qualora la rete non fosse dimensionata opportunamente. Inoltre, in caso di un guasto ad una di queste linee tutte le filiali rappresentate da nodi nel sottoalbero corrispondente verrebbero disconnesse dal CEU. Ad ogni nodo i diverso dalla radice possiamo associare quindi un peso b_i corrispondente alla massima banda utilizzata dalla corrispondente filiale: per fare in modo che tutte le filiali abbiano sempre sufficiente banda per comunicare con il CEU, dobbiamo richiedere che nella soluzione del problema la somma dei pesi dei nodi in ciascun sottoalbero della radice sia al più Q, dove Q è la capacità dei collegamenti. La figura qui sopra mostra in (a) un esempio di un albero di copertura non ammissibile ed in (b) di uno ammissibile per $Q = 3$ se tutti i nodi hanno peso unitario; i nodi evidenziati sono quelli collegati al CEU dai collegamenti "critici".

Il corrispondente problema di OC, noto come *Constrained MST* (CMST) è una variante apparentemente minore di (MST). Si tratta invece di un problema $\mathcal{NP}$-arduo (tranne per valori particolari di Q) mentre, come abbiamo visto, (MST) è polinomiale. Infatti, nella pratica istanze di (MST) su grafi con decine di migliaia di nodi non presentano alcuna difficoltà dal punto di vista della risolubilità, mentre istanze di (CMST) con poche centinaia di nodi sono a tutt'oggi per lo più insolubili.

Si può affermare che la grande maggioranza dei problemi di OC che si incontrano nella realtà sono

difficili. Pertanto chiunque si confronti, occasionalmente o regolarmente, con la soluzione di questi problemi (che, tipicamente, emergono dai modelli di situazioni reali) deve essere conscio che riuscire effettivamente a risolverli per le dimensioni richieste dalle applicazioni sia tutt'altro che scontato.

Dal punto di vista pratico, le modalità tipiche con cui si affronta la soluzione di problemi di *OC* si sono evolute con lo sviluppo della tecnologia e l'affinamento delle metodologie algoritmiche. Quando queste dispense sono state inizialmente scritte, alla fine del millennio scorso, non erano disponibili strumenti software generali in grado di affrontare con qualche speranza di successo la stragrande maggioranza dei problemi. Pertanto, un esperto di Ricerca Operativa doveva necessariamente avere un'ampia conoscenza dei problemi di *OC* più rilevanti, e dei relativi algoritmi risolutivi, che potessero essere utilizzati come strumenti di base per lo sviluppo di approcci *ad hoc* per ciascun problema. Per questo era necessaria sia la comprensione delle principali metodologie (meta-)algoritmiche disponibili per la soluzione di questo tipo di problemi, quali quelle discusse nei prossimi capitoli, che competenze relative alla soluzione efficiente di problemi più "facili", quali la *PL* o i problemi di flusso su rete visti nei capitoli precedenti, in quanto come vedremo tipicamente gli algoritmi per problemi di *OC* "difficili" fanno ricorso ad algoritmi per questi ultimi. Di conseguenza, il lavoro richiedeva principalmente abilità di sviluppo algoritmico, ancorché siano sempre state fondamentali anche le capacità modellistiche.

Le evoluzioni negli ultimi due decenni hanno però cambiato in modo sostanziale questo stato di fatto. L'effetto della crescita esponenziale delle capacità dei sistemi di calcolo (c.d. "Legge di Moore") da solo ovviamente non può rendere accessibile la soluzione di problemi il cui costo computazionale aumenta anch'esso esponenzialmente con le dimensioni; ben più rilevante, in questo caso, è stato il sostanziale affinamento delle metodologie algoritmiche. Questo ha portato allo sviluppo di potenti *solutori general-purpose* in grado di affrontare (ancorché senza garanzie di successo) qualsiasi istanza di classi molto ampie di problemi di *OC*, tra le quali spicca principalmente la *Programmazione Lineare Intera* (PLI), alla quale non per caso è interamente dedicato il prossimo paragrafo. Mediante il continuo affinamento delle tecniche per la soluzione di questi problemi, tra le quali ha avuto un ruolo di particolare spicco quella delle *disuguaglianze valide*, i solutori sono gradualmente diventati in grado di risolvere regolarmente istanze di problemi con decine di migliaia (ed anche più) di variabili. Questo non è (e non potrebbe essere, a meno che $\mathcal{P} = \mathcal{NP}$) garantito: esistono comunque casi istanze anche con poche centinaia di variabili che risultano sostanzialmente irresolubili. Ma sempre più spesso ci si trova nella situazione in cui semplicemente modellando un problema come *PLI* ed utilizzando un solutore general-purpose—di cui sono disponibili anche implementazioni open-source molto efficienti, ancorché quelle commerciali normalmente abbiano prestazioni superiori—si riescono a risolvere le istanze provenienti dai problemi reali in tempi compatibili con le esigenze del processo decisionale.

Pertanto, il lavoro dell'esperto di Ricerca Operativa, almeno per quanto riguarda la sua componente relativa alla soluzione di problemi reali, è gradualmente cambiato in modo significativo. Al giorno di oggi lo sviluppo di algoritmi ad-hoc non è più la prima (e fondamentalmente unica, come era un tempo) opzione: piuttosto, il processo passa tipicamente per lo sviluppo di un modello, spesso di *PLI*, e la verifica delle prestazoni dei solutori general-purpose. In diversi casi già questo può essere sufficiente per supportare in maniera adeguata il processo decisionale, il che porta ad una sempre maggiore enfasi sulle capacità modellistiche (cf. Capitolo 1) rispetto alle più tradizionali competenze algoritmiche.

Ciò non significa però assolutamente che le competenze algoritmiche siano ormai non necessarie, per diversi ordini di motivi:

- I solutori general-purpose sono software altamente complessi che combinano molti elementi diversi, ciascuno dei quali deve collaborare armoniosamente con gli altri ai fini di ottenere un processo risolutivo complessivamente efficiente. Questo significa tipicamente prendere decisioni non ovvie relative al bilanciamento del tempo speso in ciascuna di queste componenti, che si sostanziano in pratica nella scelta di *parametri algoritmici*. I moderni solutori general-purpose possono avere più di 100 di tali parametri; anche se è vero i valori di default vengono scelti sulla base di estese esperienze computazionali per funzionare "abbastanza bene quasi sempre", e che non tutti i parametri sono critici, si da comunque spesso il caso che opportune scelte (non di

default) dei valori di questi parametri possano dare differenze di ordini di grandezza sul tempo risolutivo finale. Pertanto, l'utente di questi sistemi deve avere una conoscenza dei principali elementi algoritmici che li compongono, in modo da saper interpretare il funzionamento del solutore, diagnosticare i problemi che eventualmente incontri e che non gli permettono di risolvere efficientemente le istanze di interesse, e provare a risolverli (tra le altre cose) attraverso l'opportuna scelta dei parametri algoritmici.

- Come verrà discusso nel seguito, un aspetto non intuitivo ma fondamentale del processo di soluzione attraverso software general-purpose è la *corretta scelta della formulazione*. Infatti, come abbiamo già avuto modo di sottolineare in qualche caso nel Capitolo 1, lo stesso problema può essere modellato in molti modi diversi. Spesso la scelta tra i diversi modelli è tutt'altro che ininfluente; al contrario, utilizzare il "giusto" modello può fare tutta la differenza tra essere in grado di risolvere le istanze con l'efficienza richiesta e non essere neanche vicini a tale risultato. La selezione del modello più efficiente è però assolutamente non banale, e si basa sulla comprensione (con un ragionevole livello di dettaglio) dell'effetto che le diverse scelte hanno sulle molte componenti del processo risolutivo alle quali si è già fatto riferimento. Pertanto, anche la capacità di sviluppare formulazioni efficienti non può prescindere dalla conoscenza approfondita degli algoritmi che vengono utilizzati per risolverle.
- Esiste ancora una comunità che si occupa dello sviluppo di solutori, siano essi per classi di problemi molto grandi (es. *PLI*) o specializzati per problemi più specifici. Questo è comunque ancora importante per diversi motivi; in particolare, molti problemi di *OC* che emergono da applicazioni di grande interesse sono troppo complessi per essere risolti semplicemente attraverso l'utilizzo di solutori general-purpose. Ciò non deve sorprendere data l'inerente difficoltà di questi problemi. Pertanto la capacità di ideare algoritmi efficienti, e la conoscenza dei principali paradigmi algoritmici, resta fondamentale per molte applicazioni allo stato dell'arte. Anche per questo tipo di applicazioni vengono spesso (se non sempre) utilizzati solutori general-purpose, data la loro grande efficienza, ma non semplicemente risolvendo l'intero problema mediante di essi; piuttosto vengono utilizzate metodologie meta-algoritmiche quali le *math-euristiche* o gli *schemi di deocmposizione* nelle quali i solutori vengono utilizzati per risolvere efficientemente "parti" del problema (molto spesso già comunque $\mathcal{NP}$-complete), le cui soluzioni formano poi l'input di ulteriori passi degli algoritmi—spesso in modo iterativo—che alla fine determinano la soluzione (esatta o euristica) del problema originale. Questo mostra ancora una volta come le competenze algoritmiche siano importanti anche nell'"era dei solutori", ancorché non avremo modo di descrivere in dettaglio questo tipo di approcci in queste dispense.
- Infine, i principi di base di questi approcci risolutivi sono un importante patrimonio dell'algoritmica e dell'informatica, e quindi la loro conoscenza dovrebbe far parte delle competenze degli esperti di queste discipline.

Per tutti i motivi sopra elencati, in questo capitolo introdurremo e discuteremo in modo generale alcune delle principali proprietà dei problemi di *OC* che hanno rilevanza per lo sviluppo di approcci algoritmici per tale classe di problemi. I Capitoli 5, 6 e 7 descriveranno invece alcune classi di algoritmi per problemi di *OC*; in particolare, vedremo come la principale classe di algoritmi esatti per questo tipo di problemi (cf. Capitolo 7) si basi sulla combinazione dell'idea dell'*enumerazione implicita* (o *divide et impera*) con due elementi—in principio, ancorché non necessariamente in pratica—completamente disgiunti (le euristiche, discusse nel Capitolo 5, ed i rilassamenti discussi nel Capitolo 6. Ma per comprendere la necessità di tutta questa complessità algoritmica sono necessarie le premesse che verranno discusse in questo capitolo.

4.2 Programmazione Lineare Intera (Mista)

I problemi di *Programmazione Lineare Intera* si differenziano da quelli di *PL* unicamente per il fatto che le variabili possono assumere solamente valori interi. Tale *vincolo di integralità* ha però un enorme impatto. Innanzitutto, "l'espressività" del modello aumenta in maniera consistente: le variabili intere

possono essere utilizzate per modellare condizioni logiche (decisioni "tutto o niente") e situazioni in cui le decisioni si prendono tra un numero finito di possibili alternative, permettendo di modellare moltissime situazioni reali come abbiamo visto nel Capitolo 1. Si può affermare che la grande maggioranza dei modelli di ottimizzazione utilizzati in pratica sono di *PLI*, in quanto nella maggior parte delle applicazioni reali esistono condizioni logiche ed è necessario compiere scelte discrete. Esistono classi ancora più generali di problemi di ottimizzazione che contengono strettamente la *PLI* e che permettono di utilizzare funzioni nonlineari nelle formulazioni. In alcuni casi le funzioni nonlineari che si possono utilizzare sono "strettamente regolate", il che porta a classi di problemi che sono "in pratica difficili quanto la *PLI*"; è anche possibile utilizzare funzioni qualsiasi, ma questo può avere il costo di un ulteriore sostanziale aumento della difficoltà dei problemi. Poiché non tutti i fenomeni del mondo si prestano ad essere descritti in termini di relazioni lineari, queste classi più generali sono di grande utilità pratica in molte applicazioni, anche se come abbiamo visto nel Capitolo 1 è possibile derivare approssimazioni lineari a tratti di funzioni nonlineari che in alcuni casi possono offrire una valida alternativa modellistica. La discussione degli approcci risolutivi per queste classi di problemi richiederebbe però l'uso di strumenti matematici molto più complessi che non possono trovare spazio in queste dispense; pertanto nel seguito ci concentreremo sulla sola *PLI*. È comunque vero che la grande parte delle idee algoritmiche che si utilizzano per classi più generali sono la naturale estensione di quelle che vedremo per la *PLI*-

Come abbiamo visto nel Capitolo 1, molti problemi di *OC* possono essere formulati come problemi di *PLI*. In effetti, nella pratica si tende a considerare sostanzialmente coincidenti le due classi dei problemi. Questo è dovuto a due ragioni concomitanti:

- da una parte, i problemi di *OC* vengono normalmente formulati come problemi di *PLI*, e buona parte degli approcci risolutivi per i problemi di *OC* si basa su tali formulazioni;
- d'altra parte, molte delle tecniche efficienti per la soluzione di problemi di *PLI* si fondano sull'individuazione e sullo sfruttamento di strutture combinatorie specifiche nel modello *PLI*, ossia sull'individuazione di (sotto)problemi di *OC* corrispondenti al modello di *PLI* o a sue parti.

Si consideri ad esempio il problema (MST) (cf. Esempio 1.6): per tale problema abbiamo fornito sia una formulazione in termini di *OC*, nella quale l'insieme ammissibile è definito come l'insieme di tutti gli alberi di copertura di un grafo dato, sia una formulazione in termini di *PLI*, in cui l'insieme ammissibile è definito come l'insieme di tutti i vettori in $\{0,1\}^m$ ($m = |A|$) che rispettano un certo insieme di vincoli lineari. Tali vincoli assicurano che tutte le soluzioni ammissibili del problema possano essere interpretate come vettori di incidenza di sottografi connessi del grafo originario, e la funzione obiettivo assicura che tra tutti questi venga selezionato il vettore di incidenza di un albero di copertura di costo minimo.

Esiste quindi una forte connessione tra i problemi di *OC* e quelli di *PLI*. Le due classi non sono però completamente coincidenti. Da una parte, la *PLI* fornisce un potente linguaggio per formulare in modo uniforme sia problemi di *OC* definiti su strutture molto diverse, ma anche problemi che può essere molto difficile ricondurre a problemi di *OC*: la *PLI* è in qualche senso "più espressiva" di *OC*. D'altra parte, possono esistere molte formulazioni *PLI* diverse di uno stesso problema di *OC*, e la formulazione come problema di *OC* è spesso "più informativa" delle formulazioni *PLI*, nel senso che può essere più facile derivare proprietà utili per la soluzione del problema lavorando direttamente sulla sua struttura combinatoria piuttosto che sulle sue formulazioni in termini di *PLI*.

Poiché ai problemi di *PLI* si possono applicare le stesse trasformazioni che abbiamo visto nel Capitolo 3 per i problemi di *PL*, possiamo assumere che i problemi di *PLI* siano esprimibili in forme standard analoghe a quelle già introdotte, quali ad esempio

$$(PLI) \qquad \max\{\, cx \,:\, Ax \leq b \;,\; x \in \mathbb{Z}^n \,\} \;.$$

Si parla inoltre di problemi di *Programmazione Lineare Mista-Intera* (*PLMI*) quando solamente alcune delle variabili sono vincolate ad essere intere: tali problemi hanno quindi la forma

$$(PLM) \qquad \max\{\, c'x' + c''x'' \,:\, A'x' + A''x'' \leq b \;,\; x' \in \mathbb{Z}^n \,\} \;.$$

Quasi tutti gli approcci per la *PLI* che descriveremo possono essere generalizzati alla *PLM*, spesso solamente al costo di complicazioni nella descrizione. Per semplicità ci riferiremo quindi sempre a problemi di *PLI*.

4.2.1 Il rilassamento continuo

Il motivo principale per cui sono disponibili molti risultati algoritmici (implementati in software disponibile e ben ingegnerizzato) per la *PLI* risiede nel fatto che, per questa classe di problemi si possono utilizzare i potenti risultati teorici e le efficienti metodologie algoritmiche relative ai problemi di *PL*. Descriviamo adesso brevemente le principali relazioni tra la *PLI* e la *PL*; questo argomento sarà poi ripreso in maggiore dettaglio nel Capitolo 6. Un esempio dell'insieme ammissibile del problema (*PLI*)

$$\mathcal{F} = \{\, x \in \mathbb{Z}^n \,:\, Ax \leq b \,\} \ ,$$

è mostrato in Figura 4.1(a): i vincoli lineari $Ax \leq b$ definiscono un poliedro convesso

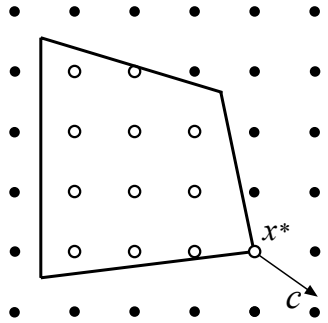


Figura 4.1: *PL* e *PLI*

$$\bar{\mathcal{F}} = \{\, x \in \mathbb{R}^n \,:\, Ax \leq b \,\} \ ,$$

e l'insieme ammissibile è formato dall'intersezione tra la "griglia" dei punti a coordinate intere e $\bar{\mathcal{F}}$, ossia da tutti i punti a coordinate intere che appartengono al poliedro (punti bianchi). Essendo formato da punti isolati, $\mathcal{F}$ non è convesso, il che spiega sostanzialmente la difficoltà dei problemi *PLI*. Senza volere entrare troppo nel dettaglio, cerchiamo adesso di descrivere intuitivamente perchè la convessità sia in grado di rendere un problema "facile", e quindi la sua assenza lo renda "difficile".

Il risultato fondamentale che rende la *PL* facile è il Lemma 3.3: le soluzioni ottime sono tutte e sole quelle per cui non esistono direzioni ammissibili di crescita. La dimostrazione del teorema, per la direzione non ovvia, usa la convessità della regione ammissibile: se per assurdo per una certa soluzione ammissibile $\bar{x}$ non esistessero direzioni ammissibili di crescita, ma $\bar{x}$ non fosse ottima, allora per una qualsiasi soluzione ottima $x^* \neq \bar{x}$ la direzione $d = x^* - \bar{x}$ è banalmente di crescita, ma è la convessità della regione ammissibile che garantisce che d sia anche ammissibile, e quindi dimostra il risultato. In assenza di convessità, non è dato che d sia una direzione ammissibile, e quindi il risultato non è valido: $\bar{x}$ può non essere ottima, ma non esistono direzioni di crescita.

Questa discussione richiede però un ulteriore approfondimento. Il punto è che la definizione di direzione ammissibile data nel §3.1.1 richiede che esista un $\bar{\lambda} > 0$ per cui $x(\lambda) = \bar{x} + \lambda\xi$ sia ammissibile *per ogni* $\lambda \in [0, \bar{\lambda}]$. In realtà, la definizione "corretta" di direzione ammissibile richiederebbe semplicemente che esista un $\bar{\lambda} > 0$ per cui $x(\lambda)$ sia ammissibile. Le due definizioni sono equivalenti nel caso della *PL* per via, appunto, della convessità della regione ammissibile, ma ovviamente non lo sono in generale. Infatti, con questa definizione il Lemma 3.3 in effetti continua a valere: banalmente, poiché x^* è ammissibile, $d = x^* - \bar{x}$ è una direzione ammissibile secondo la definzione meno stringente (con $\bar{\lambda} = 0$). Il punto critico è però la caratterizzazione dell'insieme di direzioni crescita: la Proprietà 3.1, su cui si basa tutto lo sviluppo successivo, richiede la definizione originale, e quindi la convessità. Questo è illustrato nella Figura 4.2: per la funzione obiettivo c indicata (diversa da quella della Figura 4.1), è ovvio verificare geometricamente che la soluzione x^* indicata sia quella ottima. Le frecce tratteggiate in figura sono tutte e sole le soluzioni ammissibili, secondo la definizione data qui: è immediato verificare che esse hanno tutte prodotto scalare negativo con c, e quindi non sono di crescita, confermando il teorema. Viceversa è immediato verificare che per qualsiasi soluzione non ottima esiste almeno una direzione ammissibile di crescita (quella che "porta ad x^*"). Il problema ovviamente è che l'insieme delle direzioni secondo la nuova definizione è "difficile da caratterizzare": sostanzialmente, richiede di identificare tutte le soluzioni ammissibili del problema. Questo rende ovvio che conoscere questo insieme permetta di stabilire se una soluzione è ottima:

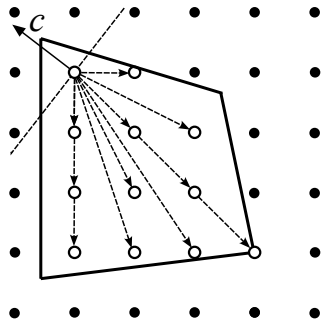


Figura 4.2: Direzioni di crescita

nell'approccio ovvio—ma non se ne conoscono di sostanzialmente migliori, almeno in teoria—occorre sostanzialmente *conoscere* la soluzione ottima per poter stabilire se una qualsiasi soluzione ammissibile non lo è, ossia utilizzare informazione *globale* sul problema. Questo è in drammatico contrasto con la semplicità della definizione $C(\bar{x})$ che risulta dalla Proprietà 3.1, che usa esclusivamente informazione *locale* al punto $\bar{x}$: l'insieme $I(\bar{x})$ dei vincoli attivi. In effetti, la Figura 4.2 mostra come la Proprietà 3.1 darebbe $\mathbb{R}^n$ come insieme di direzioni ammissibili, in quanto $I(\bar{x}) = \emptyset$, ossia la definizione "facile" non dia assolutamente alcuna indicazione utile rispetto al problema. Per riassumere, la convessità implica che l'oggetto critico al fine di determinare l'ottimalità di una soluzione—l'insieme delle direzioni di crescita—sia definibile utilizzando solamente informazione *locale* (i vincoli attivi del poliedro), proprietà che fallisce in modo drammatico quando si perde la convessità della regione ammissibile.

Un diverso modo di illustrare il concetto consiste nello "scambiare il ruolo della funzione obiettivo ed i vincoli". Nella discussione della *PLI*, la funzione obiettivo è "molto semplice" mentre sono "complicati" i vincoli. Consideriamo invece la minimizzazione di una generica funzione $f : \mathbb{R}^n \to \mathbb{R}$ su tutto lo spazio $x \in \mathbb{R}^n$: i vincoli sono quindi "molto semplici", ma la funzione obiettivo può essere "arbitrariamente complessa". In termini di problema di ottimizzazione, il punto critico in questo caso è se $f(\cdot)$ sia oppure no *convessa*; senza entrare troppo nel dettaglio, $f(\cdot)$ è convessa se lo è il suo *epigrafo* $epi(f) = \{(v, x) \in \mathbb{R}^{n+1} : v \geq f(x)\}$. In generale, minimizzare una funzione *non* convessa è difficile quanto minimizzare (o massimizzare) una funzione lineare su un insieme non convesso. Infatti, è possibile riformulare (quasi) tutti i problemi di *PLI* come problemi di minimizzazione di funzioni non convesse; si noti che abbiamo visto esempi della trasformazione inversa, cioè da problemi di minimizzazione non convessa a problemi di *PLI*, nel Capitolo 1.

Esercizio 4.1. Si dimostri formalmente l'equivalenza tra la *PLI* e la minimizzazione di una funzione non convessa (suggerimento: si determini un *singolo vincolo non lineare* equivalente alla richiesta $x \in \{0, 1\}$, si estenda l'idea a $x \in \{0, 1\}^n$ e poi a $x \in \mathbb{Z}^n$, ed infine si sostituisca il vincolo con un'adeguata modifica della funzione obiettivo, sotto opportune ipotesi se necessario).

In Figura 4.3 sono mostrate un esempio di funzione convessa (a) e uno di funzione non convessa (b) in una sola variabile: con x^* sono indicati i *minimi globali* delle funzioni, ossia i punti cui corrisponde la minimizzazione di tali funzioni. Con x_1 ed x_2 sono indicati due *minimi locali* della funzione non convessa che non sono anche minimi globali; invece, tutti i minimi locali di qualsiasi funzione convessa sono anche minimi globali. In generale, determinare un minimo locale di una funzione con opportune proprietà di regolarità—ad esempio differenziabile con continuità—è "facile": l'informazione al primo ordine sulla funzione indica "da che parte andare". Non possiamo in questo contesto scendere in ulteriori dettagli, ma richiamiamo almeno il caso elementare che dovrebbe essere a tutti noto: per $f : \mathbb{R} \to \mathbb{R}$ che abbia la derivata prima continua, in qualsiasi punto di minimo locale $\bar{x}$ risulta $f'(\bar{x}) = 0$. Per le funzioni convesse (differenziabili), questo accade solo nel minimo globale x^*, per cui si ha—analogamente a quanto visto in precedenza—un risultato molto forte per cui una proprietà *locale* di x^* (la derivata nulla) ne implica una *globale* (l'ottimalità). In altri termini, risolvere il problema richiede solamente di determinare punti a derivata nulla, compito per il quale esistono algoritmi efficienti. Quando $f(\cdot)$ è non convessa, però, la proprietà locale $f'(\bar{x}) = 0$ non implica neanche che $\bar{x}$ sia un minimo locale—basta guardare la Figura 4.3(b)—e quindi men che mai che lo sia globale. Pertanto, una volta determinato un punto a derivata nulla—ed anche ammesso che sia un minimo locale, il che non è detto—non si ha nessuna indicazione sull'esistenza di altri minimi locali (migliori) e sulla loro posizione. Sono quindi disponibili algoritmi in grado di determinare efficientemente minimi locali, che *potrebbero* anche essere globali, ma questi non danno alcuna garanzia

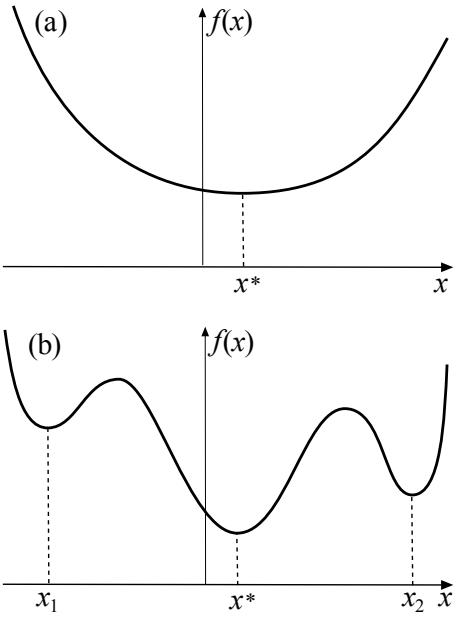


Figura 4.3: Funzioni convesse e non

di risolvere all'ottimo il problema. Questi concetti saranno ripresi nel Capitolo 5. Determinare quello tra i (potenzialmente molti) minimi locali è anche il minimo globale richiede approcci del tutto diversi, per i quali non si conoscono tecniche efficienti.

Torniamo adesso alla *PLI*. Abbiamo visto che determinare l'ottimalità di una soluzione $\bar{x}$ sia "difficile" in generale, e pertanto è di interesse sviluppare metodi in grado di farlo "almeno approssimativamente". Per questo si può sfruttare il fatto che, dato che $\mathcal{F}$ è contenuto in $\bar{\mathcal{F}}$, quest'ultimo fornisce "un'approssimazione" di $\mathcal{F}$. Si consideri infatti il *rilassamento continuo* di (PLI)

$$(RC) \quad \max\{\, cx \,:\, Ax \leq b \,\} \ ,$$

cioè il problema di *PL* corrispondente al rilassamento dei vincoli di integralità $x \in \mathbb{Z}^n$. Questo problema può essere efficientemente risolto—poiché la sua regione ammissibile $\bar{\mathcal{F}}$ è convessa, a differenza di $\mathcal{F}$—e ciò permette di derivare informazione sul problema originario. In particolare, poiché (RC) è un *rilassamento* di (PLI)—si ricordi la definizione generale (1.8) data nel §1.1—il suo valore ottimo fornisce una valutazione superiore del valore ottimo del problema originario, ossia $z(RC) \geq z(PLI)$. L'utilità di questa relazione risiede ovviamente nel fatto che $z(RC)$, al contrario di $z(PLI)$, è efficientemente calcolabile. Inoltre, è immediato verificare il seguente risultato:

Lemma 4.1. Sia x^* una soluzione ottima di (RC): se $x^* \in \mathbb{Z}^n$ allora x^* è ottima per (PLI).

Dimostrazione Poiché x^* è ottima per (RC) allora è in particolare ammissibile, ossia $Ax^* \leq b$. Ma poiché per ipotesi $x^* \in \mathbb{Z}^n$ allora è anche ammissibile per (PLI), e di conseguenza $cx^* \leq z(PLI)$. Ma (RC) è un rilassamento di (PLI) ed x^* è la sua soluzione ottima, da cui $cx^* \leq z(PLI) \leq z(RC) = cx^*$, ossia x^* è ottima per (PLI). $\diamond$

Il Lemma mostra che in questo caso "fortunato" cx^* è sia una valutazione superiore che una inferiore su $z(PLI)$—torneremo sull'importanza di questi concetti nel §4.3—e quindi lo identifica: la prima priprietà è sempre vera, mentre la seconda richiede $x^* \in \mathbb{Z}^n$, ossia l'ammissibilità Un caso in cui si verificano le condizioni del Lemma 4.1 è mostrato in Figura 4.1: è immediato verificare geometricamente che la soluzione x^* indicata è la soluzione ottima del rilassamento continuo, è intera ed è quindi anche la soluzione ottima del problema di *PLI*. Ciò è intuitivamente collegato con il fatto che x^* è un vertice del poliedro, a differenza di ciò che accade in Figura 4.2 ove x^* è interna al poliedro. Il rilassamento continuo fornisce quindi un modo per tentare di calcolare una soluzione ottima per (PLI), ed in ogni caso fornisce una valutazione superiore del valore $z(PLI)$.

Esempio 4.2. Valutazioni superiori ed inferiori
Si consideri ad esempio il problema della Pintel: abbiamo visto che il suo rilassamento continuo ha soluzione ottima $[\,4\,,\,1\,]$ (in unità di 100000 processori). Se consideriamo il vincolo di integralità sui wafers tale soluzione non è ammissibile, ma ci fornisce comunque una stima per eccesso del massimo ricavo disponibile, pari a 220 milioni di dollari. Si consideri adesso la soluzione ammissibile $[\,w_P\,,\,w_C\,] = [\,2666\,,\,334\,]$ corrispondente a $[\,x_P\,,\,x_C\,] = [\,3.995\,,\,1.002\,]$: tale soluzione permette un ricavo di 219.99 milioni di dollari. Possiamo quindi affermare che la soluzione ammissibile fornisce, alla peggio, un ricavo inferiore di 10000\$ al massimo possibile, ossia più del 99.995% del massimo ricavo possibile. Per la maggior parte degli scopi pratici, determinare una soluzione di questo tipo può essere considerato equivalente ad aver risolto il problema. Si noti che l'aver determinato la soluzione non è di per sé sufficiente: quello che permette di "esserne soddisfatti" è l'essere in grado di valutarne la "qualità", il che è reso possibile dalla valutazione superiore fornita dal rilassamento continuo.

In generale, però, non sempre verificano le condizioni del Lemma 4.1, o anche dell'Esempio 4.2, ossia la soluzione ottima del rilassamento fornisce una valutazione superiore esatta o almeno molto accurata del valore ottimo; in molti casi la valutazione può essere "molto inaccurata".

Esercizio 4.2. Si costruisca una classe di problemi di *PLI* in cui il gap tra il valore ottimo del problema e la valutazione superiore del rilassamento continuo può essere arbitrariamente grande.

Resta però il fatto che "nelle giuste condizioni" il rilassamento continuo possa risolvere il problema di *PLI* originario. Discuteremo adesso il fatto che *in teoria* sarebbe sempre essere possibile essere in queste condizioni.

4.2.2 Formulazioni di PL equivalenti per la PLI

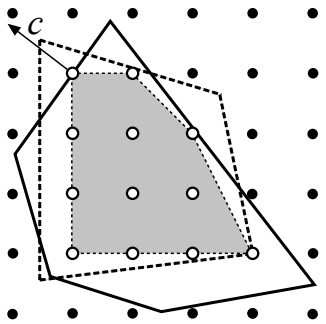


Figura 4.4: Formulazioni diverse

Un'osservazione cruciale è che *lo stesso insieme* ammissibile per un problema di *PLI* può essere specificato attraverso *poliedri diversi*. Ciò è mostrato in Figura 4.4, dove il poliedro mostrato in tratto continuo definisce lo stesso insieme di soluzioni ammissibili di quello delle Figure 4.1 e 4.2 (tratteggiato), pur essendo chiaramente "del tutto diverso". Esistono quindi *formulazioni diverse di uno stesso problema di PLI*. Queste formulazioni sono equivalenti per quanto riguarda il problema di *PLI*, ma non per quanto riguarda i rilassamenti continui. Risulta infatti chiaro geometricamente come il poliedro di Figura 4.4 soddisfi le condizioni del Lemma 4.1 per la funzione obiettivo di Figura 4.2, mentre come abbiamo visto quello originale non le soddisfaceva. Questa dipende anche da c, in quanto il poliedro di Figura 4.1 è "buono" se la funzione obiettivo è quella della figura ma "non buono" se è quella della Figura 4.4, e viceversa: nessuno dei due poliedri è "buono" per l'altra funzione obiettivo.

È intuitivo che la qualità della valutazione superiore di $z(PLI)$ fornita da $z(RC)$ sia tanto migliore quanto più il poliedro $\bar{\mathcal{F}}$ risulti "aderente" all'insieme ammissibile $\mathcal{F}$ di (PLI) "dalla parte in cui si trova la soluzione ottima": sostanzialmente, si ottiene l'ottimo quando questo sta "sul bordo" del poliedro, ossia questo "aderisce bene" all'insieme $\mathcal{F}$ almeno "nelle vicinanze di x^*". In effetti si può dimostrare che tra tutti i rilassamenti continui (ossia formulazioni) ne esiste sempre uno "buono" per *qualsiasi* funzione obiettivo, in quanto "completamente aderente" a $\mathcal{F}$. Tecnicamente questo risultato richiede che tutti gli elementi della matrice A e del vettore b siano razionali, ma questi sono comunque i numeri che i calcolatori digitali trattano efficientemente: quando si implementano questi algoritmi normalmente gli elementi A_{ij} e b_i sono numeri `double`, ossia a virgola mobile a 64 bits, e quindi hanno una precisione finita. Mediante sistemi di calcolo simbolico è possibile trattare anche numeri irrazionali ma con un'efficienza enormemente minore, il che ne limita fortemente l'uso in applicazioni come quelle descritte in queste note; pertanto, l'assunzione che $A \in \mathbb{Q}^{m\times n}$ e $b \in \mathbb{Q}^m$ può essere fatta quasi senza perdita di generalità. Con tale assunzione si può mostrare che *l'inviluppo convesso* di $\mathcal{F}$

$$\tilde{\mathcal{F}} = Conv(\,\mathcal{F}\,)$$

è un poliedro, cioè può essere rappresentato da un sistema finito di vincoli lineari $\tilde{A}x \le \tilde{b}$. In Figura 4.4 l'inviluppo convesso dei punti ammissibili è rappresentato dal poliedro tratteggiato ed ombreggiato, interno ad entrambi quelli precedentemente discussi. Senza entrare nel dettaglio delle dimostrazioni, si può visualizzare in generale $\tilde{\mathcal{F}}$ come il più piccolo (nel senso dell'inclusione) poliedro che contiene tutti i punti di $\mathcal{F}$. Per evitare complicazioni tecniche, nel seguito assumeremo che $\mathcal{F}$ sia *compatto*, ossia contenuto in una sfera di raggio finito intorno all'origine, ossia formato da un numero finito di punti; pertanto anche $\tilde{\mathcal{F}}$ sarà compatto, ossia $rec(\,\tilde{\mathcal{F}}\,) = (\,0\,)$, e quindi per il Teorema 3.4 pari all'inviluppo convesso dei suoi punti estremi. Di conseguenza, tutti i punti estremi di $\tilde{\mathcal{F}}$ (che esistono) devono essere elementi di $\mathcal{F}$, ossia punti a coordinate intere.

Quanto appena detto sembrerebbe contraddire l'affermazione secondo cui i problemi di *PLI* sono difficili. Infatti, il problema

$$(\widetilde{RC}) \quad \max\{\, cx \,:\, \tilde{A}x \le \tilde{b}\,\}$$

è chiaramente un rilassamento di (PLI), in quanto per definizione $\mathcal{F} \subset \tilde{\mathcal{F}}$, ma è possibile dimostrare che $\tilde{\mathcal{F}}$ gode della seguente *proprietà di integralità*:

Definizione 4.1. Un poliedro $\mathcal{P} = \{\, x \in \mathbb{R}^n \,:\, Ax \le b\,\}$, non vuoto e tale che $rango(\,A\,) = n$, gode della *proprietà di integralità* se vale una delle due seguenti definizioni equivalenti:

- tutti i vertici di $\mathcal{P}$ hanno coordinate intere;
- il problema $\max\{\, cx \,:\, x \in \mathcal{P}\,\}$ ammette una soluzione ottima intera per *qualsiasi* scelta del vettore $c \in \mathbb{R}^n$ per cui il problema non risulti superiormente illimitato.

Notiamo innanzi tutto che la richiesta che $rango(\,A\,) = n$ è giustificata dal fatto che altrimenti non esistono i vertici, e quindi le due definizioni non possono essere equivalenti; in ogni caso stiamo

discutendo solamente il caso dei poliedri compatti, che quindi non possono avere direzioni di linealità. Per dimostrare l'equivalenza delle due condizioni, un'implicazione è ovvia: il problema non è vuoto ed il poliedro ammette punti estremi, quindi per qualsiasi c tale che il problema ha valore ottimo finito esiste almeno una soluzione ottima che è un punto estremo di $\mathcal{P}$, e pertanto appartiene a $\mathbb{Z}^n$. Per dimostrare l'implicazione inversa procediamo per assurdo: assumiamo che esista un punto estremo $\bar{x}$ di $\mathcal{P}$ tale che $\bar{x} \notin \mathbb{Z}^n$, ma sia comunque vera la seconda condizione. Per l'equivalenza tra punti estremi e vertici, esiste almeno una base B tale che $\bar{x}$ è la soluzione di base corrispondente: in altri termini, la matrice di base A_B è invertibile e $\bar{x} = A_B^{-1} b_B$. Essendo $u \in \mathbb{Z}^n$ il vettore avente tutte le componenti pari ad 1, si consideri la funzione obiettivo $c = uA_B$: la soluzione duale di base corrispondente alla base B è

$$\bar{y} = \left[\; \bar{y}_B \, , \, \bar{y}_N \; \right] = \left[\; cA_B^{-1} \, , \, 0 \; \right] = \left[\; u \, , \, 0 \; \right] \geq 0 \; .$$

Pertanto, la base B è primale ammissibile e duale ammissibile per il problema che ha funzione obiettivo c, e quindi $\bar{x}$ è una soluzione primale ottima del problema. Ma poiché la base è duale non degenere ($\bar{y}_B = u > 0$), per il teorema degli scarti complementari $\bar{x}$ è l'*unica* soluzione primale ottima del problema; poiché $\bar{x} \notin \mathbb{Z}^n$, questo contraddice il fatto che tutte le soluzioni ottime del problema siano intere, dimostrando pertanto l'equivalenza.

Tutto ciò mostra che *qualora si conoscesse* $\tilde{\mathcal{F}}$ sarebbe possibile ottenere una soluzione ottima di (PLI) risolvendo ($\widetilde{RC}$), che è un problema di PL. Questo richiede in realtà di saper determinare una soluzione ottima di ($\widetilde{RC}$) che sia un vertice di $\tilde{\mathcal{F}}$, ma sappiamo che questo tipicamente non è un problema per gli algoritmi. Questo parrebbe contraddire il fatto che la PLI sia $\mathcal{NP}$-hard, in quanto la PL è invece un problema polinomiale (ancorché in queste dispense non vengano presentati algoritmi con una dimostrabile complessità polinomiale). In effetti i problemi di PLI per i quali sono disponibili formulazioni che hanno la proprietà di integralità sono usualmente "facili".

Ad esempio, abbiamo visto (cf. Teorema 2.12) che la formulazione "naturale" del problema di Flusso di Costo Minimo, e quindi di tutti i problemi del Capitolo 2 che ad esso possono essere ricondotti, gode della proprietà di integralità purchè le capacità degli archi e i deficit dei nodi siano interi. Quindi, è il fatto di trovarsi "al confine" tra l'ottimizzazione discreta e quella continua—ossia che la regione ammissibile, pur essendo discreta e quindi non convessa, sia rappresentabile in modo esatto attraverso una regione ammissibile convessa e quindi continua—che rende un problema chiaramente combinatorio come ad esempio (SPT) "facile". Per il generico problema di PLI, invece, la rappresentazione di $\tilde{\mathcal{F}}$, ossia l'insieme dei vincoli $\tilde{A}x \leq \tilde{b}$ che lo definiscono, non è nota: tutto quello di cui si dispone è la sua rappresentazione "approssimata" data dai vincoli $Ax \leq b$. Abbiamo però detto che una formulazione esatta, basata sull'inviluppo convesso, esiste: dobbiamo quindi capire come mai la sua esistenza non sia sufficiente a rendere la PLI "facile".

La prima osservazione, che però come vedremo è solo parzialmente pertinente, è che il fatto che $\tilde{\mathcal{F}}$ esista non implica necessariamente che sia "di dimensioni contenute". Al contrario, come abbiamo già visto per l'eliminazione di Fourier-Motzkin, può succedere—ed anzi è il caso tipico—che l'inviluppo convesso di un problema esprimibile attraverso una formulazione PLI "compatta" abbia un numero esponenziale di vincoli.

Esempio 4.3. L'n-co-cubo

Avevamo già visto nell'Esempio 3.10 un poliedro che richiede un numero esponenziale di vincoli per essere espresso, almeno nello spazio delle variabili naturali: si tratta dell'n-co-cubo, ossia la palla di raggio unitario nella norma L_1. Tale poliedro ha $2n$ vertici della forma $[\, 0 \, , \, \ldots \, , \, \pm 1 \, , \, \ldots \, 0 \,]$ per tutti gli n modi possibili di scegliere la posizione del ± 1, e quindi possiede la proprietà di integralità. Ovviamente (come discuteremo meglio nel seguito) ottimizzare una funzione lineare sull'n-co-cubo è "facile" (banale).

L'esempio precedente può sembrare un po' "artificiale", ma situazioni analoghe si hanno per altri problemi di OC singnificativi.

Esempio 4.4. Diverse formulazioni di (MST)

Per il problema (MST), introdotto nell'Esempio 1.6, ed discusso in dettaglio nel §2.3, una formulazione PLI "compatta" (cf. Esercizio 2.18) può facilmente essere costruita con le tecniche viste nel §1.2. In particolare, dato il grafo

non orientato $G = (V, E)$ del problema costruiamo il "classico" (cf. per esempio l'Esercizio 1.27) grafo orientato $G' = (V, A)$ con gli stessi nodi (vertici) di G e tale che A contiene i due archi (i, j) e (j, i) per ciascun lato $\{i, j\} \in E$. Introduciamo le classiche variabili $x_{ij} \in \{0, 1\}$ per $\{i, j\} \in E$ per identificare i lati dell'albero, ma anche variabili di flusso f_{ij} sugli archi di G'. Scegliamo arbitrariamente un nodo radice, ad esempio il nodo 1, e lo designamo come sorgente di $n-1$ unità di flusso, mentre tutti gli altri sono pozzi e richiedono un'unità di flusso: questo significa che in qualsiasi flusso ammissibile ci sarà almeno un cammino (orientato, ma in realtà dato che G' è il bi-orientamento di G nei fatti non orientato) tra 1 e tutti gli altri nodi, e quindi il flusso identifica un sottografo connesso. Utilizziamo le tecnica vista nel §1.2.8 per implementare la condizione logica "se c'è flusso su almeno uno degli archi (i, j) e (i, j) allora si deve selezionare il lato $\{i, j\}$", ottenendo la formulazione

$$
\text{(MST)} \qquad
\begin{array}{lll}
\min & \sum_{\{i,j\} \in E} c_{ij} x_{ij} & \\
& \sum_{(j,i) \in BS(i)} f_{ji} - \sum_{(i,j) \in FS(i)} f_{ij} = \begin{cases} -(n-1) & i = 1 \\ 1 & i \neq 1 \end{cases} & i \in V \\
& f_{ij} + f_{ji} \leq (n-1) x_{ij} & \{i, j\} \in E \\
& f_{ij} \geq 0 & (i, j) \in A \\
& x_{ij} \in \{0, 1\} & \{i, j\} \in E
\end{array}
$$

che ha $3m$ variabili (di cui m binarie) e $n + m$ vincoli (senza contare quelli di segno e di integralità), ed è quindi "compatta" nel senso di avere una dimensione polinomiale. Sfortunatamente, è facile vedere che questa formulazione non è esatta, ossia non rappresenta l'inviluppo convesso delle soluzioni intere del problema (MST).

Esercizio 4.3. Si determini una semplice istanza di (MST) per cui si possa risolvere in modo ovvio il rilassamento continuo della formulazione precedente e la soluzione ottima sia frazionaria nelle x_{ij}.

Abbiamo però visto nell'Esempio 1.6 la formulazione "per tagli" del problema

$$
\text{(MST)} \qquad \min \left\{ \sum_{\{i,j\} \in E} c_{ij} x_{ij} \; : \; \sum_{i \in S, j \notin S} x_{ij} \geq 1 \quad \emptyset \subset S \subset V \;\; , \;\; x_{ij} \in \{0, 1\} \quad \{i, j\} \in E \right\} \; .
$$

Tale formulazione ha solo le m variabili binarie "naturali", ma ha un numero esponenziale di vincoli. Come vedremo nel §5.1.3, questa formulazione è "esatta", ossia esprime l'inviluppo convesso dei punti interi del problema. Pertanto, anche un problema "facile" come (MST) può richiedere un numero esponenziale di vincoli per costruire una formulazione "esatta", almeno nello spazio delle variabili naturali.

Gli esempi precedenti mostrano che può essere necessario utilizzare un numero esponenziale di vincoli per costruire formulazioni "esatte" anche per problemi "facili". Questo indica come una formulazione "di grandi dimensioni" non necessariamente corrisponda al fatto che il problema sia "difficile", e quindi che l'aumento del numero di vincoli da solo non è necessariamente indicativo della difficoltà del problema. Infatti, vedremo nel prossimo paragrafo come, sotto opportune ipotesi, sia in effetti possibile risolvere efficientemente (anche in tempo dimostrabilmente polinomiale) problemi che hanno una formulazione di dimensione esponenziale. Ciò non implica però che questo sia possibile per tutti i problemi, piuttosto il contrario.

4.2.3 Diseguaglianze valide

Per approfondire la discussione dobbiamo adesso introdurre formalmente un concetto molto importante, ancorché apparentemente semplice. Una diseguaglianza $dx \leq \delta$ si dice *valida* per $\tilde{\mathcal{F}}$ se è soddisfatta da ciascun punto $x \in \tilde{\mathcal{F}}$; equivalentemente, si può dire che $dx \leq \delta$ è valida per (PLI) se è soddisfatta da tutte le soluzioni (intere) del problema. Si noti che tutti i vincoli $\tilde{A}_i x \leq \tilde{b}_i$ che definiscono $\tilde{\mathcal{F}}$ sono per definizione diseguaglianze valide; in generale sarebbe desiderabile che $\tilde{A}$ e $\tilde{b}$ formino una rappresentazione *minimale* di $\tilde{\mathcal{F}}$, ossia che la rimozione di una qualsiasi di queste diseguaglianze definisca un poliedro che contiene strettamente $\tilde{\mathcal{F}}$. In altri termini, con la nomenclatura del §3.1.1.1, si vorrebbe che ciascun vincolo $\tilde{A}_i x \leq \tilde{b}_i$ nella rappresentazione di $\tilde{\mathcal{F}}$ definsca una *faccetta*, o faccia massimale, del poliedro; questo in generale non è sempre possibile, ma ciò non è particolarmente rilevante per la nostra discussione.

La fondamentale osservazione di questo paragrafo è che in principio—ed anche in pratica—non è necessario disporre di *tutti* i vincoli $\tilde{A}_i x \leq \tilde{b}_i$ necessari alla rappresentazione di $\tilde{\mathcal{F}}$ per essere in grado di risolvere $(\widetilde{RC})$. Infatti, è sufficiente essere in grado di risolvere il seguente *problema di separazione* (un problema di decisione)

Dato $\bar{x} \in \mathbb{R}^n$, esiste una diseguaglianza valida $dx \leq \delta$ per $\tilde{\mathcal{F}}$
che non è soddisfatta da $\bar{x}$, ossia tale che $d\bar{x} > \delta$?

Il problema di separazione chiede sostanzialmente di determinare se un dato punto $\bar{x}$ appartiene o no a $\tilde{\mathcal{F}}$. Nel caso in cui $\bar{x} \notin \tilde{\mathcal{F}}$ viene richiesta una "dimostrazione" della non appartenenza sotto forma di una diseguaglianza che separa $\bar{x}$ da $\tilde{\mathcal{F}}$. Una tale diseguaglianza viene anche detta un *taglio* per $\bar{x}$. L'utilità del problema di separazione deriva dal seguente fatto: anche se la rappresentazione di $\tilde{\mathcal{F}}$ contiene un numero esponenziale di vincoli, *è possibile risolvere* $(\widetilde{RC})$ *con un numero polinomiale (o comunque "piccolo" in pratica) di chiamate all'oracolo di separazione.* Per capire il perché è sufficiente considerare l'*algoritmo del simplesso duale meccanizzato*:

```
procedure ( x̄ , ȳ , stato ) = Simplesso_Duale_Meccanizzato( A , b , c , oracolo ) {
for( stato = "" ; ; ) {    // invariante: A invertibile, cA^{-1} ≥ 0
  x̄ = A^{-1}b; ȳ = cA^{-1};
  [ d , δ , stato ] = oracolo( x̄ );
  if( stato == "ottimo" )then break;
  η = dA^{-1};
  if( η ≤ 0 ) then { stato = "P.vuoto"; break; }
  θ̄ = min{ θ_i = ȳ_i/η_i : η_i > 0 , i = 1,...,n };
  h = min{ i = 1,...,n : θ_i = θ̄ }; A_h = d; b_h = δ;
  }
}
```

Procedura 4.1: Simplesso Duale Meccanizzato

Il simplesso duale meccanizzato è semplicemente l'algoritmo del simplesso duale (cf. §3.3.2) riscritto per il caso in cui non si ha accesso diretto ai dati A e b del problema (nella nostra situazione, perchè sono "troppo grandi") ma si ha accesso ad un oracolo di separazione per il poliedro che rappresentano. Per semplicità assumiamo che in input sia fornita una matrice di base $A \in \mathbb{R}^{n\times n}$ ed il corrispondente vettore dei termini noti $b \in \mathbb{R}^n$, duale ammissibile: si noti che questo è sempre possibile—oltreutto con A e b "molto semplici"—per via delle tecniche discusse nel §3.3.2.1. A questo punto l'algoritmo procede al calcolo della coppia di soluzioni primale e duale di base complementari (tranne evitare di scrivere $\bar{y}_N$, che è inutile e comunque "troppo lungo") ed a valutare l'ammissibilità di $\bar{x}$: quest'ultima operazione viene però compiuta dall'oracolo di separazione. Se $\bar{x}$ è ammissibile, ossia non si determina alcun taglio, l'oracolo ritorna *stato* = "ottimo" e l'algoritmo termina avendo determinato una coppia di soluzioni complementari primali e duali ammissibili, ossia ottime per (P) e (D). Altrimenti l'oracolo ritorna *stato* = "" e la rappresentazione $(\, d\,,\, \delta\,)$ di un taglio valido. In questo caso l'algoritmo procede come il simplesso duale a determinare la direzione di crescita duale—ovviamente solamente la parte relativa ai vincoli in A—e quindi quale dei vincoli nella base debba essere sostituito da quello appena generato, oppure se si sia determinata una direzione ammissibile di decrescita illimitata per (D) e quindi se si sia dimostrato che (P) è vuoto.

Chiaramente questo algoritmo termina in un numero finito di iterazioni se i tagli che l'oracolo riporta sono tratti da un insieme finito, per quanto arbitrariamente grande (esistono anche i problemi di *PL semi-infinita* che hanno un numero infinito di vincoli, e la tecnica sostanzialmente funziona anche in quel caso, ma non entreremo nel dettaglio). L'algoritmo non ha garanzie di terminare in un numero polinomiale di iterazioni, visto che non ce l'ha neanche il simplesso duale "normale", ma ci si può aspettare che sia efficiente nella pratica. In effetti lo si può rendere ancora più efficiente in diversi modi, il più semplice essendo quello di partire con una rappresentazione "approssimata" $A^{(0)}x \leq b^{(0)}$ di $\tilde{\mathcal{F}}$, ad esempio quella fornita dal sistema $Ax \leq b$, e di risolvere il corrispondente rilassamento continuo. Si passa in input all'oracolo di separazione la soluzione ottima $\bar{x}^{(0)}$ determinata: se $\bar{x}^{(0)} \in \tilde{\mathcal{F}}$ allora è una soluzione ottima per $(\widetilde{RC})$, e quindi $cx^{(0)} = z(PLI)$, altrimenti viene determinato un taglio $dx \leq \delta$ per $\bar{x}^{(0)}$ che viene aggiunto ai vincoli già noti, ossia

$$A^{(1)} = \begin{bmatrix} A^{(0)} \\ d \end{bmatrix} \qquad \text{e} \qquad b^{(1)} = \begin{bmatrix} b^{(0)} \\ \delta \end{bmatrix} \ ,$$

ottenendo una nuova rappresentazione $A^{(1)}$ e $b^{(1)}$ di $\tilde{\mathcal{F}}$ "meno approssimata" della precedente; infatti, almeno il punto $x^{(0)}$ non ne fa più parte. Risolvendo il problema di PL corrispondente a questo

nuovo sistema si otterrà una nuova soluzione $\bar{x}^{(1)} \neq \bar{x}^{(0)}$, e si potrà iterare il procedimento fino a determinare una soluzione ottima di $(\widetilde{RC})$. Chiaramente questa versione è sostanzialmente un "simplesso duale meccanizzato con cache": i vincoli generati dall'oracolo non vengono immediatamente perduti al momento in cui escono di base, ma vengono conservati per il caso in cui tornino ad essere rilevanti nelle iterazioni successive. Questo può diminuire significativamente il numero di chiamate all'oracolo necessarie a risolvere il problema, il che può essere importante perché—come vedremo—l'oracolo stesso può essere computazionalmente (molto) costoso. Inoltre il metodo può partire già da una "buona approssimazione" di $\tilde{\mathcal{F}}$, il che può contribuire a ridurre ancora sostanzialmente le prestazioni. Comunque, dal punto di vista concettuale questo *algoritmo dei piani di taglio di Kelley* rimane fondamentalmente un simplesso duale meccanizzato.

Gli approcci appena discussi non garantiscono formalmente che il numero di chiamate all'oracolo sia polinomiale (in n), ma con tecniche diverse è possibile ottenere tale garanzia. In effetti è possibile dimostrare che un algoritmo in parte analogo a quelli descritti, detto *metodo degli ellissoidi*, è in grado di determinare una soluzione ottima del problema con un numero polinomiale di chiamate all'oracolo di separazione. I dettagli del metodo sono altamente complessi—si tratta in effetti di un monumentale risultato della matematica—e per essi si rimanda alla letteratura citata, anche se è corretto rimarcare che il comportamento in pratica del metodo degli ellissoidi è molto peggiore di quello delle più semplici tecniche illustrate in precedenza, che quindi sono quelle che si usano nelle applicazioni.

Ovviamente, tutto ciò che è stato discusso finora sarebbe di scarso interesse se il funzionamento del separatore fosse di scorrersi la lista—esponenzialmente lunga—dei vincoli che definiscono $\tilde{\mathcal{F}}$, poiché questo risulterebbe comunque in un algoritmo di costo esponenziale anche a fronte di un numero polinomiale (in effetti, anche a fronte di un numero costante) di chiamate all'oracolo. Pertanto, è possibile risolvere in tempo polinomiale—e normalmente efficientemente in pratica—un problema di *PLI* se per il suo inviluppo convesso $\tilde{\mathcal{F}}$, anche se definito da un numero esponenziale di disequazioni, si disponga di un *separatore polinomiale* per $\mathcal{P}$, ossia si sia in grado di risolvere il corrispondente problema di separazione in tempo polinomiale.

Esempio 4.5. Un separatore per l'n-co-cubo

Costruiamo adesso un separatore per la regione ammissibile definita dall'n-co-cubo, come nell'Esempio 4.3. È dato in input un vettore $\bar{x} \in \mathbb{R}^n$, e si vuole determinare se esiste un modo di scegliere i segni dei coefficienti nella generica disuguaglianza che definisce l'n-co-cubo

$$\pm x_1 \pm x_2 \ldots \pm x_n \leq 1$$

tale che la corrispondente diseguaglianza sia violata da $\bar{x}$. Per fare questo basta notare che il lato destro del vincolo è fisso (1): pertanto il problema di decisione può essere ricondotto al problema di ottimizzazione

$$(SEPncc) \quad \max\{ \pm\bar{x}_1 \pm \bar{x}_2 \ldots \pm \bar{x}_n \} \ .$$

Se $z(SEPncc) \leq 1$, il lato sinistro del vincolo è ≤ 1 per qualsiasi scelta dei segni dei coefficienti, e quindi tutti i 2^n vincoli sono rispettati. Altrimenti, la scelta ottima dei segni dei coefficienti porta ad un particolare vincolo della famiglia che ha lato sinistro > 1 in $\bar{x}$, e quindi rappresenta la risposta dell'oracolo. Il problema $(SEPncc)$ si risolve banalmente in tempo lineare: è immediato verificare che basta, per ogni i, scegliere il segno $+$ se $\bar{x}_i > 0$ ed il segno $-$ se $\bar{x}_i < 0$ (la scelta è indifferente e quindi arbitraria se $\bar{x}_i = 0$). Pertanto, esiste un separatore polinomiale (lineare) per l'n-co-cubo, e quindi il problema di ottimzzare una qualsiasi funzione lineare su questo insieme è polinomiale. Ciò è assolutamente non sorprendente, in quanto la soluzione ottima stessa si ottiene banalmente in tempo lineare.

Esercizio 4.4. Si descriva una procedura che determina in $O(n)$ il massimo di una qualsiasi funzione lineare cx sull'n-co-cubo.

Esempio 4.6. Un separatore per (MST)

Anche la formulazione "per tagli" di (MST) (cf. l'Esempio 4.4) che usa le disuguaglianze (1.12), dette *cutset inequalities*, ammette un separatore polinomiale. Si consideri infatti una soluzione $\bar{x}$ (possibilmente frazionaria): vogliamo verificare se esiste $S \subseteq V$ a cui corrisponde un vincolo è violato, ossia tale che risulti

$$\textstyle\sum_{i \in S \, , \, j \notin S} \bar{x}_{ij} < 1 \ .$$

Per determinarlo, analogamente al caso precedente, si può risolvere un problema di ottimizzazione: poiché il lato sinistro del vincolo è fissato (a 1), è sufficiente determinare il sottoinsieme S a cui corrisponde il minimo del lato

sinistro. Ma il lato sinistro del vincolo corrisponde alla *capacità del taglio* $(S, V \setminus S)$ se $\bar{x}_{ij}$ è interpretata come la capacità del lato $\{i, j\}$. Supponiamo quindi di determinare il taglio (V', V'') di capacità minima rispetto a quelle capacità: se la capacità del taglio è minore di 1 allora abbiamo individuato una specifica disuguaglianza violata, mentre se è maggiore od uguale ad 1 allora non esiste nessuna disuguaglianza violata. Il problema del taglio di capacità minima può essere risolto in tempo polinomiale (si veda il §2.4), e quindi esiste un separatore polinomiale per le cutset inequalities.

Esercizio 4.5. I problemi di taglio di capacità minima per i quali abbiamo discusso algoritmi risolutivi efficienti sono definiti su grafi orientati e sono relativi ad una specifica coppia di nodi s e t. Si discuta come adattarli alla definizione di un separatore per (MST).

(MST) ammette infatti algoritmi polinomiali (si veda il §2.3), che in pratica risultano notevolmente più efficienti rispetto alla soluzione di una sequenza di problemi di PL e di separazione; l'esempio serve principalmente a sottolineare ancora la fondamentale relazione tra i due concetti.

Gli esempi precedenti hanno mostrato casi in cui è possibile risolvere in maniera efficiente il problema di separazione rispetto a $\tilde{\mathcal{F}}$, e quindi si dispone di un algoritmo per risolvere $(\widetilde{RC})$, e quindi (PLI). In realtà gli algoritmi mostrano sostanzialmente che se (PLI) è facile, allora lo è anche il problema di separazione. Purtroppo la relazione vale anche in senso inverso. Infatti, l'algoritmo degli ellissoidi mostra che è possibile risolvere $(\widetilde{RC})$, e quindi (PLI), in tempo polinomiale se si dispone di un separatore polinomiale per $\tilde{\mathcal{F}}$: pertanto si è dimostrato che

Teorema 4.1. Sia (PLI) una formulazione di un problema di OC $\mathcal{NP}$-arduo: il problema di separazione associato all'inviluppo convesso $\tilde{\mathcal{F}}$ delle corrispondenti soluzioni intere è anch'esso $\mathcal{NP}$-arduo.

Ciò significa che le rappresentazioni $\tilde{\mathcal{F}}$ degli inviluppi convessi di problemi $\mathcal{NP}$-arduo sono necessariamente "complesse": non dispongono di separatori polinomiali (a meno che $\mathcal{P} = \mathcal{NP}$), il che implica che sono di dimensione esponenziale. Il seguente esempio fornisce un supporto a questa conclusione generale.

Esempio 4.7. Una classe di tagli per (KP) ed il loro separatore

Si consideri il problema dello zaino (KP) visto nell'Esempio 1.5, ed in particolare i dati a_i, $i \in I$ e b che ne caratterizzano univocamente la regione ammissibile. La formulazione di PLI naturale usa i vincoli

$$\sum_{i \in I} a_i x_i \leq b \ , \ \ x_i \in [0, 1] \quad i \in I$$

oltre naturalmente a quelli di integralità. È facile vedere che tale formulazione *non* ha la proprietà di integralità considerando la semplicissima istanza

$$\max\{ 3x_1 + 2x_2 \ : \ 2x_1 + 2x_2 \leq 3 \ , \ [x_1, x_2] \in \{0, 1\}^2 \} \ .$$

È evidente che i due oggetti non entrano contemporaneamente nello zaino, e che quindi la soluzione ottima $x^* = [1, 0]$ consiste nello scegliere quello più profittevole, con un valore ottimo pari a 3. Ma è anche facile verificare che questa non è la soluzione ottima del rilassamento continuo, che invece è $\bar{x} = [1, 1/2]$ (si veda il §5.1.2.1 per la dimostrazione) con un valore ottimo pari a 4.

L'esame del semplice esempio suggerisce immediatamente una classe di disuguaglianze valide per il problema (KP) note come *cover inequalities*. Una cover per il problema è semplicemente qualsiasi $S \subset I$ tale che $\sum_{i \in S} a_i > b$, ossia un sottoinsieme di elementi che non possono entrare tutti contemporaneamente nello zaino (almeno uno di tali sottoinsiemi deve esistere, altrimenti il problema è banale). Data una cover S, la cover inequality corrispondente è semplicemente

$$\sum_{i \in S} x_i \leq |S| - 1$$

ed esprime esattamente la condizione che almeno uno degli oggetti in S deve essere lasciato fuori dallo zaino, cosa che deve necessariamente accadere in qualsiasi soluzione intera del problema. Nel nostro semplice esempio, $S = I = \{1, 2\}$ è l'unica cover, e corrisponde alla cover inequality $x_1 + x_2 \leq 1$ che è ovviamente violata dalla soluzione $\bar{x} = [1, 1/2]$ del rilassamento continuo standard. È facile verificare algebricamente e/o geometricamente che aggiungere tale vincolo alla formulazione fa si che la regione ammissibile del rilassamento continuo abbia la proprietà di integralità (si ottiene il simplesso unitario, ossia il "quarto in alto a destra dell'n-co-cubo) e che quindi il rilassamento continuo così rafforzato ha la soluzione ottima $\bar{x} = [1, 0]$, intera e quindi ottima per il problema originale. Si noti che le cover inequalities in generale *non* sono sufficienti a caratterizzare $\bar{\mathcal{F}}$, tranne per valori molto piccoli di n: per questo sono necessarie altre e più complesse famiglie di disuguaglianze valide che non è il caso di discutere in questa sede.

Poniamoci adesso il problema di realizzare un separatore per le cover inequalities: prendendo in input $\bar{x} \in [0, 1]^n$, $a \in \mathbb{R}^n$ e $b \in \mathbb{R}$ si vuole determinare se esiste una cover inequality violata. Si noti che in questo caso il lato destro

del vincolo non è costante, ma può essere reso tale riscrivendo la cover come

$$\sum_{i\in S}(\,1-x_i\,)\geq 1$$

(ossia, ancora "almeno uno degli oggetti non deve stare nello zaino"). Quindi, minimizzando il lato sinistro del vincolo si determina la cover più violata, e si è trovata una cover violata se si ottiene un valore ottimo minore di 1. Di questo problema combinatorio può essere facilmente data una formulazione di *PLI*:

$$(CS_{\bar{x}}) \qquad \min\Big\{\ \textstyle\sum_{i\in I}(\,1-\bar{x}_i\,)z_i\ :\ \sum_{i\in I}a_iz_i>b\ ,\ z_i\in\{\,0\,,\,1\,\}\quad i\in I\Big\}\ .$$

È immediatamente evidente come questo problema sia sostanzialmente equivalente a (KP) stesso. Il fatto che il vincolo sia di maggiore stretto parrebbe essere problematico, ma per semplificare la discussione assumiamo che tutti gli a_i siano numeri interi, e quindi anche che b lo sia (altrimenti basta arrotondarlo all'intero superiore): il vincolo "$\ldots>b$" può quindi essere riscritto "$\ldots\geq b+1$", eliminando il problema. Che il problema sia di minimo e non di massimo, e che il vincolo sia di $\geq$ invece che di $\leq$ è irrilevante: si può facilmente riportare $(CS_{\bar{x}})$ nell'esatta forma di (KP) con semplici manipolazioni algebriche.

Esercizio 4.6. Si mostri la correttezza dell'ultima affermazione (suggerimento: potrebbe essere utile il cambio di variabile $y_i=1-z_i$).

Pertanto, il problema di separazione per le cover inequalities di (KP)—che non esauriscono neanche tutte le facette del corrispondente $\bar{\mathcal{F}}$—appare essere sostanzialmente equivalente al problema di partenza, ossia $\mathcal{NP}$-hard, confermando il Teorema 4.1.

Per riassumere, l'esistenza di una formulazione di *PL* equivalente a ciascun problema di *PLI* non rende la *PLI* facile: la formulazione esiste, ma non abbiamo nessun modo efficiente per generarla, e neanche per generarne una parte sufficiente a caratterizzare la soluzione ottima (si noti che basterebbe una base ottima). Questo però suggerisce alcuni interessanti approcci per la *PLI*, detti *metodi poliedrali*, che possono risultare molto utili per la soluzione di ampie classi di problemi. Questi metodi verranno descritti più in dettaglio nel seguito, ma la loro struttura di base può essere descritta dalle seguenti due idee:

1. Anche se il problema di separazione per $\tilde{\mathcal{F}}$ è $\mathcal{NP}$-arduo, può accadere che alcuni dei vincoli importanti che lo caratterizzano (faccette) possano essere separati efficientemente. Si può cioè disporre di separatori efficienti non per $\tilde{\mathcal{F}}$, ma per un suo soprainsieme che sia comunque "molto più piccolo" della regione ammissibile $\mathcal{F}$ delle formulazioni *PLI* "naturali".
2. Come si è visto, i problemi di separazione vengono tipicamente formulati attraverso il problema di ottimizzazione di determinare "il vincolo più violato" tra quelli di interesse. Anche se questo problema è $\mathcal{NP}$-arduo da risolvere all'ottimo, in effetti ai fini dell'oracolo è possibile risolverlo solamente "con l'accuratezza sufficiente" per determinare un qualsiasi vincolo violato (computazionalmente non è neanche necessariamente vero che determinare quello più violato porti ad un comportamento più efficiente in pratica). Si possono pertanto usare *euristiche* (cf. Capitolo 5) per risolvere il problema di separazione: se si determina un vincolo violato $(\,d\,,\,\delta\,)$ questo viene utilizzato, altrimenti ci si ferma non avendo aggiunto tutti i tagli che sarebbe stato in teoria possibile aggiungere, ma sperabilmente avendo significativamente ridotto la regione ammissibile del rilassamento rispetto a quella $\mathcal{F}$ delle formulazioni *PLI* "naturali".

In altri termini, è possibile risolvere un rilassamento di (PLI) "più debole" di $(\widetilde{RC})$, ma che comunque comprende molta più "informazione" sul problema originario rispetto ad una formulazione "naturale". Questo può rendere possibile risolvere (PLI) in modo più efficiente in quanto il rilassamento migliore fornisce quindi valutazioni superiori di $z(PLI)$ più accurate di quelle fornite da formulazioni più semplici, anche se potenzialmente ad un costo computazionale superiore. Il prossimo paragrafo fornisce ulteriori indicazioni generali rispetto al motivo per cui queste valutazioni superiori, per quanto non necessariamente esatte, possono essere rilevanti ai fini della soluzione algoritmica del problema di *PLI*.

4.3 Dimostrazioni di ottimalità

In generale, il processo di soluzione di un qualsiasi problema di ottimizzazione, ad esempio della forma

$$(P)\quad \max\{\,c(\,x\,)\ :\ x\in X\,\}\ ,$$

può essere considerato come composto di due parti distinte:

- individuazione di una soluzione ottima x^*;
- individuazione di una valutazione superiore di $z(P)$ che *dimostri* l'ottimalità di x^*, ossia un valore $\bar{z}$ per il quale sia garantito che $z(P) \leq \bar{z}$, ma per il quale risulti anche $c(x^*) = \bar{z}$, in modo tale che

$$\bar{z} = c(x^*) \leq z(P) \leq \bar{z} \ .$$

In molti algoritmi visti nei capitoli precedenti le valutazioni superiori (o inferiori) erano esplicitamente descritte. In altri le dimostrazioni di ottimalità non facevano uso esplicito di valutazioni superiori (o inferiori, per problemi di minimo), ma tali valutazioni potrebbero essere costruite e mostrate. Per valutazioni superiori disponibili in modo esplicito, si considerino i problemi dell'albero dei cammini minimi, del Flusso Massimo e di *PL*. Nel primo caso, è facile dimostrare, usando il Teorema 2.1 e la definizione della funzione obiettivo, che una valutazione inferiore del costo dell'albero ottimo è data dalla somma delle etichette associate ai nodi per qualsiasi vettore di etichette che rispetti le condizioni di Bellman: quando l'algoritmo termina, la valutazione inferiore è pari al costo dell'albero individuato. Nel secondo caso, una valutazione superiore del valore del massimo flusso è fornita dalla capacità di un qualsiasi taglio che separi la sorgente dal pozzo (Teorema 2.5): al termine, l'algoritmo ha costruito un taglio di capacità pari al valore del flusso determinato, che risulta quindi massimo. Nel caso della *PL*, una valutazione superiore del valore della funzione obiettivo di qualsiasi soluzione primale ammissibile è data dal valore della funzione obiettivo di qualsiasi soluzione dual ammissibile (Teorema 3.9): al termine, gli algoritmi del Simplesso hanno individuato una coppia di soluzioni ammissibili per il primale ed il duale con lo stesso valore di funzione obiettivo, e quindi hanno una dimostrazione esplicita di ottimalità per la soluzione primale (e per quella duale). In tutti questi casi la valutazione superiore deriva sostanzialmente dal problema duale, anche se nei primi due la connessione non è stata fatta in modo esplicito. Nel caso di (MCF), valutazioni sul valore ottimo della funzione obiettivo erano in effetti disponibili, anche se non ne abbiamo fatto esplicitamente uso. Per (MST) le valutazioni (inferiori) non sono discusse nel §2.3 ma lo sono nel §5.1.3. Ad un'attenta ispezione tutte le valutazioni superiori (o inferiori) descritte risultano derivare dalla teoria della dualità della *PL*, che è infatti uno dei metodi più potenti e generali per derivare condizioni di ottimalità. Nel Capitolo 6 vedremo però una tecnica che la generalizza.

Per i problemi $\mathcal{NP}$-ardui di *OC* non si hanno, in generale, tecniche per determinare efficacemente valutazioni esatte del valore del problema. Un diverso modo di vedere la cosa è il seguente: per quasi tutti i problemi di *OC*, *la difficoltà del problema consiste fondamentalmente nella determinazione del suo valore ottimo*. Infatti, moltissimi problemi di *OC* godono di una proprietà chiamata *auto-riducibilità*, dalla quale segue che se esistesse un algoritmo efficiente per determinarne il valore ottimo allora esisterebbe un algoritmo efficiente per determinare una soluzione ottima del problema.

Esempio 4.8. Auto-riducibilità della Programmazione Lineare 0–1
Come abbiamo visto, la *Programmazione Lineare 0–1*

$$(PL01) \quad \max\{\, cx \,:\, Ax \leq b \ , \ x \in \{\,0\,,\,1\,\}^n \,\}$$

è un'ampia classe di modelli attraverso la quale si possono esprimere tutti i problemi di *OC* $\mathcal{NP}$-ardui (cf. §1.9). Vogliamo adesso mostrare che la capacità di calcolare esattamente il valore ottimo di uno di tali problemi è polinomialmente equivalente alla capacità di calcolarne la soluzione ottima.

Una delle due direzioni è ovvia: data la soluzione ottima x^* è banale calcolarne il valore della funzione obiettivo. Per l'altra direzione, si supponga di avere un algoritmo $\mathcal{A}$ in grado di calcolare, data una qualunque istanza p di $(PL01)$, il valore ottimo $z(p)$, *senza però fornire una corrispondente soluzione ottima*. Vediamo come sia possibile utilizzare $\mathcal{A}$ per costruire una soluzione ottima del problema.

Per prima cosa, si utilizza $\mathcal{A}$ per calcolare $z^* = z(p)$. A questo punto, si seleziona una qualunque tra le variabili del problema, ad esempio la prima (x_1), e si verifica, utilizzando l'algoritmo $\mathcal{A}$, se nelle soluzioni ottime del problema si ha $x_1^* = 0$ oppure $x_1^* = 1$. Per questo è sufficiente costruire una nuova istanza di $(PL01)$, p_1^0, ottenuta da p fissando a 0 il valore di x_1: ciò si ottiene facilmente eliminando la prima colonna di A, A^1, ed il corrispondente elemento c_1 del vettore dei coefficienti. Utilizziamo adesso $\mathcal{A}$ per calcolare $z_1^0 = z(p_1^0)$. Se $z_1^0 = z^*$, allora esiste

effettivamente una soluzione ottima x^* di p in cui $x^*_1 = 0$: è quindi possibile fissare definitivamente in questo modo il valore della variabile. In altri termini, da questo momento si rimpiazza p con p^0_1. Se invece $z_1 < z^*$, allora $x^*_1 = 1$ in qualsiasi soluzione ottima di p. È facile costruire una nuova istanza p^1_1 corrispondente al fissare $x^*_1 = 1$: per questo, è sufficiente eliminare la prima colonna di A ed il corrispondente elemento del vettore dei coefficienti, come nel caso precedente, ma modificando anche il vettore dei lati destri b in $b - A^1$. Anche in questo caso si rimpiazza p, ma stavolta con p^1_1: pertanto, in entrambi i casi abbiamo ricondotto l'istanza ad una più piccola, con una variabile (x_1) in meno. Si può quindi iterare il procedimento considerando una variabile della nuova istanza (ad esempio la prima, che corrisponde a x_2 dell'istanza originale). Con al più $n + 1$ chiamate all'algoritmo $\mathcal{A}$ si determina una soluzione ottima x^* dell'istanza originaria. Di conseguenza, se $\mathcal{A}$ fosse polinomiale sarebbe possibile costruire una soluzione ottima di $(PL01)$ in tempo polinomiale, il che implicherebbe $\mathcal{P} = \mathcal{NP}$.

Questa osservazione giustifica, anche dal punto di vista teorico, l'interesse per tecniche in grado di determinare valutazioni superiori (inferiori nel caso di un problema di minimo) del valore ottimo di un dato problema di *OC*. Usando la terminologia propria della teoria della complessità computazionale, possiamo considerare il valore ottimo della funzione obiettivo di un problema di *OC* come un *certificato di ottimalità*. I problemi di *OC* "facili" sono quelli per i quali sono disponibili tecniche efficienti—tipicamente basate sulla dualità—per costruire un tale certificato, mentre quelli "difficili" sono quelli per cui non sono note tecniche in grado di svolgere efficientemente questo compito. Per questo vengono sviluppate tecniche—alcune già accennate nei paragrafi precedenti, altre discusse nel Capitolo 6—che determinano valutazioni superiori (inferiori nel caso di un problema di minimo) *approssimate* del problema, che poi vengono usate (in un modo che in effetti ricorda molto l'Esempio 4.8, si veda il Capitolo 7) per guidare la ricerca di una soluzione ottima del problema.

Le tecniche per determinare valutazioni superiori del valore ottimo della funzione obiettivo di un dato problema di *OC* sono infatti molto importanti nelle applicazioni. Per molte classi di problemi di *OC* esiste *in pratica* una consistente differenza tra *produrre* una soluzione ε-ottima e *certificare* la ε-ottimalità di una soluzione data. Ad esempio, in molti algoritmi enumerativi per problemi "difficili" (si veda il Capitolo 7) capita sovente che l'algoritmo determini la soluzione ottima in tempo relativamente breve, ma sia poi ancora necessario un grandissimo sforzo computazionale per *dimostrare* che tale soluzione è effettivamente ottima. In altri termini, la difficoltà del problema risiede non tanto nel costruire una soluzione ottima, quando nel verificarne l'ottimalità, ossia nel determinare il valore ottimo del problema. Alle tecniche utili a determinare questo valore, o una sua approssimazione accurata, è dedicata una parte rilevante della ricerca attuale volta a sviluppare algoritmi "efficienti" per problemi di *OC*.

Riferimenti Bibliografici

M. Grötschel, L. Lovász, A. Schrijver "Geometric Algorithms and Combinatorial Optimization" Springer, 1988

J. Lee "A First Course in Linear Optimization v4.06" 2022
https://github.com/jon77lee/JLee_LinearOptimizationBook/blob/master/JLee.4.06.zip

M. Pappalardo, M. Passacantando "Ricerca Operativa" Edizioni Plus, 2013

L. Wolsey "Integer Programming" Wiley, 2020

Capitolo 5

Algoritmi euristici

Dato che molti problemi di *OC* sono "difficili" è spesso necessario sviluppare algoritmi *euristici*, ossia algoritmi che non garantiscono di ottenere una soluzione ottima, ma sono spesso in grado di fornire un "buone" soluzioni ammissibili. È però possibile, in generale, che un algoritmo euristico "fallisca" e non sia in grado di determinare nessuna soluzione ammissibile del problema, pur senza essere in grado di dimostrare che non ne esistano; questo perché per diversi problemi di *OC* è "difficile" anche il problema decisionale di determinare se esiste una qualsiasi soluzione ammissibile.

Normalmente gli algoritmi euristici hanno una struttura semplice ed una ridotta complessità in tempo, ma in alcuni casi, per problemi di grandi dimensioni e struttura complessa, può essere necessario sviluppare algoritmi euristici sofisticati e di elevata complessità. Anche per lo stesso problema, come vedremo, si possono sviluppare euristiche con livelli di sofisticazione, e di conseguenza complessità, molto diverse: non è quindi ovvio navigare gli inevitabili compromessi tra la qualità della soluzione fornita (e persino la probailità che venga identificata una soluzione) ed il tempo dedicato a cercarla. Questo è particolarmente vero perché le euristiche possono avere due modalità di uso molto diverse:

- essere utilizzate una volta solamente per fornire direttamente la soluzione del problema che viene poi passata al decisore;
- essere utilizzate ripetutamente, anche moltissime volte, all'interno di approcci più complessi per la soluzione (eventualmente esatta) del problema, quali quelli descritti nel Capitolo 7.

Ovviamente i due patterns di uso portano a scelte molto diverse in termini del rapporto tra tempo speso e qualità attesa della soluzione. Per contro, ovviamente un'euristica che produca quasi sempre in tempo molto breve soluzioni di ottima qualità (qualora fosse disponibile) ovviamente sarebbe preferibile in ogni caso. Questo capitolo presenterà quindi approcci che siano utilizzabili in entrambe le situazioni.

Il progetto di algoritmi euristici efficienti ed efficaci richiede un'attenta analisi del problema da risolvere, volta ad individuarne la "struttura", ossia le caratteristiche specifiche utili. È però innanzitutto necessaria una buona conoscenza delle principali tecniche algoritmiche disponibili. Infatti, anche se ogni problema ha le sue caratteristiche specifiche, esistono un certo numero di tecniche generali che possono essere applicate, in modi diversi, a moltissimi problemi, producendo *classi* di algoritmi di ottimizzazione ben definite. In questo capitolo ci soffermeremo su due tra le principali tecniche algoritmiche utili per la realizzazione di algoritmi euristici per problemi di *OC*: gli algoritmi *greedy* e quelli di *ricerca locale*. Queste tecniche algoritmiche non esauriscono certamente lo spettro delle euristiche possibili, ma forniscano una buona base di partenza per l'analisi e la caratterizzazione di moltissimi approcci. In particolare, vale la pena sottolineare che l'enfasi sulla "struttura" del problema di ottimizzazione è comune anche alle tecniche utilizzate per la costruzione di valutazioni del valore ottimo del problema, che saranno esaminate nel Capitolo 6. Questo fa sì che una stessa struttura del problema possa essere utilizzata sia per realizzare euristiche che per determinare valutazioni superiori; si può così avere una "collaborazione" tra euristiche e rilassamenti, come nei casi delle *tecniche di arrotondamento* e delle *euristiche Lagrangiane*, che saranno discusse nel Capitolo 6. Altri esempi di situazioni in cui la computazione di un rilassamento è parte integrante di—o comunque guida—un approccio euristico saranno presentate già in questo capitolo.

5.1 Algoritmi greedy

Gli algoritmi *greedy* (voraci) determinano una soluzione attraverso una sequenza di decisioni "localmente ottime", senza mai tornare, modificandole, sulle decisioni prese. Questi algoritmi sono di facile implementazione e notevole efficienza computazionale ma, sia pure con alcune eccezioni di rilievo, in generale non garantiscono l'ottimalità, ed a volte neppure l'ammissibilità, della soluzione trovata.

La definizione che abbiamo dato di algoritmo greedy è molto generale, e quindi possono essere ricondotti a questa categoria algoritmi anche all'apparenza molto diversi tra loro. È comunque possibile, a titolo di esemplificazione, definire uno schema generale di algoritmo greedy, adatto a tutti quei casi in cui l'insieme ammissibile può essere rappresentato come una famiglia $F \subset 2^E$ di sottoinsiemi di un dato insieme "base" E.

```
procedure S = Greedy( E , F )  {
  S := ∅; Q = E;
  do { e = Best( Q ); Q = Q \ { e };
       if( S ∪ { e } ∈ F ) then S = S ∪ { e };
       } while( Q ≠ ∅ and not Maximal( S ) );
  }
```

Procedura 5.1: Algoritmo *Greedy*

Nella procedura, S è l'insieme degli elementi di E che sono stati inseriti nella soluzione (parziale) corrente, e Q è l'insieme degli elementi di E ancora da esaminare: gli elementi in $E \setminus (S \cup Q)$ sono quelli per cui si è deciso che *non* faranno parte della soluzione finale. La funzione $Maximal()$ restituisce *vero* se non è più possibile aggiungere elementi alla soluzione S, cosa che in molti casi si può determinare in modo computazionalmente efficiente (ad esempio perché è noto a priori il massimo numero di elementi che qualsiasi soluzione ammissibile può avere); in questo caso l'algoritmo può terminare senza esaminare gli elementi eventualmente ancora presenti in Q. In molti casi, le soluzioni ammissibili del problema sono solamente gli elementi "massimali" di F: quindi, se l'algoritmo greedy termina avendo esaurito gli elementi di Q senza che $Maximal()$ restituisca *vero*, allora l'algoritmo "fallisce", ossia non è in grado di determinare una soluzione ammissibile del problema. La funzione $Best()$ fornisce il miglior elemento di E tra quelli ancora in Q sulla base di un prefissato criterio, ad esempio l'elemento di costo minimo nel caso di problemi di minimo; la scelta di tale criterio è tipicamente cruciale per l'efficacia dell'approccio.

5.1.1 Esempi di algoritmi greedy

È immediato verificare che l'algoritmo di Kruskal (cf. §2.3.1) per (MST) ricade nello schema generale appena proposto: E è l'insieme degli archi del grafo, e S è la soluzione parziale corrente, ossia un sottografo privo di cicli. La funzione $Best()$ determina un arco di costo minimo tra quelli non ancora considerati: ciò viene eseguito scorrendo la lista degli archi in ordine di costo non decrescente. Il controllo "$S \cup (e) \in F$" corrisponde a verificare che il lato $e = \{ i , j \}$ selezionato non formi un ciclo nel sottografo individuato da S. La funzione $Maximal()$ restituisce *vero* quando S contiene esattamente $n - 1$ archi, ossia è un albero di copertura; se il grafo non è connesso, $Maximal()$ non restituisce mai *vero* e l'algoritmo "fallisce", ossia non è in grado di determinare un albero di copertura (semplicemente perchè non ne esistono). Ricordiamo che l'algoritmo di Kruskal ha complessità $O(m \log n)$, essenzialmente dovuta all'ordinamento degli archi per costo non decrescente. Come abbiamo visto (e rivedremo, cf. §5.1.3), per il caso di (MST) si può dimostrare che la soluzione generata dall'algoritmo greedy è ottima (l'algoritmo è *esatto*). Vediamo adesso altri esempi di algoritmi greedy, per diversi problemi di *OC*, che risultano invece avere un comportamento euristico.

5.1.1.1 Il problema dello zaino

Si consideri il problema dello zaino (KP) dell'Esempio 1.5. Un semplice algoritmo greedy per questo problema consiste nel costruire una soluzione inserendo per primi nello zaino gli oggetti "più promettenti", ossia quelli che con maggiore probabilità appartengono ad una soluzione ottima, secondo un qualche criterio euristico. L'algoritmo inizializza l'insieme $S \subset I = \{ 1 , 2 , \ldots , n \}$ degli oggetti selezionati come l'insieme vuoto, e poi scorre la lista degli oggetti ordinati secondo il criterio euristico:

l'oggetto a_h di volta in volta selezionato viene accettato se la capacità residua dello zaino è sufficiente, cioè se $b - \sum_{i \in S} a_i \geq a_h$; in questo caso l'oggetto a_h viene aggiunto ad S, altrimenti viene scartato e si passa al successivo nell'ordinamento. L'algoritmo termina quando tutti gli oggetti sono stati esaminati oppure la capacità residua dello zaino diviene 0. È immediato verificare che anche questo algoritmo ricade nello schema generale. E è l'insieme degli oggetti tra i quali scegliere, la funzione $Best()$ determina l'oggetto migliore secondo il criterio euristico, il controllo "$S \cup \{ e \} \in F$" corrisponde a verificare che la capacità residua dello zaino sia sufficiente ad accogliere il nuovo oggetto, e la funzione $Maximal()$ restituisce *vero* quando la capacità residua dello zaino è zero. In questo caso le soluzioni sono ammissibili anche se l'algoritmo termina senza che $Maximal()$ restituisca *vero*; ciò non sarebbe più vero qualora il problema richiedesse di determinare un insieme di elementi di peso *esattamente uguale* a b.

Si noti che quello appena proposto non è *un* algoritmo, ma piuttosto una *famiglia* di algoritmi greedy per (KP) che si differenziano per il criterio con cui vengono selezionati gli oggetti. Consideriamo ad esempio i seguenti tre criteri:

- *pesi non decrescenti*: $a_1 \leq a_2 \leq \ldots \leq a_n$;
- *costi non crescenti*: $c_1 \geq c_2 \geq \ldots \geq c_n$;
- *costi unitari non crescenti*: $c_1/a_1 \geq c_2/a_2 \geq \ldots \geq c_n/a_n$.

Ciascuno dei tre criteri è "ragionevole": con il primo si cercano di inserire nello zaino "molti" oggetti, col secondo quelli di costo maggiore, con il terzo quelli che hanno il maggiore costo per unità di spazio occupato. Nessuno dei tre criteri di ordinamento degli elementi domina gli altri; tuttavia, è facile rendersi conto del fatto che l'ultimo (costi unitari non crescenti) è il più ragionevole e quello dei tre che fornisce risultati migliori nella maggioranza dei casi. Chiamiamo CUD l'algoritmo greedy per (KP) che utilizzi il criterio dei costi unitari non crescenti.

Esercizio 5.1. Per ciascuno dei tre criteri, costruire un esempio in cui la soluzione fornita secondo quel criterio domini le soluzioni fornite in accordo agli altri.

Esempio 5.1. Esecuzione di Greedy-CUD per (KP)
Consideriamo la seguente istanza del problema dello zaino:

$$\begin{array}{rrcrcrcrcrcrcc} \max & 7x_1 & + & 2x_2 & + & 4x_3 & + & 5x_4 & + & 4x_5 & + & x_6 & & \\ & 5x_1 & + & 3x_2 & + & 2x_3 & + & 3x_4 & + & x_5 & + & x_6 & \leq & 8 \\ & x_1 & , & x_2 & , & x_3 & , & x_4 & , & x_5 & , & x_6 & \in & \{ 0 , 1 \} \end{array}$$

In questo caso, l'algoritmo CUD esegue i seguenti passi:

1. la variabile con costo unitario maggiore è x_5, per cui risulta $c_5/a_5 = 4$: si pone allora $x_5 = 1$, e lo zaino rimane con una capacità residua di 7 unità;
2. la seconda variabile, nell'ordine scelto, è x_3, il cui costo unitario è 2: essa ha un peso minore della capacità residua, e si pone quindi $x_3 = 1$; la capacità residua dello zaino scende di conseguenza a 5;
3. la terza variabile esaminata è x_4, il cui costo unitario è 5/3: anche essa ha un peso minore della capacità residua, e si pone quindi $x_4 = 1$ cosicché lo zaino rimane con una capacità residua di 2 unità;
4. la quarta variabile considerata è x_1, il cui costo unitario è 7/5: essa ha però peso 5, superiore alla capacità residua 2 dello zaino, e pertanto si pone $x_1 = 0$;
5. la quinta variabile, nell'ordine, è x_6, che ha costo unitario 1: la variabile ha peso 1, inferiore alla capacità residua, pertanto si pone $x_6 = 1$ e lo zaino rimane con una capacità residua di 1 unità;
6. l'ultima variabile considerata è x_2: tale variabile ha un peso (5) superiore alla capacità residua (1), e quindi si pone $x_2 = 0$.

La soluzione ottenuta è allora $[\,0\,,\,0\,,\,1\,,\,1\,,\,1\,,\,1\,]$, con costo 14 e peso totale 7: è facile vedere che questa soluzione non è ottima, dato che la soluzione $[\,1\,,\,0\,,\,1\,,\,0\,,\,1\,,\,0\,]$, con peso totale 8, ha un costo di 15.

Esercizio 5.2. Si esegua l'algoritmo greedy per (KP) sull'istanza dell'esempio precedente utilizzando gli altri due criteri euristici per la selezione dell'elemento.

L'algoritmo greedy ha complessità $O(\, n \log n \,)$, con ciascuno dei tre criteri sopra descritti, essenzialmente dovuta all'ordinamento degli oggetti; se gli oggetti sono forniti in input già ordinati secondo il criterio selezionato, la complessità dell'algoritmo è lineare.

5.1.1.2 Il problema dell'assegnamento di costo minimo

Si consideri il problema accoppiamento di massima cardinalità e costo minimo discusso al §2.6.2. Un algoritmo greedy per questo problema può essere facilmente costruito nel modo seguente. Si ordinano gli archi per costo non crescente, e si inizializza un vettore di booleani, con una posizione per ogni nodo, a $falso$, per indicare che nessun nodo è assegnato. Si esaminano poi gli archi nell'ordine dato; se entrambe gli estremi dell'arco non sono assegnati l'arco viene aggiunto all'accoppiamento e si pongono a *vero* le posizioni nell'array corrispondenti ai nodi adesso accoppiati. L'algoritmo termina quando non ci sono più nodi da accoppiare oppure quando sono stati esaminati tutti gli archi. Questo algoritmo ricade ovviamente nello schema generale: E è l'insieme A degli archi del grafo, la funzione $Best()$ determina l'arco di costo minimo tra quelli non ancora esaminati (il prossimo nell'ordinamento), il controllo "$S \cup \{\, e \,\} \in F$" corrisponde a verificare che entrambe gli estremi dell'arco non siano già assegnati, la funzione $Maximal()$ ritorna *vero* se tutti i nodi sono accoppiati. Se si desidera un assegnamento (accoppiamento perfetto) l'algoritmo "fallisce" se termina avendo esaminato tutti gli archi senza che $Maximal()$ abbia ritornato *vero*. È facile dimostrare che questo algoritmo non costruisce necessariamente una soluzione ottima del problema; in particolare può non essere neanche in grado di determinare un assegnamento nel grafo anche se ne esiste uno.

Esercizio 5.3. Si fornisca un esempio che dimostri l'affermazione precedente (suggerimento: si veda il §5.1.3).

In compenso l'algoritmo ha complessità $O(\, m \log n \,)$, essenzialmente dovuta all'ordinamento degli archi, sostanzialmente inferiore a quella $O(\, mn^2 \,)$ degli algoritmi esatti discussi nel §2.6.2; quindi questo algoritmo potrebbe ad esempio risultare utile per ottenere rapidamente "buoni" assegnamenti per problemi di grandissima dimensione. Si noti inoltre che questo algoritmo, a differenza di quelli del §2.6.2, è adatto anche al caso in cui il grafo G non sia bipartito. Sono comunque stati proposti algoritmi esatti per il caso bipartito con complessità inferiore a $O(\, mn^2 \,)$ (tra cui alcuni basati sull'idea di "trasformare" la soluzione ottenuta dall'algoritmo greedy), ed anche algoritmi esatti per il caso non bipartito; per ulteriori dettagli si rimanda alla letteratura citata.

5.1.1.3 Il problema del commesso viaggiatore

Si consideri il problema del commesso viaggiatore (TSP) definito nell'Esempio 1.7. Una famiglia di algoritmi greedy per questo problema può essere costruita come segue. L'algoritmo inizializza l'insieme S dei lati appartenenti al ciclo come l'insieme vuoto, e definisce un qualsiasi vertice (ad esempio, 1) come quello "corrente". Ad ogni iterazione esamina il vertice corrente i ed ordina tutti i lati ad esso incidenti secondo un certo criterio euristico. Esamina quindi tali lati nell'ordine dato, e non appena trova un lato $\{\, i \,,\, j \,\}$ tale che j non è ancora toccato dal ciclo parziale S ve lo aggiunge e definisce j come nuovo vertice corrente. L'algoritmo termina quando tutti i vertici sono toccati da S, inserendo il lato "di ritorno" dall'ultimo vertice toccato al primo. Anche questo algoritmo ricade nello schema generale: E è l'insieme dei lati del grafo, $Best()$ determina il prossimo lato $\{\, i \,,\, j \,\}$ uscente dal vertice corrente i secondo il criterio euristico, il controllo "$S \cup \{\, e \,\} \in F$" corrisponde a verificare che j non sia già stato visitato, e $Maximal()$ restituisce *vero* quando tutti i nodi sono stati visitati.

Anche in questo caso abbiamo una famiglia di algoritmi che si differenziano per il criterio utilizzato per determinare l'arco "più promettente". Un criterio molto intuitivo è quello di ordinare i lati uscenti da i per lunghezza non decrescente: ciò corrisponde a scegliere ad ogni passo, come prossima tappa, la località non ancora visitata più vicina a quella in cui ci si trova attualmente, noto come algoritmo "Nearest Neighbour".

Esempio 5.2. Esempio di esecuzione di "Nearest Neighbour"

Come nel caso del problema dello zaino, l'algoritmo "Nearest Neighbour" non è esatto (del resto, entrambi i problemi sono $\mathcal{NP}$-ardui): questo può essere facilmente verificato mediante l'istanza rappresentata in Figura 5.1(a). Partendo dal nodo 1 l'algoritmo produce il ciclo rappresentato in Figura 5.1(b), con lunghezza 12, che è peggiore del ciclo

ottimo rappresentato in Figura 5.1(c), che ha costo 11.

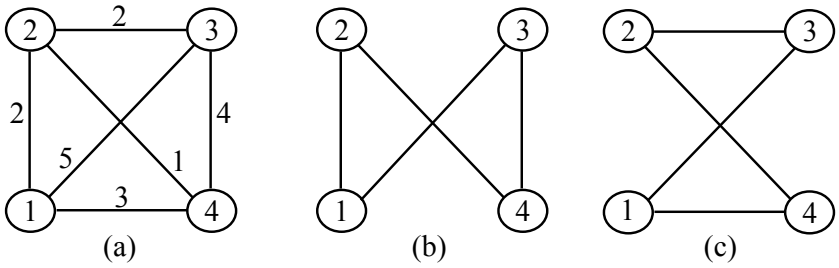


Figura 5.1: Esecuzione di "Nearest Neighbour" su un'istanza di (TSP)

Si noti che l'algoritmo costruisce sicuramente un ciclo Hamiltoniano se il grafo G è completo, mentre può "fallire" nel caso in cui G non lo sia: ciò accade se tutti i lati uscenti dal vertice corrente i portano a vertici già visitati dal ciclo parziale S, oppure se i è l'ultimo dei vertici da visitare ma non esiste il lato fino al nodo iniziale. Quindi, a differenza del problema dello zaino, l'algoritmo greedy non solo non garantisce di determinare una soluzione ottima, ma può non essere in grado di produrre neanche una qualsiasi soluzione ammissibile. Ciò non deve stupire: mentre per il problema dello zaino è immediato costruire una soluzione ammissibile (lo zaino vuoto), il problema di decidere se esiste un ciclo Hamiltoniano in un grafo non completo è $\mathcal{NP}$-arduo, e quindi non è pensabile che un algoritmo greedy sia in grado di risolverlo.

Sono comunque stati proposti molti altri criteri di selezione del nodo successivo che possono rivelarsi più efficienti in pratica. Ad esempio, quando il grafo G può essere rappresentato su un piano (il che ovviamente accade quando i vertici corrispondono effettivamente a località geografiche) un criterio interessante è quello che seleziona j in modo tale che il segmento (lato) $\{ i , j \}$ formi il più piccolo angolo possibile con il segmento (lato) $\{ h , i \}$, dove h è il vertice visitato immediatamente prima di i nel ciclo parziale (ossia $\{ h , i \} \in S$). Pur senza entrare nei dettagli, segnaliamo il fatto che questo criterio è motivato da alcune proprietà della frontiera dell'inviluppo convesso di un insieme di punti del piano e dalle relazioni che esistono tra l'inviluppo convesso dei punti che rappresentano i vertici ed il ciclo Hamiltoniano di costo minimo; quindi, per quanto il criterio sia semplice da capire e da implementare, la sua ideazione è stata resa possibile solamente da uno studio accurato delle proprietà (di alcuni casi rilevanti) del problema in oggetto. Si noti come, comunque, ancora una volta, il costo computazionale della procedura è molto basso, essendo approssimativamente lineare nel numero dei lati del grafo.

5.1.1.4 Ordinamento di lavori su macchine per minimizzare il tempo di completamento

Si consideri il problema di ordinamento di lavori su macchine con minimizzazione del tempo di completamento (MMMS) definito nell'Esempio 1.18. Una famiglia di algoritmi greedy per questo problema può essere costruita come segue. All'inizio, nessun lavoro (trasporto) è assegnato e tutte le macchine (moli) sono scariche, ossia $T(m) = \emptyset$ per $m \in M$. Ad ogni iterazione si seleziona uno dei lavori $i \in T$ ancora da assegnare, secondo un certo criterio euristico, e lo si assegna ad una delle macchine $m \in M$ "più scariche", ossia quelle con tempo di completamento $D(m) = \sum_{i \in T(m)} d_i$—relativo alla soluzione parziale corrente—più basso. L'algoritmo termina quando tutti i lavori sono stati assegnati. Anche questo algoritmo può essere fatto ricadere nello schema generale: E è l'insieme delle coppie $\{ i , m \}$ con $i \in T$ e $m \in M$, $Best()$ seleziona prima un lavoro $i \in T$ non ancora assegnato, secondo il criterio euristico, e poi la macchina $m \in M$ (più scarica) a cui assegnarlo, il controllo "$S \cup \{ e \} \in F$" non esiste in quanto è sempre possibile assegnare un lavoro non assegnato a qualsiasi macchina, ed anche $Maximal()$ non fa nulla in quanto le soluzioni sono ammissibili se e solo se tutti i lavori sono stati assegnati, ossia Q è vuoto.

Anche in questo caso quella appena proposta è una famiglia di algoritmi greedy, detti *list scheduling*, che si differenziano solamente per il criterio utilizzato per determinare il prossimo lavoro i da assegnare. Tra tutti i possibili criteri, due sono quelli più significativi:

- *SPT (Shortest Processing Time)*: i lavori vengono assegnati in ordine non decrescente dei tempi di esecuzione (quelli più "corti" per primi);

- *LPT (Longest Processing Time)*: i lavori vengono assegnati in ordine non crescente dei tempi di esecuzione (quelli più "lunghi" per primi).

Esempio 5.3. Esecuzione di algoritmi list scheduling

Nella Figura 5.2 sono riportati i risultati ottenuti con i due criteri su due esempi: nel primo caso LPT fornisce una soluzione ottima, mentre nel secondo caso nessuno dei due algoritmi riesce a trovare l'ottimo.

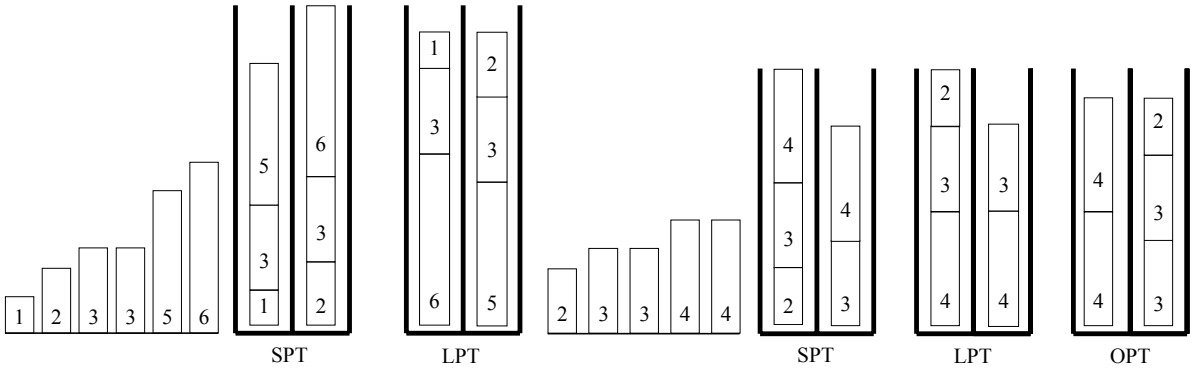


Figura 5.2: Due istanze di (MMMS) risolte con SPT e LPT

Come vedremo in seguito, LPT ha "migliori proprietà" di SPT, e di fatto in generale risulta più efficace, ossia produce soluzioni di migliore qualità. Per entrambe i criteri di ordinamento, comunque, l'algoritmo greedy è facile da implementare e risulta molto efficiente in pratica.

Esercizio 5.4. Si discuta la complessità dell'algoritmo greedy per (MMMS); come cambia il risultato se i lavori sono forniti in input già ordinati secondo il criterio selezionato?

5.1.1.5 Ordinamento di lavori su macchine per minimizzare il numero delle macchine

Si consideri adesso invece il problema (MCMS), simile ma diverso, definito all'esempio 1.12. Una famiglia di algoritmi greedy per questo problema può essere costruita in modo analogo a quello visto nel caso precedente. All'inizio, nessun lavoro è assegnato e nessuna macchina è utilizzata. Ad ogni iterazione si seleziona uno dei lavori i ancora da assegnare, secondo un certo criterio euristico, e si scorre la lista delle macchine già utilizzate, secondo un altro opportuno criterio euristico, assegnando il lavoro alla prima macchina sulla quale è possibile eseguirlo: se non è possibile eseguire i su nessuna delle macchine già utilizzate lo si assegna ad una nuova macchina fino a quel momento scarica, che viene quindi aggiunta all'insieme di macchine utilizzate. Questo algoritmo può essere fatto ricadere nello schema generale analogamente a quanto visto nel paragrafo precedente; si noti che, ancora una volta, quella appena proposta è una famiglia di algoritmi greedy che si differenziano per i criteri utilizzati per determinare il prossimo lavoro i da assegnare e la macchina m già utilizzata a cui assegnarlo (se possibile).

Esercizio 5.5. Si propongano almeno due criteri diversi per la selezione del prossimo lavoro i da assegnare e almeno due criteri diversi per la selezione della macchina m già utilizzata a cui assegnarlo, discutendo i possibili vantaggi e svantaggi di ciascuna combinazione.

Nonostante i due problemi appaiano molto simili, in realtà la loro complessità è molto diversa. Infatti, tra tutti gli algoritmi greedy appartenenti allo schema appena introdotto ne esiste uno che costruisce certamente una soluzione ottima per il problema. Si consideri infatti l'algoritmo in cui il lavoro da assegnare viene selezionato in ordine di tempo di inizio t_i non crescente; in altre parole vengono assegnati per primi i lavori che iniziano prima. Inoltre le macchine vengono ordinate in un qualsiasi ordine (del resto sono tutte identiche) ed esaminate per l'inserzione del lavoro corrente sempre nell'ordinamento dato; se il lavoro non può essere inserito nelle macchine attualmente utilizzate sarà attivata la successiva macchina nell'ordinamento. È possibile dimostrare facilmente che questo algoritmo costruisce un assegnamento che utilizza sempre il minor numero possibile di macchine. Sia infatti k l'indice dell'ultima macchina attivata nel corso dell'algoritmo, ossia il numero di macchine utilizzate nella

soluzione costruita, e sia i il primo lavoro assegnato a quella macchina. Si consideri adesso lo stato delle altre macchine "attive" al momento in cui è stato esaminato i: a ciascuna di esse era sicuramente stato assegnato almeno un lavoro h incompatibile con i, ossia tale che $[\, t_i \,,\, t_i + d_i \,] \cap [\, t_h \,,\, t_h + d_h \,] \neq \emptyset$. Ma, per via della strategia di selezione dei lavori, ciascun lavoro h assegnato ad una macchina in quel momento ha $t_h \leq t_i$: non è quindi possibile che il lavoro i sia incompatibile con esso h perchè $t_i < t_h \leq t_i + d_i$, e pertanto deve necessariamente risultare $t_i \in [\, t_h \,,\, t_h + d_h \,]$. Di conseguenza, nell'istante t_i devono necessariamente essere in esecuzione k lavori: i, che inizia in quel momento, e gli altri $k-1$ che occupano le altre macchine nell'assegnamento (parziale) costruito fino a quel momento. Pertanto sono necessarie almeno k macchine per eseguire tutti i lavori: poichè la soluzione costruita ne usa esattamente k, essa è ottima.

Esercizio 5.6. Si discuta come implementare l'algoritmo greedy "ottimo" per (MCMS) in modo da ottenere una bassa complessità computazionale.

Esercizio 5.7. Si proponga, fornendone una descrizione formale, un algoritmo greedy per il problema (GC) di colorare i nodi di un grafo $G = (\, N \,,\, A \,)$ con il minimo numero di colori con il vincolo che due nodi adiacenti non abbiano mai lo stesso colore descritto nell'Esempio 1.11. Si discuta sotto quali ipotesi sul grafo si può costruire un algoritmo greedy equivalente a quello "ottimo" per (MCMS) che riporti sicuramente una soluzione ottima per (GC). Si discuta poi come modificare l'algoritmo per il caso più generale in cui ad ogni nodo i devono essere assegnati esattamente n_i colori diversi e/o i colori assegnati a nodi adiacenti devono essere "distanti" di almeno una soglia δ fissata.

Questo esempio mostra come la conoscenza di un algoritmo greedy per un certo problema di OC possa suggerire algoritmi greedy analoghi per problemi di OC "simili", ma anche come problemi apparentemente "simili" possano in effetti risultare molto diversi in termini di facilità di soluzione. Infatti, (MCMS) appare simile (almeno dal punto di vista modellistico) tanto a (MMMS) che a (GC), ma entrambi questi ultimi sono $\mathcal{NP}$-ardui mentre il primo è polinomiale.

5.1.1.6 Il problema di copertura

Si consideri il problema di copertura (SC) definito al §1.2.5. Una famiglia di algoritmi greedy per questo problema può essere costruita come segue. L'algoritmo inizializza l'insieme S dei sottoinsiemi selezionati come l'insieme vuoto. Ad ogni iterazione, seleziona uno dei sottoinsiemi F_j ancora da esaminare, secondo un certo criterio euristico, e lo aggiunge a S se il nuovo sottoinsieme "copre" almeno un elemento di E che non era "coperto" dalla soluzione parziale precedente, ossia se $F_j \not\subset F_S = \cup_{F_i \in S} F_i$. L'algoritmo termina quando Q è vuoto oppure quando $F_S = E$, ossia tutti gli elementi di E sono "coperti" da S.

Esercizio 5.8. Si mostri che l'algoritmo appena descritto ricade nello schema generale di algoritmo greedy e si discuta se esso possa "fallire".

Anche in questo caso, quella appena proposta è una famiglia di algoritmi greedy che si differenziano per il criterio utilizzato per determinare il sottoinsieme "più promettente". Consideriamo ad esempio i seguenti tre criteri:

- *costi non decrescenti*: vengono esaminati prima i sottoinsiemi F_j con costo c_j più basso;
- *costi unitari non decrescenti*: vengono esaminati prima i sottoinsiemi F_j con "costo unitario" $c_j \,/\, |\, F_j \,|$ più basso, ossia si tiene conto del numero di oggetti che il sottoinsieme può coprire;
- *costi unitari attualizzati non decrescenti*: vengono esaminati prima i sottoinsiemi F_j con "costo unitario attualizzato" $c_j \,/\, |\, F_j \setminus F_S \,|$ più basso, ossia si tiene conto del numero di oggetti che il sottoinsieme copre e che non sono già coperti dalla soluzione parziale S.

Non è sorprendente che il terzo criterio sia in pratica spesso migliore degli altri due, in quanto è l'unico dei tre che utilizza informazione sulla soluzione corrente S per decidere il prossimo sottoinsieme da esaminare. Si noti anche, però, che tale criterio è più costoso da implementare: infatti per i primi due criteri è possibile ordinare gli oggetti all'inizio e poi semplicemente scorrere la lista ordinata, il che ha complessità $O(\, m \log m \,)$, mentre nel terzo l'ordinamento deve essere ricalcolato ogniqualvolta un oggetto viene inserito in S, e quindi F_S aumenta.

Esercizio 5.9. Si sviluppino e discutano algoritmi greedy per le varianti di partizione (SPP) e di riempimento (SPA) presentate nel §1.2.5.

5.1.1.7 Il problema (CMST)

Si consideri il problema dell'albero di copertura di costo minimo capacitato (CMST) definito nell'Esempio 4.1. Per il problema "più semplice" dell'albero di copertura di costo minimo (MST) conosciamo algoritmi greedy esatti, ossia in grado determinare una soluzione ottima. Chiaramente, tali algoritmi ottengono la soluzione ottima per (CMST) se la capacità Q dei lati incidenti nella radice, ossia il massimo peso dei sottoalberi, è "grande"; per questo, è ragionevole cercare di costruire algoritmi greedy per (CMST) che si ispirino agli algoritmi per (MST). Nell'algoritmo di Kruskal (cf. §2.3.1), ad esempio, si pone $S = \emptyset$ e si esaminano e lati in ordine di costo non decrescente: il lato $\{ i , j \}$ esaminato viene aggiunto ad S se non crea cicli con i lati già presenti in S, ossia se collega due diverse componenti connesse del grafo. Non è difficile modificare l'algoritmo in modo tale che tenga conto delle capacità: basta mantenere il peso (somma del peso dei vertici) di ogni componente connessa, e non accettare il lato $\{ i , j \}$ se la sua inserzione in S causa l'unione di due componenti connesse la cui somma dei pesi è maggiore di Q. Questo ovviamente non si applica ai lati di tipo $\{ r , i \}$, ossia che collegano una componente connessa alla radice. Se esistono lati da r a tutti gli altri vertici allora l'algoritmo così modificato costruisce sicuramente una soluzione ammissibile per il problema, altrimenti può "fallire". È possibile implementare il controllo sul peso delle componenti connesse, mediante opportune strutture dati, in modo da non aumentare la complessità dell'algoritmo di Kruskal.

Questo esempio mostra come la conoscenza di algoritmi per un dato problema possa essere utilizzata per guidare la realizzazione di approcci per problemi simili ma più complessi. Naturalmente, non sempre è facile adattare gli algoritmi noti per risolvere problemi più complessi in modo naturale: ad esempio, adattare l'algoritmo di Prim (cf. §2.3.2) al (CMST) è molto meno immediato.

5.1.2 Algoritmi greedy con garanzia sulle prestazioni

Nel paragrafo precedente sono stati presentati (famiglie di) algoritmi greedy per alcuni problemi di ottimizzazione rilevanti. Una volta che un algoritmo sia stato ideato ed implementato, si pone il problema di valutarne l'efficacia, ossia la capacità di fornire effettivamente soluzioni "buone", ossia con errore relativo basso. Ci sono essenzialmente due modi per studiare l'efficacia di un algoritmo euristico:

- *sperimentale*: si seleziona un sottoinsieme "rilevante" di istanze del problema, ad esempio tali che siano rappresentative delle istanze da risolvere in una o più applicazioni pratiche, si esegue l'algoritmo su quelle istanze misurando l'errore relativo ottenuto e poi se ne esaminano le caratteristiche statistiche (media, massimo, minimo, varianza, ...); si noti che per fare questo è necessario essere in grado di calcolare il valore ottimo, o una sua buona approssimazione, per le istanze test;
- *teorico*: si dimostrano matematicamente valutazioni sul massimo errore compiuto dall'algoritmo quando applicato ad istanze con caratteristiche date; più raramente è possibile valutare anche altre caratteristiche statistiche dell'errore compiuto, quali la media.

Le due metodologie di valutazione non sono alternative ma complementari. Lo studio teorico risulta in genere meno accurato nel valutare l'errore compiuto per le istanze reali, ma fornisce valutazioni generali valide per grandi classi di istanze e aiuta a comprendere meglio il comportamento dell'algoritmo, eventualmente suggerendo come modificarlo per renderlo più efficace. D'altro canto lo studio sperimentale permette di valutare con maggiore accuratezza l'errore compiuto sulle istanze utilizzate e di estrapolare con ragionevole accuratezza l'errore che ci si può attendere su istanze simili, ma non fornisce alcuna garanzia, specialmente per istanze con caratteristiche diverse da quelle effettivamente testate. Ciò è del tutto analogo alla differenza tra lo studio della complessità nel caso pessimo di un algoritmo e la verifica sperimentale dell'efficienza pratica della sua implementazione.

Mostreremo adesso alcuni semplici risultati relativi alla valutazione dell'errore di alcuni algoritmi greedy, e illustreremo come sia possibile progettare algoritmi in modo da garantire che abbiano determinate

prestazioni. Si noti che lo studio teorico delle prestazioni degli algoritmi è in principio possibile anche per altre classi di algoritmi euristici, come quelli di ricerca locale che presenteremo nel §5.2, ma risulta spesso molto difficile per algoritmi che non abbiano una struttura semplice come gli algoritmi di tipo greedy.

Dato un algoritmo euristico $\mathcal{A}$, si distinguono due diversi modi per valutare l'errore compiuto da $\mathcal{A}$: *a priori* e *a posteriori*. La valutazione a posteriori dell'errore viene effettuata dopo che l'algoritmo ha determinato la soluzione corrispondente ad una singola istanza I, e permette di valutare l'errore compiuto per quella particolare istanza. La valutazione a priori invece fornisce una stima del massimo errore compiuto da $\mathcal{A}$ per *qualsiasi* istanza I, ed è quindi disponibile prima che l'istanza venga risolta. La valutazione a priori è quindi più generale, ma siccome è una valutazione nel caso pessimo è anche usualmente (molto) meno precisa di quella a posteriori.

5.1.2.1 L'algoritmo CUD per (KP)

Per valutare l'errore compiuto dall'algoritmo CUD, è necessario per prima cosa ottenere una valutazione superiore del valore ottimo della funzione obiettivo del problema. Per fare ciò consideriamo il rilassamento continuo del problema dello zaino, ossia il problema

$$(\underline{KP}) \quad \max \left\{ \sum_{i\in I} c_i x_i \,:\, \sum_{i\in I} a_i x_i \le b \;,\; x \in [\,0\,,\,1\,]^n \right\} .$$

È possibile verificare che una soluzione x^* ottima per $(\underline{KP})$ può essere costruita nel modo seguente: si ordinano gli oggetti per costo unitario non crescente (cf. §5.1), si inizializza l'insieme S degli oggetti selezionati (ossia degli indici delle variabili i con $x_i^* = 1$) all'insieme vuoto e si iniziano ad inserire oggetti in S (porre variabili a 1) finchè è possibile, esattamente come nell'algoritmo CUD. Quando però si raggiunge il primo oggetto h (nell'ordine dato) per cui la capacità residua dello zaino non è più sufficiente, cioè $b - \sum_{i\in S} a_i < a_h$, si pone $x_h^* = (\,b - \sum_{i\in S} a_i\,) / a_h$ e $x_i^* = 0$ per $i \notin S \cup \{\,h\,\}$; si noti che h è ben definito in quanto per ipotesi risulta $b < \sum_i a_i$. Infatti, si consideri il duale di $(\underline{KP})$:

$$(\underline{DKP}) \quad \min \left\{ yb + \sum_{i\in I} w_i \,:\, ya_i + w_i \ge c_i \quad i \in I \;,\; y \ge 0 \;,\; w_i \ge 0 \quad i \in I \right\} .$$

Dalla teoria della *PL* sappiamo che una coppia di soluzioni $\bar{x}$ e $(\,\bar{y}\,,\,\bar{w}\,)$ è ottima per $(\underline{KP})$ e $(\underline{DKP})$ se e solo è ammissibile e rispetta le condizioni degli scarti complementari, che in questo caso sono

$$\bar{y}\big(\,b - \textstyle\sum_{i\in I} a_i \bar{x}_i\,\big) = 0 \quad , \quad \bar{w}_i(\,1 - \bar{x}_i\,) = 0 \quad , \quad \bar{x}_i(\,\bar{y}a_i + \bar{w}_i - c_i\,) = 0 \qquad i \in I \;.$$

È immediato verificare che la soluzione duale

$$y^* = c_h \,/\, a_h \qquad w_i^* = \begin{cases} c_i - y^* a_i & \text{se } i < h, \\ 0 & \text{altrimenti} \end{cases}$$

è ammissibile, in quanto, essendo gli oggetti ordinati per costi unitari non crescenti, si ha $w_i^* = c_i \,/\, a_i - c_h \,/\, a_h \ge 0$. È anche facile controllare che $(\,y^*\,,\,w^*\,)$ verifica le condizioni degli scarti complementari con x^*: per la prima condizione si ha $\sum_{i\in I} a_i x_i^* = b$, per la seconda condizione si ha che $w_i^* > 0$ solo per per gli indici $i < h$ per cui $x_i^* = 1$, per la terza condizione si ha che $x_i^* > 0$ al più per gli indici $i \le h$ (può essere $x_h^* = 0$) per cui $y^* a_i + w_i^* = c_i$.

Possiamo adesso procedere a valutare *a posteriori* l'errore relativo commesso da CUD. Infatti si ha $z(KP) \le \sum_{i\in I} c_i x_i^* = \sum_{i<h} c_i + c_h(\,b - \sum_{i<h} a_i\,) \,/\, a_h$ e $\sum_{i<h} c_i \le z_{CUD} \le z(KP)$, da cui

$$R_{CUD} = \frac{z(KP) - z_{CUD}}{z(KP)} \le \frac{c_h(\,b - \sum_{i<h} a_i\,) \,/\, a_h}{\sum_{i<h} c_i} \le \frac{c_h}{\sum_{i<h} c_i} \;. \tag{5.1}$$

Si noti che se $\sum_{i<h} a_i = b$, ossia l'ultimo oggetto (nell'ordine dei costi unitari non crescenti) che entra interamente nello zaino ne satura la capacità, allora $R_{CUD} = 0$, ossia CUD determina una soluzione ottima del problema. Infatti, in questo caso si ha che la soluzione prodotta da CUD è proprio x^*, in quanto x_h^* (l'unica componente di x^* che può assumere valore frazionario) vale 0. Quindi, la soluzione ottima del rilassamento continuo di (KP) ha valori interi, ossia è una soluzione ammissibile per (KP); il Lemma 4.1 garantisce quindi che x^* sia una soluzione ottima per (KP).

Esempio 5.4. Stime dell'errore per Greedy-CUD

Consideriamo la seguente istanza del problema dello zaino:

$$\begin{array}{rrcrcrcrcl} \max & 11x_1 & + & 8x_2 & + & 7x_3 & + & 6x_4 & & \\ & 5x_1 & + & 4x_2 & + & 4x_3 & + & 4x_4 & \leq & 12 \quad . \\ & x_1 & , & x_2 & , & x_3 & , & x_4 & \in & \{\,0\,,\,1\,\} \end{array}$$

Gli oggetti sono già ordinati per costo unitario non crescente. L'algoritmo CUD riporta la soluzione $S = \{\,1\,,\,2\,\}$ di costo 19, mentre la soluzione ottima è $\{\,2\,,\,3\,,\,4\,\}$ di costo 21: l'errore relativo commesso da CUD in questo caso è $(\,21-19\,)\,/\,21 \approx 0.095$, ossia del 9.5%. La soluzione ottima di $\underline{KP}$ è $x^* = [\,1\,,\,1\,,\,3/4\,,\,0\,]$ di costo $97/4$ $(= 24+1/4)$; ad essa corrisponde infatti la soluzione duale $y^* = 7/4$, $w^* = [\,9/4\,,\,1\,,\,0\,,\,0\,]$, anch'essa di costo $97/4$. Utilizzando questa valutazione superiore su $z(KP)$ per stimare l'errore si ottiene $R_{CUD} \leq (\,97/4 - 19\,)\,/\,19 \approx 0.276$, ossia il 27.6%. Utilizzando la formula che dipende solamente dai costi e da h si ottiene $R_{CUD} \leq 7/(11+8) \approx 0.368$.

L'analisi precedente mostra anche quale sia il caso "critico" per questo algoritmo, ossia quello in cui il primo oggetto che non entra nello zaino ha un costo molto elevato rispetto a tutti quelli che sono entrati fino a quel momento. Sulla base della valutazione (5.1) si possono quindi facilmente costruire istanze in cui l'errore complessivo è arbitrariamente alto. Questa analisi però suggerisce anche una piccola modifica all'algoritmo Greedy-CUD che permette di trasformarlo in modo da ottenere una valutazione $R_{CUD} \leq 1$ al caso pessimo. La modifica è semplice: al momento in cui si determina h, si controlla se $c_h > \sum_{i<h} c_i$. Se così non è, (5.1) mostra già che $R_{CUD} \leq 1$. Se invece questo capita, è facile modificare la soluzione dell'euristica in modo da migliorarla: semplicemente, si eliminano tutti gli elementi $i < h$ e si aggiunge l'unico elemento h. La soluzione è sicuramente ammissibile ($a_h \leq b$), e chiaramente migliore di quella disponibile (almeno in termini della funzione obiettivo, anche se l'occupazione totale potrebbe essere minore). Inoltre, il fatto che $z(KP) \leq \sum_{i<h} c_i + c_h$ mostra immediatamente che per la soluzione $x_h = 1$, $x_i = 0$ per $i \neq h$ ha $R_{CUD} \leq (\,z(KP) - c_h\,)\,/\,c_h < 1$ (per via del fatto che $\sum_{i<h} c_i < c_h$. Quindi, l'analisi dell'errore compiuto da un algoritmo può consente di identificare modifiche che ne migliorano le prestazioni, almeno al caso pessimo.

Esercizio 5.10. Dimostrare che vale la relazione

$$z(KP) \leq max\{\,U'\,,\,U''\,\} \leq \textstyle\sum_{i<h} c_i + c_h\big(\,b - \sum_{i<h} a_i\,\big)/a_h \quad \text{, dove}$$

$$U' = \textstyle\sum_{i<h} c_i + c_{h+1}\big(\,b - \sum_{i<h} a_i\,\big)/a_{h+1} \quad \text{e} \quad U'' = \sum_{i\leq h} c_i + c_{h-1}\big(\,b - \sum_{i\leq h} a_i\,\big)/a_{h-1} \ ,$$

ossia che il massimo tra U' ed U'' fornisce una valutazione superiore di $z(KP)$ non peggiore di quella utilizzata in (5.1) (suggerimento: U' ed U'' sono i valori dele soluzioni ottime di due opportuni rilassamenti ottenuti supponendo che x_h sia rispettivamente 0 e 1 in una soluzione ottima di KP).

5.1.2.2 Gli algoritmi SPT e LPT per (MMMS)

Come per il caso del problema dello zaino, per poter valutare l'errore compiuto dagli algoritmi SPT e LPT per (MMMS) dobbiamo innanzitutto determinare una valutazione—in questo caso inferiore—-sul valore ottimo del problema. Per (MMMS) utilizziamo la relazione

$$z(MMMS) \geq L = \textstyle\sum_{i\in T} d_i\,/\,p$$

che discende immediatamente dal fatto che il caso migliore è quello in cui i lavori sono perfettamente ripartiti tra le p macchine, che terminano tutte allo stesso momento (si ha "speedup lineare").

Iniziamo l'analisi dimostrando una valutazione *a priori* che vale per qualsiasi algoritmo di tipo list scheduling, ossia indipendentemente dall'ordine con cui sono assegnati i lavori: in ogni caso, l'errore relativo compiuto non supera $(\,p-1\,)\,/\,p$. Si noti che l'errore peggiora all'aumentare del numero di macchine, tendendo a 1 (cioè al 100%) quando p cresce. Per dimostrare il risultato occorre definire alcune quantità critiche: in particolare, indichiamo con h il lavoro che termina per ultimo (l'ultimo lavoro eseguito sulla macchina che termina per ultima) e con $H = (\,\sum_{i\in T} d_i - d_h\,)\,/\,p$: H è il minimo tempo possibile di completamento per le macchine se h fosse rimosso dalla lista dei lavori. Chiamiamo z_{LS} il valore della funzione obiettivo della soluzione determinata da un generico algoritmo di list scheduling: l'osservazione fondamentale è che risulta $z_{LS} \leq H + d_h = L + d_h(\,p-1\,)\,/\,m$. Infatti, quando il lavoro h viene assegnato alla macchina "più scarica" rispetto alla soluzione corrente al

momento in cui h è esaminato: è quindi ovvio che la situazione peggiore, quella alla quale corrisponde il maggior valore di z_{LS}, è quella in cui h è anche l'ultimo lavoro ad essere assegnato, e le macchine erano "perfettamente bilanciate" al momento in cui h è stato assegnato (si veda la figura 5.3). Da questo e da $z(MMMS) \geq L$ otteniamo

$$R_{LS} = \frac{z_{LS} - z(MMMS)}{z(MMMS)} \leq \frac{L + d_h(\,p-1\,)\,/\,p - L}{z(MMMS)}$$

Ma siccome chiaramente $z(MMMS) \geq d_h$ (almeno il tempo necessario a completare d_h deve passare) si ha $R_{LS} \leq (\,p-1\,)\,/\,p$.

Osserviamo che questa valutazione non è "ottimistica", ossia non è possibile migliorarla. Infatti, esiste almeno una classe di istanze ed un ordinamento, in particolare SPT, per cui si ottiene una soluzione che ha esattamente errore relativo $(\,p-1\,)\,/\,p$. Si consideri infatti l'istanza che contiene $(\,p-1\,)p$ lavori di lunghezza unitaria ed un solo lavoro di lunghezza p: l'algoritmo SPT assegna i lavori di lunghezza unitaria per primi, ottenendo $p-1$ macchine con tempo di completamento $p-1$ ed una singola macchina con tempo di completamento $2p-1$. In figura 5.3 è mostrato un esempio per il caso $p = 4$. Ma l'assegnamento ottimo consiste invece nell'assegnare ad una macchina l'unico lavoro di lunghezza p e distribuire poi uniformemente sulle altre i restanti $(\,p-1\,)p$: in questo caso tutte le macchine terminano in tempo p. È facile vedere che l'algoritmo LPT applicato all'istanza "critica" produce la soluzione ottima.

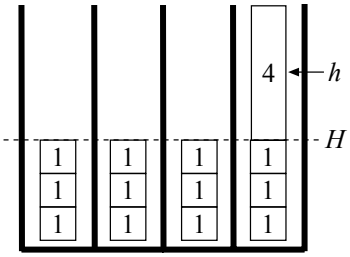


Figura 5.3: Valutazione dell'errore per (MMMS)

In effetti, la valutazione precedente è valida per qualsiasi algoritmo di tipo list scheduling, ma se utilizziamo il particolare ordinamento LPT è possibile dimostrare una valutazione migliore, ossia che $R_{LPT} \leq (\,p-1\,)\,/\,3p$. Per semplicità di notazione ci limiteremo a considerare il caso $p = 2$ (per cui $R_{LPT} \leq 1/6$) e supporremo che i lavori siano forniti in input già ordinati per durata non crescente, ossia $d_1 \geq d_2 \geq \ldots \geq d_n$. La dimostrazione è per assurdo: supponiamo che esista un'istanza per cui l'errore relativo sia maggiore di 1/6. Consideriamo in particolare l'istanza I *con* n *minimo* per cui LPT ottiene un errore relativo maggiore di 1/6: in questa istanza si ha $h = n$, ossia l'ultimo lavoro completato è anche quello di durata minore. Infatti, se fosse $h < n$ potremmo eliminare tutti gli oggetti di indice maggiore di h, che chiaramente non contribuiscono alla determinazione del tempo di completamento (essendo h quello che termina per ultimo): otterremmo così un'istanza I' con meno lavori e con errore relativo non minore di quello di I, contraddicendo l'ipotesi che I sia l'istanza con meno lavori che ha errore relativo maggiore di 1/6. Sia quindi z_{LPT} il tempo di completamento della soluzione ottenuta dall'algoritmo LPT sull'istanza I: per ipotesi abbiamo

$$R_{LPT} \geq \frac{z_{LPT} - z(MMMS)}{z(MMMS)} > \frac{1}{6}$$

da cui $z_{LPT} > 7z(MMMS)/6$. Siccome LPT è un particolare algoritmo di list scheduling, vale

$$z_{LPT} \leq H + d_h = L + d_h/2 \leq z(MMMS) + d_n/2$$

si ha quindi $7z(MMMS)/6 < z(MMMS) + d_n/2$, ossia $z(MMMS) < 3d_n$. Siccome d_n è il più corto dei lavori, dalla relazione precedente si ottiene che $n \leq 4$: qualsiasi istanza con almeno 5 lavori di lunghezza almeno d_n ha tempo di completamento non inferiore a $3d_n$. È però facile verificare che per istanze con non più di 4 lavori LPT fornisce la soluzione ottima, da cui l'assurdo.

Esercizio 5.11. Si dimostri che qualsiasi istanza con al più 4 lavori viene sicuramente risolta all'ottimo da LPT (suggerimento: i casi con 1, 2 e 3 lavori sono banali; chiamando $a \geq b \geq c \geq d$ le lunghezze dei 4 lavori, i primi tre sono sempre assegnati da LPT nello stesso modo—a da solo sulla prima macchina, b e c insieme sulla seconda—mentre d viene assegnato in modo diverso a seconda che risulti $a \geq b + c$ oppure $a < b + c$).

Esercizio 5.12. Si estenda la dimostrazione della valutazione superiore dell'errore relativo di LPT al caso generale $p > 2$.

5.1.2.3 Algoritmo "twice around MST" per il (TSP)

Consideriamo nuovamente il problema del commesso viaggiatore (TSP), per il quale abbiamo già descritto euristiche greedy senza garanzie sulle prestazioni. Definiamo adesso euristiche greedy completamente diverse per le quali è possibile stabilire una limitazione all'errore relativo sotto opportune ipotesi sull'istanza, ossia che i) il grafo G è completo, e ii) i costi sugli archi sono non negativi e rispettano la *diseguaglianza triangolare*

$$c_{ij} + c_{jh} \leq c_{ih} \qquad \forall \{ i , j \} , \{ j , h \} , \{ i , h \} \in E \ . \tag{5.2}$$

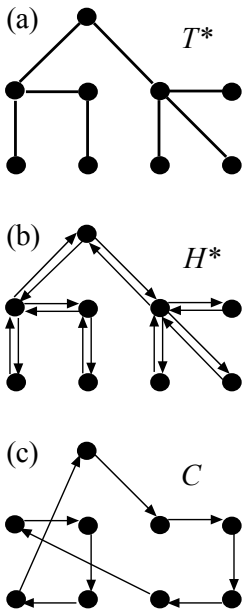


Figura 5.4: "twice around MST"

L'algoritmo si basa sulla seguente osservazione: il costo di un albero di copertura di costo minimo sul grafo G fornisce una valutazione inferiore del costo del ciclo Hamiltoniano ottimo del grafo. Infatti, sia C^* il ciclo Hamiltoniano di costo minimo del grafo, e consideriamo il cammino P^* ottenuto eliminando un qualsiasi arco da C^*: il costo di P^* è non superiore a quello di C^* e non inferiore a quello dell'albero di copertura di costo minimo T^*, essendo P^* un particolare albero di copertura per il grafo. Ma T^* è efficientemente calcolabile (cf. §2.3) e può essere trasformato in un cammino Hamiltoniano che, se i costi soddisfano (5.2), non può avere costo maggiore di due volte il costo originale. La trasformazione è illustrata in Figura 5.4. Per questo si costruisce il grafo orientato H^* ottenuto da T^* trasformando ogni lato in una coppia di archi orientati nelle due direzioni possibili (Figura 5.4(b)): H^* è un *ciclo di copertura per G orientato e non semplice*, il cui costo—considerando i costi degli archi pari a quelli dei lati originali—è pari a due volte quello di T^*. A sua volta H^* può essere trasformato in un ciclo Hamiltoniano (non orientato) C iterando l'operazione di "scorciatoia" come illustrato in Figura 5.4(c). Si inizializza $C = \emptyset$ ed un vertice qualsiasi, ad esempio 1, come l'ultimo vertice i che è stato aggiunto a C; inoltre si definisce il vertice "corrente" h che è stato per ultimo raggiunto da i, con $h = i$ inizialmente. Si seleziona quindi arbitrariamente un arco (h , j) di H^* e lo si rimuove da H^*. Se j è un vertice già visitato si pone $h = j$ e si itera, continuando a visitare H^* a partire da i. Altrimenti si è raggiunto il nuovo vertice j, cosa che si ottiene semplicemente aggiungendo $\{ i , j \}$ a C e ponendo $i = h = j$. Si noti che se $h = i$ si è sostituito il singolo arco (i , j) di H^* con il lato $\{ i , j \}$ di C, altrimenti si sono sostituiti tutti gli archi del cammino tra i e j (che sono stati man mano eliminati da H^*, e dei quali l'ultimo è (h , j)) con il singolo lato $\{ i , j \}$ di C: questa viene detta "scorciatoia" (shortcut), e da (5.2) segue immediatamente che ogni operazione di "scorciatoia" sostituisce un insieme di archi con un lato che ha costo non superiore alla somma dei loro costi. Quando viene visitato l'ultimo vertice, C viene chiuso (come ciclo) aggiungendo il lato fino al vertice iniziale, ossia operando una scorciatoia dall'ultimo vertice visitato al primo. Pertanto al termine del procedimento si ha

$$C(T^*) \leq C(C^*) \leq C(C) \leq C(H^*) \leq 2C(T^*)$$

e quindi l'errore relativo è non superiore a 1. Questo algoritmo prende, per ovvie ragioni, il nome di euristica "twice around MST".

Esercizio 5.13. Si proponga una descrizione formale, in pseudo-codice, dell'euristica "twice around MST".

Esiste una versione di questa euristica, nota come *euristica di Christofides*, che ha una migliore garanzia sulla prestazioni. In breve, il funzionamento dell'euristica "twice around MST" si basa fondamentalmente sul fatto che in H^* tutti i vertici hanno grado pari. L'idea è quindi quella di sostituire H^* con un grafo non orientato in cui ogni vertice ha grado pari e che "costi il meno possibile". Ciò può essere ottenuto semplicemente partendo da T^*, considerando l'insieme dei vertici che hanno grado dispari (che deve avere cardinalità pari) e costruendo il sottografo M^* di costo minimo che collega tali nodi a coppie attraverso la soluzione di un opportuno problema di assegnamento di costo minimo (cf. §2.6.2). Il grafo ottenuto dall'unione di T^* ed M^* ha costo pari al massimo ad $1.5T^*$, e può essere trasformato in un ciclo Haniltoniano C attraverso operazioni di "scorciatoia", come fatto per H^*, che non ne aumentano il costo; pertanto C avrà costo al massimo pari a $1.5T^*$, e quindi l'euristica di Christofides ha errore relativo al caso pessimo di 0.5. Questo esempio mostra come la conoscenza di diversi algoritmi

per problemi "facili" possa essere utile per costruire approcci per problemi più "difficili".

Esercizio 5.14. Si proponga una descrizione formale, in pseudo-codice, dell'euristica di Christofides e si dimostrino formalmente le sue proprietà sopra enunciate.

5.1.2.4 Un algoritmo greedy per il Weighted Vertex Cover

Nei due esempi precedenti abbiamo prima introdotto gli algoritmi e poi, separatamente, valutato le loro prestazioni. È però possibile costruire algoritmi *facendosi guidare* dalle dimostrazioni di efficacia, ossia in modo tale che "naturalmente" se ne possano dimostrare buone proprietà di approssimazione. Esistono alcune tecniche generali per fare questo, una delle quali sarà illustrata per il problema del seguente esempio.

Esempio 5.5. Un problema di selezione di nodi

L'agenzia spionistica EMC, sempre un pò indietro rispetto alla concorrente CIA, ha deciso di dotarsi anch'essa di un sistema per monitorare tutte le comunicazioni di Internet. Per fare questo dispone di una mappa dei "backbones" di Internet attraverso un grafo (non orientato) $G = (V, E)$, in cui i vertici rappresentano i routers mentre i lati rappresentano i collegamenti tra di essi. L'agenzia dispone di apparecchiature che, se installate su un certo router, permettono di monitorare tutte le comunicazioni che transitano attraverso di esso. Installare l'apparecchiatura in un certo vertice i ha però un costo $c_i > 0$, dovuto in parte al costo dell'hardware ed in parte al costo di corrompere o intimidire i gestori del router per convincerli a permetterne l'installazione. L'agenzia dispone dei fondi necessari per installare le apparecchiature in tutti i nodi, ma ciò è evidentemente inutile: per poter monitorare tutte le comunicazioni, è sufficiente che per ogni lato $\{i, j\} \in E$ *almeno* uno dei nodi i e j sia monitorato. Per risparmiare soldi da poter inserire nei propri fondi neri, l'agenzia EMC deve quindi risolvere il seguente problema, detto di *Weighted Vertex Cover* (WVC): selezionare un sottoinsieme di nodi, $S \subseteq V$, di costo minimo (dove $C(S) = \sum_{i \in S} c_i$), che "copra" tutti i lati del grafo, ossia tale che per ogni $\{i, j\} \in E$ risulti vera almeno una delle due condizioni $i \in S$ e $j \in S$.

Un modello analitico per il problema è il seguente:

$$(WVC) \qquad \min \left\{ \sum_{i=1}^{n} c_i x_i \, : \, x_i + x_j \geq 1 \quad \{i, j\} \in E \;\; , \;\; x \in \mathbb{N}^n \right\} .$$

Chiaramente, qualsiasi soluzione ottima x^* del problema avrà solamente componenti 0 e 1, ossia $x^* \in \{0, 1\}^n$, anche se non sono presenti vincoli espliciti $x_i \leq 1$: infatti, per qualsiasi soluzione ammissibile $\bar{x}$ con $\bar{x}_i > 1$ è possibile costruire un'altra soluzione ammissibile x' identica a $\bar{x}$ tranne per il fatto che $x'_i = 1$, e siccome i costi sono positivi x' ha un costo minore di $\bar{x}$.

Vogliamo costruire un algoritmo greedy per (WVC) che abbia buone proprietà di approssimazione: come abbiamo visto nei casi precedenti, per stimare l'errore compiuto è necessario per prima cosa ottenere una valutazione (in questo caso inferiore) del valore ottimo della funzione obiettivo. Come nel caso del problema dello zaino, otteniamo una tale valutazione considerando il *rilassamento continuo* di (WVC)

$$(\underline{WVC}) \qquad \min \left\{ \sum_{i \in V} c_i x_i \, : \, x_i + x_j \geq 1 \quad \{i, j\} \in E \;\; , \;\; x \geq 0 \right\}$$

o, equivalentemente, dal suo duale

$$(\underline{DWVC}) \qquad \max \left\{ \sum_{\{i,j\} \in E} y_{ij} \, : \, \sum_{\{i,j\} \in S(i)} y_{ij} \leq c_i \quad i \in V \;\; , \;\; y \geq 0 \right\} ,$$

dove $S(i)$ è l'insieme dei lati indicenti nel nodo i. Vogliamo costruire un algoritmo per (WVC) che costruisca contemporaneamente una soluzione ammissibile per (WVC) *intera*, e quindi ammissibile per (WVC), ed una soluzione ammissibile per (DWVC) che ci permetta di valutare la bontà della soluzione primale ottenuta. In effetti, ci serviremo della soluzione duale (parziale) per *guidare* la costruzione della soluzione primale: per questo, l'algoritmo viene detto *primale-duale*. Le condizioni degli scarti complementari per la coppia di problemi duali (WVC) e (DWVC) sono

$$x_i \Big(c_i - \sum_{\{i,j\} \in S(i)} y_{ij} \Big) = 0 \qquad i \in V \tag{5.3}$$

$$y_{ij} \big(x_i + x_j - 1 \big) = 0 \qquad \{i, j\} \in E \;\; . \tag{5.4}$$

Se le soluzioni primale e duale ottenute dall'algoritmo rispettassero sia (5.3) che (5.4), allora avremmo determinato una coppia di soluzioni ottime per (WVC) e (DWVC); dato che la soluzione primale sarà costruita in modo da essere ammissibile anche per (WVC), si sarebbe quindi ottenuta una soluzione ottima per (WVC) (si veda il Lemma 4.1). Naturalmente, essendo (WVC) un problema $\mathcal{NP}$-arduo

non è pensabile che ciò sia sempre possibile: per questo l'algoritmo si limiterà ad assicurare che sia verificata (5.3), mentre permetterà violazioni in (5.4).

```
procedure S = Greedy-WVC( G , c )  {
  S = ∅; Q = E;
  foreach( { i , j } ∈ E ) do y_ij = 0;
  do { { i , j } = Next(Q); Q = Q \ { { i , j } };
       y_ij = min{ c_i − Σ_{{i,h}∈S(i)} y_ih , c_j − Σ_{{j,h}∈S(j)} y_jh };
       if( c_i == Σ_{{i,h}∈S(i)} y_ih ) then {
           foreach( { i , h } ∈ E ) do Q = Q \ { { i , h } };
           S = S ∪ { i };
           }
       if( c_j == Σ_{{j,h}∈S(j)} y_jh ) then {
           foreach( { j , h } ∈ E ) do Q = Q \ { { j , h } };
           S = S ∪ { j };
           }
       } while( Q ≠ ∅ );
  }
```

Procedura 5.2: Algoritmo *Greedy-WVC*

L'algoritmo mantiene in Q l'insieme dei lati non ancora "coperti" dalla soluzione corrente S, che inizialmente non contiene alcun nodo. Ad ogni passo seleziona un qualsiasi lato $\{ i , j \}$ non ancora coperto ed aumenta il valore della corrispondente variabile y_{ij} al massimo valore possibile che non viola le condizioni (5.3) per i e j; questo valore è tale per cui, dopo l'aumento di y_{ij}, vale almeno una delle due condizioni

$$c_i = \sum_{\{ i , h \} \in S(i)} y_{ih} \qquad \text{e} \qquad c_j = \sum_{\{ j , h \} \in S(j)} y_{jh} \ .$$

Se vale la prima condizione i viene aggiunto ad S e tutti i lati incidenti in i (che sono a questo punto coperti da S) sono eliminati da Q; analogamente, se vale la seconda condizione j viene aggiunto ad S e tutti i lati incidenti in j sono eliminati da Q. È facile verificare che la soluzione y costruita dall'algoritmo è duale ammissibile ad ogni iterazione: lo è infatti sicuramente all'inizio (dato che i costi sono positivi), e non appena il vincolo duale corrispondente al nodo i diviene "attivo" tutti i lati incidenti in i vengono eliminati da Q, "congelando" i loro valori y_{ij} fino al termine dell'algoritmo e quindi assicurando che il vincolo resti verificato. A terminazione, la soluzione primale S "copre" tutti i nodi ($Q = \emptyset$), ed è quindi ammissibile.

Esempio 5.6. Algoritmo *Greedy-WVC*

Un esempio del funzionamento dell'algoritmo *Greedy-WVC* è mostrato in Figura 5.5. L'istanza è mostrata in (a), con i costi indicati vicino a ciascun nodo. Nelle figure successive, i vertici evidenziati sono quelli selezionati per la cover finale (appartenenti ad S) ed i lati tratteggiati sono quelli eliminati da Q in quanto già coperti dai vertici in S; accanto ai lati viene mostrato il valore della corrispondente variabile y_{ij}, se non nullo.

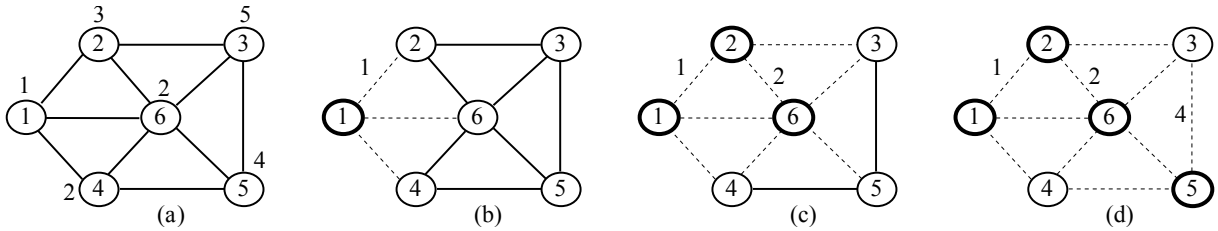


Figura 5.5: Esecuzione dell'algoritmo *Greedy-WVC*

La prima iterazione è mostrata in (b): viene selezionato il lato $\{ 1 , 2 \}$, e si pone $y_{12} = 1$, in modo tale che $y_{12} + y_{16} + y_{14} = 1 + 0 + 0 = 1 = c_1$. Quindi, il nodo 1 viene inserito in S ed i lati $\{ 1 , 2 \}$, $\{ 1 , 6 \}$ e $\{ 1 , 4 \}$ risultano quindi coperti.

La seconda iterazione è mostrata in (c): viene selezionato il lato $\{ 2 , 6 \}$, e si pone $y_{26} = 2$; in questo modo risulta sia $y_{12} + y_{26} + y_{23} = 1 + 2 + 0 = 3 = c_2$ che $y_{16} + y_{26} + y_{36} + y_{46} + y_{56} = 0 + 2 + 0 + 0 + 0 = 2 = c_6$, e quindi sia il nodo 2 che il nodo 6 vengono aggiunti a S, coprendo i corrispondenti lati adiacenti.

La terza ed ultima iterazione è mostrata in (d): viene selezionato il lato $\{ 3 , 5 \}$, e si pone $y_{35} = 4$, in modo che risulti $y_{35} + y_{45} + y_{56} = 4 + 0 + 0 = 4 = c_5$, e quindi che 5 sia aggiunto a S coprendo gli ultimi lati e determinando quindi la soluzione ammissibile $S = \{ 1 , 2 , 5 , 6 \}$ di costo 10.

Valutiamo adesso l'efficacia dell'algoritmo *Greedy-WVC*. L'osservazione fondamentale è che ad ogni passo dell'algoritmo si ha

$$c(S) = \sum_{i \in S} c_i \leq 2 \sum_{\{ i , j \} \in E} y_{ij} \ ;$$

infatti, quando il generico nodo i è stato inserito in S si aveva $c_i = \sum_{\{ i , h \} \in S(i)} y_{ih}$ (e gli y_{ih} non sono più cambiato da quell'iterazione), ma ciascun y_{ij} può contribuire ad al più due sommatorie, quella corrispondente ad i e quella corrispondente a j. Di conseguenza si ha

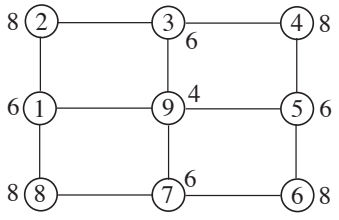
$$\sum_{\{ i , j \} \in E} y_{ij} \leq z(\underline{DWVC}) = z(\underline{WVC}) \leq z(WVC) \leq c(S) \leq 2 \sum_{\{ i , j \} \in E} y_{ij}$$

da cui $R_{Greedy-WVC} \leq 1$; la soluzione ottenuta dall'algoritmo *Greedy-WVC* può costare al più il doppio della soluzione ottima. Quindi, *a priori* possiamo affermare che l'algoritmo *Greedy-WVC* compie al massimo un errore del 100%; questa valutazione può poi essere raffinata *a posteriori* esaminando i risultati per l'istanza specifica. Nel caso di Figura 5.5 si ha ad esempio

$$\sum_{\{ i , j \} \in E} y_{ij} = 7 \leq z(WVC) \leq c(S) = 10$$

da cui $R_{Greedy-WVC} \leq (10 - 7) / 7 \approx 0.428$, ossia la soluzione ottenuta da *Greedy-WVC* è al più il 42.8% più costosa della soluzione ottima. Si noti che anche questa valutazione non è esatta: è possibile verificare (enumerando tutte le possibili soluzioni) che la soluzione ottenuta da *Greedy-WVC* è in effetti *ottima* per l'istanza in questione. La soluzione duale ottenuta non è però in grado di dimostrare l'ottimalità della soluzione primale. Si possono dare tutti i casi tra i due estremi possibili, come gli esempi seguenti mostrano.

Esempio 5.7. Algoritmo *Greedy-WVC*, esempio ottimo



Si vuole determinare una soluzione ammissibile per l'istanza del Weighted Vertex Cover in figura utilizzando l'euristica primale-duale. Si userà la seguente strategia: si seleziona il lato che permette il maggior incremento della variabile duale corrispondente, e tra quelli ai quali corrisponde lo stesso l'incremento si seleziona quello corrispondente alla coppia $\{ i , j \}$ lessicograficamente minima.

Detto $G = (V , E)$ il grafo, la soluzione iniziale è $x_i = 0$ per $i \in V$ (in altri termini, la cover parziale $S = \emptyset$) e $y_{ij} = 0$ per $\{ i , j \} \in E$; inoltre, si pone l'insieme dei lati "non coperti" Q pari ad E. Usiamo la notazione $y(h) = \sum_{\{ h , i \} \in S(h)} y_{hi}$: chiaramente, nella soluzione iniziale si ha $y(h) = 0$ per ogni $h \in V$.

1^a **iterazione**. Si seleziona il lato $\{ 1 , 2 \} \in Q$, e si calcola $\min\{ c_1 - y(1) , c_2 - y(2) \} = \min\{ 8 , 6 \} = 6$; si pone pertanto $y_{12} = 6$. Poiché con la nuova soluzione duale (di valore 6) si ha $c_1 = y(1)$ e $c_2 > y(2)$, si pone $x_1 = 1$ (ossia la cover parziale $S = \{ 1 \}$, di valore 6) e si eliminano da Q tutti i lati incidenti in 1.

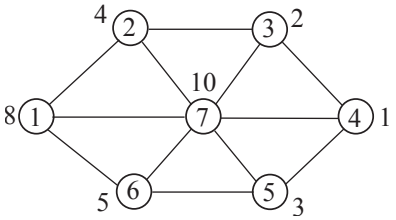
2^a **iterazione**. Si seleziona il lato $\{ 3 , 4 \} \in Q$, e si calcola $\min\{ c_3 - y(3) , c_4 - y(4) \} = \min\{ 6 , 8 \} = 5$; si pone pertanto $y_{34} = 6$. Poiché con la nuova soluzione duale (di valore 12) si ha sia $c_3 = y(3)$ e $c_4 > y(4)$, si pone $x_3 = 1$ (ossia la cover parziale $S = \{ 1 , 3 \}$, di valore 12) e si eliminano da Q tutti i lati incidenti in 3.

3^a **iterazione**. Si seleziona il lato $\{ 5 , 6 \} \in Q$, e si calcola $\min\{ c_5 - y(5) , c_6 - y(6) \} = \min\{ 6 , 8 \} = 6$; si pone pertanto $y_{45} = 6$. Poiché con la nuova soluzione duale (di valore 18) si ha $c_5 = y(5)$ e $c_6 > y(6)$, si pone $x_5 = 1$ (ossia la cover parziale $S = \{ 1 , 3 , 5 \}$, di valore 18) e si eliminano da Q tutti i lati incidenti in 5.

4^a **iterazione**. Si seleziona il lato $\{ 7 , 8 \} \in Q$, e si calcola $\min\{ c_7 - y(7) , c_8 - y(8) \} = \min\{ 6 , 8 \} = 6$; si pone pertanto $y_{78} = 6$. Poiché con la nuova soluzione duale (di valore 24) si ha sia $c_7 = y(7)$ e $c_8 > y(8)$, si pone $x_7 = 1$ (ossia la cover parziale $S = \{ 1 , 3 , 5 , 7 \}$ di valore 24) e si eliminano da Q tutti i lati incidenti in 8.

Poiché Q è vuoto, ossia $S = \{ 1 , 3 , 5 , 7 \}$ è una Vertex Cover, l'algoritmo termina. S ha costo 24 e la soluzione duale ha costo 24: ciò mostra che la cover ottenuta S è ottima. Infatti, ad ogni iterazione è sempre stato inserito in S un singolo nodo, e quindi le condizioni degli scarti complementari (5.4) sono soddisfatte per ogni lato $\{ i , j \} \in E$ (il che, come è noto, è equivalente al fatto che la soluzione del duale e del primale hanno lo stesso valore, come infatti si è già visto). Pertanto la soluzione intera ottenuta è ottima per il rilassamento continuo, e quindi ottima per il problema intero. Incidentalmente, tutto ciò mostra che anche la soluzione del duale ottenuta è una soluzione ottima per quel problema. È anche facile notare che tale soluzione non è unica: infatti la soluzione "simmetrica" $y_{23} = y_{45} = y_{67} = y_{18} = 6$ (tutte le altre variabili a zero) è ammissibile ed ha lo stesso costo.

Esempio 5.8. Algoritmo *Greedy-WVC*, esempio pessimo



Si vuole determinare una soluzione ammissibile per l'istanza del Weighted Vertex Cover in figura utilizzando l'euristica primale-duale. Si userà la seguente strategia: si seleziona il lato che permette il minor incremento della variabile duale corrispondente, e tra quelli ai quali corrisponde lo stesso l'incremento si seleziona quello corrispondente alla coppia $\{ i , j \}$ lessicograficamente massima.

Come al solito la soluzione iniziale è $x_i = 0$ per $i \in V$ (in altri termini, la cover parziale $S = \emptyset$) e $y_{ij} = 0$ per $\{ i , j \} \in E$; inoltre, si pone l'insieme dei lati "non coperti" Q pari ad E. Con la stessa notazione dell'esempio precedente, nella soluzione iniziale si ha $y(h) = 0$ per ogni $h \in V$.

1^a **iterazione**: si seleziona il lato $\{ 7 , 4 \} \in Q$, e si calcola $\min\{ c_4 - y(4) , c_7 - y(7) \} = \min\{ 1 , 10 \} = 1$; si pone pertanto $y_{74} = 1$. Poiché con la nuova soluzione duale (di valore 1) si ha $c_4 = y(4)$ e $c_7 > y(7)$, si pone $x_4 = 1$ (ottenendo quindi la cover parziale $S = \{ 4 \}$ di valore 1) e si eliminano da Q tutti gli archi incidenti in 4.

2^a **iterazione**: si seleziona il lato $\{ 7 , 2 \} \in Q$, e si calcola $\min\{ c_2 - y(2) , c_7 - y(7) \} = \min\{ 4 , 9 \} = 4$; si pone pertanto $y_{72} = 4$. Poiché con la nuova soluzione duale (di valore 5) si ha $c_2 = y(2)$ e $c_7 > y(7)$, si pone $x_2 = 1$ (ottenendo quindi la cover parziale $S = \{ 2 , 4 \}$ di valore 5) e si eliminano da Q tutti gli archi incidenti in 2.

3^a **iterazione**: si seleziona il lato $\{ 7 , 6 \} \in Q$, e si calcola $\min\{ c_6 - y(6) , c_7 - y(7) \} = \min\{ 5 , 5 \} = 5$; si pone pertanto $y_{76} = 5$. Poiché con la nuova soluzione duale (di valore 10) si ha sia $c_6 = y(6)$ che $c_7 = y(7)$, si pone $x_6 = x_7 = 1$ (ottenendo quindi la cover $S = \{ 2 , 4 , 6 , 7 \}$ di valore 20) e si eliminano da Q tutti gli archi incidenti in 6 ed in 7. In questo momento quindi si è violata la condizione degli scarti complementari (5.4) relativamente al lato $\{ 7 , 6 \}$, in quanto $x_6 + x_7 - 1 = 1 \neq 0$ ma $y_{76} = 5 \neq 0$.

Poiché Q è vuoto l'algoritmo termina, avendo individuato la cover $S = \{ 2 , 4 , 6 , 7 \}$ di costo 20; la valutazione relativa a posteriori della bontà di tale soluzione è quindi $(20 - 10) / 10 = 1$ (il caso peggiore possibile).

È facile verificare che la soluzione ottenuta è in effetti ottima: le uniche altre cover possibili sono $\{ 1 , 2 , 3 , 4 , 5 , 6 \}$ e $\{ 1 , 3 , 5 , 7 \}$, entrambe di costo 23. Il gap è quindi interamente dovuto alla debolezza della valutazione inferiore fornita dal rilassamento continuo.

L'idea alla base dell'algoritmo *Greedy-WVC*, ossia quella di costruire contemporaneamente sia una soluzione primale intera che una duale ammissibile per il duale del rilassamento continuo, è generale e può essere applicata per produrre algoritmi con garanzia sul massimo errore compiuto per molti altri problemi combinatori. Questa è comunque soltanto una delle molte tecniche possibili, per ulteriori dettagli ed approfondimenti si rimanda alla letterature citata.

Esercizio 5.15. Il problema (WVC) è un caso particolare del problema di copertura (cf. §1.2.5) in cui tutti gli insiemi F_j hanno esattamente due elementi. Si estenda l'algoritmo primale-duale per (WVC) al problema di copertura: che valutazione può essere data sull'errore commesso?

Esercizio 5.16. Si mostri che l'algoritmo CUD per il problema dello zaino può essere interpretato come un algoritmo primale-duale (suggerimento: per qualsiasi valore di $\bar{y}$, la migliore soluzione $\bar{w}$ duale ammissibile compatibile con quel valore di $\bar{y}$ si ottiene ponendo $\bar{w}_i = c_i / a_i - \bar{y}$ se $c_i / a_i > \bar{y}$ e $\bar{w}_i = 0$ altrimenti).

5.1.3 Matroidi

Nei paragrafi precedenti abbiamo visto che per alcuni algoritmi greedy è possibile ricavare valutazioni, a priori o a posteriori, sull'errore compiuto, e che un algoritmo greedy può anche essere esatto. Si pone quindi il quesito di caratterizzare i problemi per i quali gli algoritmi greedy sono esatti. È possibile rispondere parzialmente alla domanda per una classe particolare di problemi e di algoritmi greedy, ossia facendo tre ulteriori assunzioni sul problema e sull'algoritmo. La prima è:

- $\{ E , F \}$ sono un *sistema di insiemi indipendenti*, ossia $A \in F$ e $B \subseteq A$ implica $B \in F$.

Da questo deriva immediatamente che $\emptyset \in F$; inoltre, possiamo assumere senza perdita di generalità che sia $\{ e \} \in F \ \forall e \in E$, in quanto se fosse $\{ e \} \notin F$ per un qualche $e \in E$ allora nessuno dei sottoinsiemi in F potrebbe contenere e, e quindi potremmo rimuovere e da E (ciò è analogo a quanto già osservato per gli oggetti nel problema dello zaino). Dato un sistema di insiemi indipendenti $\{ E , F \}$ diciamo che S è *massimale* per E se $S \in F$ e $S \cup \{ e \} \notin F$ per ogni $e \in E \setminus S$, ossia se non esiste nessun elemento di F che contiene strettamente S. Più in generale, S è massimale per $E' \subseteq E$ se $S \in F$, $S \subseteq E'$ e $S \cup \{ e \} \notin F$ per ogni $e \in E' \setminus S$. La seconda assunzione è:

- a ciascun elemento $e_i \in E$ è associato un costo $c_e \geq 0$, ed il problema da risolvere è

$$\text{(MSII)} \qquad \max \left\{ c(S) = \sum_{e \in S} c_e \,:\, S \text{ massimale per } E \right\} .$$

La funzione obiettivo *non* è di questo tipo in alcuni dei problemi per i quali abbiamo presentato algoritmi greedy, come ad esempio (MMMS) e (MCMS). Infatti, anche per via di questa assunzione, la teoria sviluppata in questo paragrafo non copre tutti i casi di algoritmi greedy che determinano l'ottimo di un problema di *OC*: il problema (MCMS), che è risolto all'ottimo da un algoritmo greedy (cf. §5.1.1.5), non ha una funzione obiettivo di questo tipo. Un esempio di problema che rispetta le assunzioni è invece (MST), in quanto, dato un grafo non orientato e connesso $G = (V, E)$, la coppia (E, F) dove F è la famiglia dei sottoinsiemi di lati che non inducono cicli su G è chiaramente un sistema di insiemi indipendenti, e gli alberi di copertura per G sono chiaramente i suoi insiemi massimali. Per un problema nella forma (MSII) esiste un'implementazione "naturale" della funzione $Best()$; la terza assunzione è infatti

- la funzione $Best()$ dell'algoritmo ritorna l'elemento $e \in Q$ di costo c_e massimo.

Si noti come molte delle funzioni *Best* che abbiamo discusso *non* rientrano in questa categoria, quali ad esempio quelle per (TSP). L'algoritmo di Kruskal è invece un esempio di algoritmo greedy che rispetta le assunzioni (se invertiamo il segno dei costi o, alternativamente, vogliamo risolvere il problema dall'albero di copertura di costo massimo) e che determina la soluzione ottima per il problema. Vogliamo adesso caratterizzare i problemi di tipo (MSII) per cui un algoritmo greedy di questo tipo è esatto.

Un sistema di insiemi indipendenti (E, F) è detto *matroide* se tutti gli insiemi massimali hanno la stessa cardinalità; più precisamente, si richiede che

$$\forall E' \subseteq E, \text{ se } I \text{ e } J \text{ sono massimali per } E', \text{ allora } |I| = |J|. \tag{5.5}$$

La proprietà (5.5) è *necessaria* affinchè l'algoritmo greedy possa risolvere (MSII) per *qualsiasi* scelta dei costi c_i. Infatti, supponiamo che la proprietà non valga, ossia siano I e J due insiemi massimali rispetto ad un certo $V \subseteq E$ tali che $|I| < |J| \leq n$. Poniamo allora $c_e = 1 + \epsilon$ per $e \in I$, $c_e = 1$ per $e \in J \setminus I$, e $c_e = 0$ per tutti gli altri elementi di E. L'algoritmo greedy pone in S inizialmente tutti gli elementi di I, poi esamina e scarta tutti gli elementi di $J \setminus I$ ($V \supseteq I \cup J$ e I è massimale per V), infine eventualmente aggiunge ad S altri elementi di costo nullo, ottenendo una soluzione di costo $|I|(1+\epsilon)$, ma la soluzione J di costo $|J| \geq |I|+1$ è migliore di I se scegliamo $\epsilon < 1/|I|$. Si noti che dalla proprietà (5.5) segue che l'assunzione $c_e \geq 0$ può essere fatta senza perdita di generalità: come abbiamo visto per il caso di (MST), dato che ogni soluzione ammissibile del problema ha la stessa cardinalità è possibile sommare al costo di ogni elemento un'opportuna costante C lasciando invariato l'insieme delle soluzioni ottime.

Dato un sistema di insiemi indipendenti (E, F), il *rango* di un qualsiasi insieme $E' \subseteq E$ è

$$rango(E') = \max \left\{ |S| \,:\, S \text{ massimale per } E' \right\} ;$$

se (E, F) è un matroide, allora la funzione $rango()$ può essere calcolata facilmente.

Esercizio 5.17. Si dimostri che l'algoritmo greedy con costi $c_e = 1$ per $e \in E'$ e $c_e = 0$ altrimenti determina una soluzione S tale che $c(S) = rango(E')$ (suggerimento: se al termine dell'algoritmo $S \cup E'$ non fosse massimale per E', allora dovrebbe esistere un $S' \subseteq E'$ con $|S'| > |S|$; si consideri cosa accade al momento in cui l'algoritmo greedy esamina l'ultimo elemento di $S \cup S'$ e si usi (5.5) per ottenere una contradizione).

Quindi, l'algoritmo greedy risolve all'ottimo almeno alcuni problemi di tipo (MSII). Vogliamo ora mostrare che l'algoritmo greedy risolve all'ottimo tutti i problemi di tipo (MSII) per qualsiasi scelta dei costi; un modo interessante per farlo è quello di considerare la seguente formulazione *PLI* di (MSII)

$$\text{(MSII-PLI)} \qquad \begin{array}{lll} \max & \sum_{e \in E} c_e x_e & \\ & \sum_{e \in S} x_e \leq rango(S) & \emptyset \subset S \subseteq E \\ & x_e \in \mathbb{N} & e \in E \end{array}$$

Esercizio 5.18. Si verifichi che tutte le soluzioni ammissibili di (MSII-PLI) hanno $x_e \in \{ 0 , 1 \}$ per ogni $e \in E$ e che sono tutti e soli i vettori di incidenza degli elementi di F, ossia vettori nella forma

$$x_e = \begin{cases} 1 & \text{se } e \in S \\ 0 & \text{altrimenti} \end{cases}$$

per un qualche $S \in F$ (suggerimento: per $S = \{ e \} \in F$ il vincolo diviene $x_e \leq 1$, e per ogni $S \in F$ si ha $rango(S) = | S |$ mentre per $S \notin F$ si ha $rango(S) < | S |$).

Come abbiamo visto nei §5.1.2.1 e 5.1.2.4, consideriamo il rilassamento continuo di (MSII-PLI), (MSII-PLI), ottenuto sostituendo il vincolo $x_e \in \mathbb{N}$ con $x_e \geq 0$, ed il suo duale

$$\text{(DMSII)} \qquad \begin{array}{lll} \min & \sum_{S \subseteq E} rango(S) \, y_S & \\ & \sum_{S \, : \, e \in S} y_S \geq c_e & e \in E \\ & y_S \geq 0 & \emptyset \subset S \subseteq E \end{array} \quad .$$

Possiamo mostrare che l'algoritmo greedy costrusce una soluzione primale ammissibile *intera* per (MSII-PLI) ed una soluzione duale ammissibile per (DMSII) che rispettano le condizioni degli scarti complementari

$$x_e \big(\sum_{S \, : \, e \in S} y_S - c_e \big) = 0 \qquad e \in E \ , \tag{5.6}$$

$$y_S \big(rango(S) - \sum_{e \in S} x_e \big) = 0 \qquad \emptyset \subset S \subseteq E \ . \tag{5.7}$$

Per semplificare la notazione, supponiamo che sia $E = \{ 1 , 2 , \ldots , n \}$ e che gli oggetti siano ordinati per costo non crescente, ossia $c_1 \geq c_2 \geq \ldots \geq c_n \geq 0$. Introduciamo inoltre gli insiemi $S(1) = \{ 1 \}$, $S(2) = \{ 1 , 2 \}$, $\ldots$, $S(e) = \{ 1 , 2 , \ldots , e \}$. Possiamo allora riscrivere l'algoritmo greedy sotto forma di un algoritmo primale-duale per la coppia (MSII-PLI), (DMSII):

```
procedure ( S , y ) = Greedy-PD( E , F ) {
  y_{1} = c_1; S = S(1) = {1};                    // x_1 = 1
  for( e = 2, ..., n ) do {
    y_{S(e)} = c_e; y_{S(e-1)} = y_{S(e-1)} - c_e;
    if( S ∪ {e} ∈ F ) then S = S ∪ {e};           // x_e = 1 */
                    /* else                          x_e = 0 */
  }
}
```

Procedura 5.3: Algoritmo *Greedy-PD*

È facile verificare che, data l'assunzione sulla funzione *Best*() e l'ordinamento di E, la procedura *Greedy-PD* produce la stessa soluzione S della procedura *Greedy*. Inoltre, alla generica iterazione e la soluzione primale x (implicitamente) calcolata è ammissibile e rispetta le condizioni (5.6) e (5.7) con la soluzione duale y (assumendo ovviamente $y_S = 0$ per tutti gli S ai quali non è esplicitamente dato un valore diverso), in quanto si ha che:

- $S \subseteq S_e$;
- siccome (E , F) è un matroide, $rango(S) = rango(S(e))$;
- $\sum_{S \, : \, h \in S} y_S = c_h$ per $h = 1, \ldots, e$.

All'iterazione e viene soddisfatto il vincolo duale relativo all'elemento e, senza violare nessuno dei vincoli relativi agli elementi $h < e$; di conseguenza, a terminazione la soluzione duale y costruita dall'algoritmo è ammissibile, e quindi dimostra che la soluzione primale S ottenuta è ottima per (MSII-PLI).

Esempio 5.9. Algoritmo *Greedy-PD*

Esemplifichiamo i concetti della dimostrazione applicando l'algoritmo *Greedy-PD* al semplice problema di (MST) su un grafo completo con 3 nodi in cui $c_{12} = 6$, $c_{13} = 3$ e $c_{23} = 2$. In questo caso l'insieme E è l'insieme dei lati del grafo; per semplificare la notazione chiameremo $a = \{ 1 , 2 \}$, $b = \{ 1 , 3 \}$ e $c = \{ 2 , 3 \}$, per cui $E = \{ a , b , c \}$. I

problemi (MSII) e (DMSII) in questo caso sono

$$\begin{array}{lrcl}\max & 6x_a + 3x_b + 2x_c & & \\ & x_a + x_b + x_c & \leq & 2 \\ & x_a + x_b & \leq & 2 \\ & x_a + x_c & \leq & 2 \\ & x_b + x_c & \leq & 2 \\ & x_a & \leq & 1 \\ & x_b & \leq & 1 \\ & x_c & \leq & 1 \\ & x_a \ , \ x_b \ , \ x_c & \geq & 0\end{array} \qquad \begin{array}{lrcl}\min & 2y_{abc} + 2y_{ab} + 2y_{ac} + 2y_{bc} + y_a + y_b + y_c & & \\ & y_{abc} + y_{ab} + y_{ac} + y_a & \geq & 6 \\ & y_{abc} + y_{ab} + y_{bc} + y_b & \geq & 3 \\ & y_{abc} + y_{ac} + y_{bc} + y_c & \geq & 2 \\ & y_{abc} \ , \ y_{ab} \ , \ y_{ac} \ , \ y_{bc} \ , \ y_a \ , \ y_b \ , \ y_c & \geq & 0\end{array}$$

All'inizializzazione si pone $x_a = 1$ e $y_{\{a\}} = 6$. Alla prima iterazione si pone $y_{\{a,b\}} = 3$ e $y_{\{a\}} = 6 - 3 = 3$; siccome b non forma cicli, si pone $x_b = 1$. Alla terza iterazione si pone $y_{\{a,b,c\}} = 2$ e $y_{\{a,b\}} = 3 - 2 = 1$; siccome c forma un ciclo, si pone $x_c = 0$. È immediato verificare che y è duale ammissibile: tutti i vincoli del duale sono soddisfatti come uguaglianza. È anche facile verificare che le condizioni degli scarti complementari sono rispettate, ma più semplicemente si può notare che il costo della soluzione primale è $6 + 3 = 9$ ed il costo della soluzione duale è $2 \cdot 2 + 2 \cdot 1 + 1 \cdot 3 = 9$.

Un interessante corollario dell'analisi appena svolta è il seguente:

Corollario 5.1. Se (E, F) è un matroide allora (MSII) gode della proprietà di integralità.

Quindi, i problemi di ottimizzazione su matroidi godono di una proprietà analoga a quella dei problemi di flusso (si veda il Teorema 2.12): esiste una formulazione di *PLI* "naturale" per il problema il cui rilassamento continuo ha sempre soluzioni intere, ossia tutti i vertici del poliedro sono interi. A differenza dei problemi di flusso, la formulazione è di dimensione esponenziale, ma, come abbiamo visto nel §4.2.3, ammette un separatore polinomiale.

I matroidi sono dunque strutture combinatorie i cui corrispondenti problemi di ottimizzazione sono risolti esattamente dall'algoritmo greedy. Esempi di matroidi sono:

- E è l'insieme degli archi di un grafo non orientato ed F è la famiglia dei sottoinsiemi di archi che non inducono cicli: questo viene detto un *matroide grafico*;
- $E = \{ A_1, A_2, \ldots, A_n \}$ con $A_i \in \mathbb{R}^m$, e F è la famiglia di tutti gli insiemi di vettori di E linearmente indipendenti: questo viene detto un *matroide matrico*;
- E è un insieme, $P = \{ E_1, E_2, \ldots, E_m \}$ è una sua partizione ed $F = \{ S \subseteq E : |I \cap E_j| \leq 1 \quad j = 1, \ldots, m \}$; questo viene detto un *matroide di partizione*.

Naturalmente molte strutture combinatorie *non* sono matroidi: si consideri ad esempio il caso in cui E è l'insieme degli archi di un grafo bipartito non orientato $G = (O \cup D, E)$ ed F è la famiglia dei suoi accoppiamenti (si veda il §2.6). È facile verificare che questo non è un matroide; infatti, non tutti gli insiemi indipendenti massimali hanno la stessa cardinalità. Del resto, abbiamo visto che per risolvere i problemi di accoppiamento sono necessari algoritmi "più complessi" del semplice greedy. È interessante notare però che i nodi appartenenti a O ed a D inducono su E due diverse partizioni

$$P' = \{ S(i), \ i \in O \} \qquad P'' = \{ S(i), \ i \in D \}$$

corrispondenti alle stelle dei due insiemi di nodi. A partire da tali partizioni possiamo definire due matroidi di partizione (E, F') ed (E, F''): chiaramente $F = F' \cap F''$, ossia il sistema di insiemi indipendenti (E, F) che rappresenta gli accoppiamenti è l'intersezione dei due matroidi (E, F') e (E, F''), da cui si deduce che, in generale, l'intersezione di due matroidi *non* è un matroide. Per quanto gli algoritmi greedy non siano esatti per i problemi di accoppiamento, esistono algoritmi polinomiali per risolverli; in effetti si può dimostrare che qualsiasi problema di ottimizzazione che può essere espresso come l'*intersezione* di *due* matroidi ammette algoritmi polinomiali che ricalcano il funzionamento degli algoritmi per i problemi di accoppiamento visti al §2.6.

Abbiamo anche già osservato al §1.10 che il (TSP) può essere visto come un problema di ottimizzazione sull'intersezione di un problema di accoppiamento e di uno di albero minimo; in altri termini, (TSP) è problema di ottimizzazione sull'intersezione di *tre* matroidi. Ciò dimostra che i problema di ottimizzazione sull'intersezione di tre matroidi sono invece, in generale, $\mathcal{NP}$-ardui.

5.2 Algoritmi di ricerca locale

Gli algoritmi di ricerca locale sono basati su un'idea semplice ed intuitiva: data una "buona" soluzione ammissibile, è possibile che ce ne siano altre "simili ad essa" che sono ancora migliori. Pertanto, a partire ad una qualche soluzione ammissibile si esaminano quelle ad essa "vicine" in cerca di una "migliore" (tipicamente, con miglior valore della funzione obiettivo): se una tale soluzione viene trovata essa diventa la "soluzione corrente" ed il procedimento viene iterato, altrimenti—ossia quando nessuna delle soluzioni "vicine" è migliore di quella corrente—l'algoritmo termina avendo determinato un *ottimo locale* per il problema (si consideri la Figura 4.3). Elemento caratterizzante di un algoritmo di questo tipo è la definizione di "vicinanza" tra le soluzioni. In generale, se F è l'insieme ammissibile del problema in esame, possiamo definire una *funzione intorno* $I : F \to 2^F$: l'insieme $I(x)$ è detto *intorno* di x, e contiene le soluzioni considerate "vicine" ad x. Sulla base di $I(x)$ si definisce poi opportunamente una *funzione di transizione* $\sigma : F \to F$ che seleziona un elemento "promettente" $\sigma(x) = y \in I(x)$ all'interno dell'intorno come successiva iterata dell'algoritmo di ricerca locale, che può quindi essere schematizzato come segue:

procedure $x =$ *Ricerca_Locale*(F, c, x)
  while$(\sigma(x) \neq x)$ **do** $x = \sigma(x)$;

Procedura 5.4: Algoritmo di Ricerca Locale

L'algoritmo necessita di una soluzione ammissibile x^0 da cui partire: una tale soluzione può essere costruita, ad esempio, usando un algoritmo greedy. Per semplicità e generalità assumiamo che questa venga fornita in input (il modo in cui questa è costruita è tipicamente indipendente da come poi viene effettuata la ricerca locale). A partire da essa, l'algoritmo genera una sequenza di soluzioni ammissibili $\{x^0, x^1, \ldots, x^k, \ldots\}$ tale che $x^{i+1} = \sigma(x^i)$; la tipica implementazione di σ (per un problema di minimo) è

$$\sigma(x) = \text{argmin}\{c(y) : y \in I(x)\} \ . \tag{5.8}$$

In questo modo, ad ogni passo dell'algoritmo viene risolto un problema di ottimizzazione ristretto all'intorno considerato. Più in generale si può definire σ in modo tale che, per ogni x, fornisca un qualsiasi elemento $y \in I(x)$ con $c(y) < c(x)$, se un tale elemento esiste, o x stesso altrimenti; in questo modo, ad ogni passo dell'algoritmo viene risolto un problema decisionale. Si ha quindi $x = \sigma(x)$ quando nell'intorno di x non esiste nessuna soluzione migliore di x. In generale, la soluzione determinata dall'algoritmo di ricerca locale non è ottima per il problema, ma solamente un *ottimo locale* relativamente alla funzione intorno scelta. I è detta una funzione intorno *esatta* per un problema (P) se l'algoritmo di ricerca locale che usa (5.8) è in grado di fornire una soluzione ottima per ogni istanza di (P) e comunque scelto il punto di partenza x_0. Questo modo di operare è estremamente generale, ed infatti moltissimi algoritmi per problemi di ottimizzazione—tra cui quasi tutti quelli che abbiamo descritto nei capitoli precedenti—possono essere classificati come algoritmi di ricerca locale.

Esercizio 5.19. Per tutti gli algoritmi presentati nei capitoli precedenti si discuta se essi possono o no essere considerati algoritmi di ricerca locale, fornendo la definizione delle relative funzioni intorno.

La definizione della funzione intorno è quindi la parte fondamentale della definizione di un algoritmo di ricerca locale. Usualmente, si richiede che I possieda le due seguenti proprietà:

1. $x \in F \Longrightarrow x \in I(x)$;
2. comunque scelti $x \in F$ e $y \in F$ esiste un insieme finito $\{z^0, z^1, \ldots, z^p\} \subseteq F$ tale che $z^0 = x$, $z^i \in I(z^{i-1})$ per $i = 1, \ldots, p$, e $z^p = y$.

La proprietà 1. richiede che ogni soluzione appartenga all'intorno di se stessa; ciò implica in particolare che (5.8) restituisce automaticamente x quando chiamata su un minimo locale. La proprietà 2. richiede che sia teoricamente possibile per l'algoritmo di ricerca locale raggiungere in un numero finito di passi qualsiasi soluzione $y \in F$—ad esempio quella ottima—a partire da qualsiasi altra soluzione $x \in F$—ad esempio quella iniziale x^0. Si noti che la proprietà 2. non garantisce affatto che un algoritmo di ricerca locale, partendo da x, possa effettivamente arrivare a y: per questo sarebbe anche necessario

che $c(z_i) < c(z_{i-1})$ per $i = 1, \ldots, p$, il che in generale non è vero. Inoltre, in pratica si utilizzano anche funzioni intorno che non soddisfano questa proprietà. In generale è ovviamente preferibile (ma anche praticamente sempre vero in pratica) che il massimo numero p di passi nella proprietà 2. sia polinomiale nella dimensione dell'istanza.

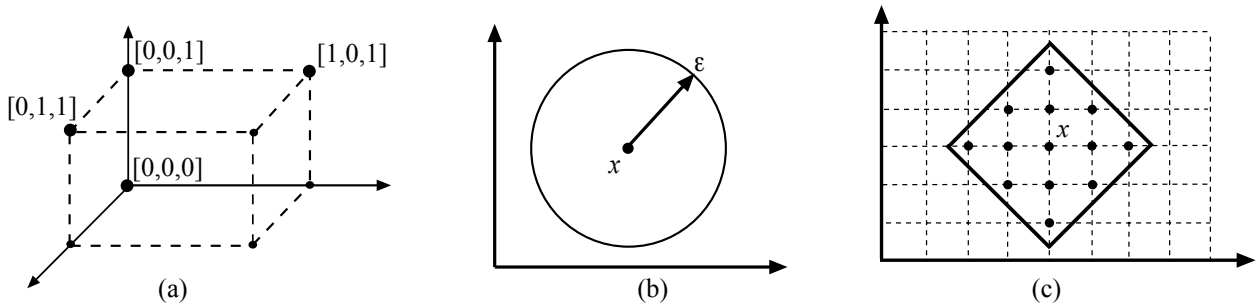


Figura 5.6: Alcuni esempi di funzioni intorno

Alcuni esempi di funzioni intorno sono mostrati in Figura 5.6, in particolare

- Figura 5.6(a): $F \subseteq \{0, 1\}^n$, $I(x) = \{y \in F : \sum_i |x_i - y_i| \leq 1\}$ (è mostrato l'intorno di $[0, 0, 1]$);
- Figura 5.6(b): $F \subseteq \mathbb{R}^n$, $I_\varepsilon(x) = \{y \in F : \|x - y\| \leq \varepsilon\}$ (intorno Euclideo);
- Figura 5.6(c): $F \subseteq \mathbb{Z}^n$, $I(x) = \{y \in F : \sum_i |x_i - y_i| \leq 2\}$ (i punti evidenziati costituiscono l'intorno di x).

I tre intorni precedenti sono molto generali; usualmente, gli intorni utilizzati negli algoritmi di ricerca locale sono più specifici per il problema trattato. Inoltre, spesso la definizione di intorno è fornita in modo implicito, ovvero definendo una serie di operazioni ("mosse") che trasformano una soluzione ammissibile del problema in un'altra soluzione ammissibile.

5.2.1 Esempi di algoritmi di ricerca locale

Discutiamo adesso funzioni intorno per alcuni problemi di *OC*, in modo da fornire una panoramica di alcune delle principali metodologie utilizzate per costruire approcci di ricerca locale.

5.2.1.1 Il problema dello zaino

Si consideri il problema dello zaino (KP). Un primo esempio di funzione intorno per questo problema potrebbe essere identificato dalle seguenti "mosse di inserzione e cancellazione": data una soluzione ammissibile dello zaino, si costruisce una diversa soluzione ammissibile inserendo nello zaino uno degli oggetti attualmente non selezionati—e per il quale esiste ancora sufficiente capacità residua—oppure togliendo dallo zaino uno degli oggetti attualmente selezionati (il che non può che determinare una nuova soluzione ammissibile). Data una soluzione ammissibile x per il problema dello zaino, che possiamo considerare un vettore di n variabili binarie x_i, $i = 1, \ldots, n$, questa funzione intorno, che chiameremo I_1, associa ad x tutte le soluzioni ammissibili che possono essere ottenute trasformando un singolo x_i da 0 a 1 o viceversa; si noti che questa è esattamente la funzione intorno rappresentata in Figura 5.6(a). Tali soluzioni sono quindi al più n. È quindi facile verificare che (5.8) può essere calcolata in tempo lineare $O(n)$.

Esercizio 5.20. Si descriva formalmente, in pseudo-codice, una procedura che calcola la funzione σ, in $O(n)$, per la funzione intorno I_1.

È però facile osservare che questo intorno non permette sicuramente di migliorare le soluzioni ottenute dall'algoritmo *greedy* descritto nel §5.1.1.1. Infatti, al termine di quell'algoritmo lo zaino non ha capacità residua sufficiente per nessuno degli oggetti non selezionati: se l'avesse allora l'avrebbe avuta anche al momento in cui l'oggetto è stato esaminato (la capacità residua è non crescente nel corso dell'algoritmo), e quindi l'oggetto sarebbe stato inserito. Inoltre, per l'ipotesi che i costi siano non negativi togliere un oggetto dallo zaino non può migliorare (aumentare) il valore della fuunzione obiettivo. In altri termini, l'algoritmo *greedy* produce un *ottimo locale* rispetto all'intorno I_1. Se si volesse

utilizzare un algoritmo di ricerca locale per tentare di migliorare la soluzione ottenuta dall'algoritmo *greedy* occorrerebbe quindi definire un intorno diverso. Adesempio, si potrebbe utilizzare la funzione intorno I_2 che, oltre alle mosse di inserzione e cancellazione, utilizza anche "mosse di scambio" tra un oggetto selezionato ed uno non selezionato. In altri termini, la funzione associa ad x tutte le soluzioni (ammissibili) x' che differiscono da x in *al più* due posizioni; se x' differisce da x in i e j, con $i \neq j$, deve essere $x_i = 0$ e $x_j = 1$, e quindi $x'_i = 1$ e $x'_j = 0$. È facile verificare che la funzione σ data da (5.8) per I_2 può essere calcolata in tempo $O(\,n^2\,)$, in quanto il numero di soluzioni $x' \in I(\,x\,)$ è certamente minore o uguale al numero di coppie $\{\,i\,,\,j\,\}$ con i e j in $1, \ldots, n$, e la verifica dell'ammissibilità di una mossa può essere fatta in $O(\,1\,)$.

Esercizio 5.21. Si descriva formalmente, in pseudo-codice, una procedura che calcola la funzione σ, in $O(\,n^2\,)$, per la funzione intorno I_2.

Le soluzioni prodotte dall'algoritmo *greedy* possono *non* essere ottimi locali rispetto a questo nuovo intorno.

Esempio 5.10. Mosse di scambio per il problema dello zaino
Si consideri la seguente istanza del problema dello zaino:

$$\begin{array}{lrcrcrcrcrcl} \max & 2x_1 & + & 8x_2 & + & 5x_3 & + & 6x_4 & + & x_5 & & \\ & 3x_1 & + & 2x_2 & + & 4x_3 & + & 6x_4 & + & 3x_5 & \leq & 8 \\ & x_1 & , & x_2 & , & x_3 & , & x_4 & , & x_5 & \in & \{\,0\,,\,1\,\} \end{array}$$

L'ordine CUD è 2, 3, 4, 1, 5: pertanto, l'euristica Greedy CUD determina la soluzione $x = [\,0\,,\,1\,,\,1\,,\,0\,,\,0\,]$ di valore $cx = 13$. Ma scambiando l'oggetto 3, nello zaino, con l'oggetto 4, fuori dallo zaino, si ottiene la soluzione $x' = [\,0\,,\,1\,,\,0\,,\,1\,,\,0\,]$, pure ammissibile, di valore $cx' = 14$. Pertanto, x non è un ottimo locale rispetto all'intorno I_2.

Esercizio 5.22. Per ciascuno degli altri due ordinamenti (costi non crescenti e pesi non decrescenti), si costruisca (se esiste) un'istanza del problema dello zaino per cui la soluzione prodotta dall'algoritmo greedy con l'ordinamento dato non sia un ottimo locale rispetto all'intorno I_2.

Analizzando ulteriormente le mosse si possono poi scoprire proprietà che risultano utili nell'implementazione dell'algoritmo. Ad esempio, si consideri il caso di due oggetti i e j che hanno lo stesso costo; se hanno anche lo stesso peso allora qualsiasi scambio è inutile. Se invece si ha $c_i = c_j$ e $a_i < a_j$, allora è facile vedere che se esiste una soluzione ottima che contiene j allora esiste anche una soluzione ottima che contiene i al posto di j (la si ottiene semplicemente scambiando i con j). Si può quindi operare in modo tale che in ciascuna soluzione generata siano presenti, tra tutti gli oggetti con un dato costo, solo quelli di peso minore, e quindi in modo tale che non vengano mai scambiati oggetti con lo stesso costo, evitando di valutare mosse "inutili".

La funzione intorno I_2 *domina* la funzione intorno I_1, ossia $I_2(\,x\,) \supseteq I_1(\,x\,)$ per ogni x. In questo caso si può affermare che I_2 è "migliore" di I_1 nel senso che tutti gli ottimi locali rispetto a I_2 sono anche ottimi locali rispetto a I_1, ma il viceversa può non essere vero. Infatti, x è un ottimo locale rispetto a I_2 se $c(\,x'\,) \leq c(\,x\,)$ per ogni $x' \in I_2(\,x\,)$, e questo sicuramente implica che $c(\,x'\,) \leq c(\,x\,)$ per ogni $x' \in I_1(\,x\,)$. Ciò non significa che un algoritmo di ricerca locale che usa I_1 determinerà necessariamente una soluzione peggiore di quella determinata da un algoritmo di ricerca locale che usa I_2, perchè le "traiettorie" seguite dai due algoritmi nello spazio delle soluzioni ammissibili saranno diverse. È però vero che, avendo I_1 "più" minimi locali, un algoritmo di ricerca locale che usa I_1 può arrestarsi una volta giunto ad una certa soluzione dalla quale un algoritmo di ricerca locale che usa I_2 proseguirebbe invece la ricerca, come l'esempio precedente mostra.

È interessante notare come ciascuna mossa di scambio sulla coppia $\{\,i\,,\,j\,\}$ possa essere considerata la *concatenazione* di due mosse, rispettivamente una di cancellazione di j ed una di inserzione di i. Il motivo per cui l'intorno basato sullo scambio è "più potente" è che il risultato della concatenazione delle due mosse viene "visto" immediatamente. Infatti, nessuna mossa di cancellazione presa a sè potrebbe mai essere accettata da un algoritmo di ricerca locale, in quanto comporta (se i costi sono tutti positivi) un peggioramento della funzione obiettivo. Cancellare un oggetto può però permettere di inserirne uno di costo maggiore, ottenendo un miglioramento complessivo del valore della funzione obiettivo; quindi

la concatenazione delle due mosse è conveniente, anche se la prima mossa presa singolarmente non lo è. In generale, intorni basati su "mosse complesse", ottenute concatenando un certo numero di "mosse semplici", permettono di "vedere le conseguenze" delle singole mosse semplici e quindi di effettuare mosse che non sarebbero accettabili da intorni basati direttamente sulle "mosse semplici". Per contro, spesso, come nell'esempio precedente, valutare la funzione σ per intorni basati su "mosse complesse" è più costoso. Inoltre, si noti che questo comportamento è dovuto al fatto che, nell'algoritmo di ricerca locale, si insiste sull'ottenere un miglioramento ad ogni passo. L'osservazione precedente suggerisce due strategie, in qualche modo alternative, per migliorare la qualità delle soluzioni determinate da un algoritmo di ricerca locale:

- aumentare la "complessità" delle mosse, ossia la "dimensione" dell'intorno, per permettere di "vedere" una frazione maggiore dello spazio delle soluzioni ad ogni passo,
- permettere peggioramenti della funzione obiettivo purchè ci sia un "miglioramento a lungo termine".

Nei prossimi paragrafi discuteremo entrambe queste strategie, non prima però di aver introdotto altri esempi di intorno.

5.2.1.2 Ordinamento di lavori su macchine

Per il problema di ordinamento di lavori su macchine con minimizzazione del tempo di completamento—l'altra forma che abbiamo visto è risolubile all'ottimo con un algoritmo greedy e non avrebbe senso discutere procedure di ricerca locale—è facile definire una funzione intorno basata su "mosse" analoghe a quelle viste per il problema dello zaino:

- spostamento: selezionare un lavoro i attualmente assegnato ad una certa macchina h ed assegnarlo ad una macchina $k \neq h$;
- scambio: selezionare un lavoro i attualmente assegnato ad una certa macchina h ed un lavoro j attualmente assegnato ad una certa macchina $k \neq h$, assegnare i alla macchina k ed assegnare j alla macchina h.

Come nel caso del problema dello zaino, una mossa di scambio può essere vista come la concatenazione di due mosse di spostamento, e l'intorno che usa entrambe i tipi di mosse domina l'intorno che usa solamente quelle di spostamento. Si consideri ad esempio il caso di un problema con due macchine e cinque lavori di lunghezza 5, 4, 4, 3 e 2, e la soluzione (determinata da LPT) che assegna $\{ 5 , 3 , 2 \}$ alla prima macchina e $\{ 4 , 4 \}$ alla seconda, con makespan 10. Qualsiasi mossa di spostamento peggiora il makespan della soluzione e quindi non viene accettata; per contro, scambiare il lavoro di lunghezza 5 con uno di quelli di lunghezza 4 produce una soluzione migliore (in particolare ottima).

Esercizio 5.23. Si determini la complessità di valutare (5.8) per l'intorno proposto; si indichi poi come cambia il risultato qualora si effettuino solamente mosse di spostamento.

Questo esempio mostra come per costruire funzioni intorno per un certo problema di *OC* si possa trarre ispirazione da funzioni intorno già viste per altri problemi di *OC*. Ogni problema ha però le sue caratteristiche, che devono essere prese in debita considerazione. Il caso di (MMMS) e quello di (KP) sono diversi in molti aspetti: ad esempio, mentre in (KP) le mosse devono tenere in conto del vincolo di capacità dello zaino, in (MMMS) non ci sono di fatto vincoli, e quindi tutte le mosse producono soluzioni ammissibili. Inoltre, in (KP) tutte le mosse producono una variazione della funzione obiettivo (a meno che non siano scambiati due oggetti dello stesso costo, il che come abbiamo visto può essere evitato), mentre in (MMMS) tutte le mosse che non coinvolgono almeno una delle macchine che determinano il makespan della soluzione non possono migliorare il valore della funzione obiettivo. In effetti, è facile vedere che sono ottimi locali per l'algoritmo di ricerca locale basato sull'intorno così definito tutte le soluzioni in cui ci siano almeno due macchine che determinano il makespan: nessuna mossa di scambio può ridurre il tempo di completamento di due macchine (tipicamente ridurrà il tempo di completamento di una delle macchine coinvolte ed aumenterà quello dell'altra), e quindi migliorare il makespan della soluzione. Per questo è conveniente, nell'algoritmo di ricerca locale,

accettare di spostarsi anche su soluzioni con lo stesso makespan di quella corrente, perché siano migliori per qualche altra caratteristica. Ad esempio, è possibile accettare mosse che diminuiscano il *numero* di macchine che determinano il makespan; ciò corrisponde a minimizzare una funzione obiettivo del tipo $v + \varepsilon ns$, dove v è il valore del makespan, ns è il numero di macchine che hanno tempo di completamento esattamente pari a v e ε è un valore opportunamente piccolo per cui non si preferirà mai diminuire ns a discapito di non diminuire v (quindi $\varepsilon < 1/m$). Questa funzione obiettivo *gerarchica*—la minimizzazione di v è il criterio fondamentale, ma esiste il criterio secondario ns che viene considerato a parità del valore di v—discrimina tra soluzioni con lo stesso makespan, e quindi le soluzioni in cui ci siano almeno due macchine che determinano il makespan non sono più (necessariamente) ottimi locali per l'intorno.

5.2.1.3 Il problema del commesso viaggiatore

Nel problema del commesso viaggiatore (TSP), i costituenti elementari di una soluzione sono lati, e quindi si può pensare, in analogia con gli esempi precedenti, ad operazioni di tipo "scambio" che coinvolgano i lati del ciclo. In questo caso però non è chiaramente possibile operare su un solo lato del grafo, in quanto cancellandone uno qualsiasi si ottiene un cammino (Hamiltoniano), e l'unico modo possibile per trasformare il cammino in un ciclo Hamiltoniano è quello di aggiungere nuovamente il lato appena cancellato. È quindi necessario operare su almeno due lati contemporaneamente: ad esempio, data una soluzione ammissibile x, l'intorno $I(x)$ basato sui "2-scambi" contiene tutti i cicli Hamiltoniani che si possono ottenere da x selezionando due lati $\{i, j\}$ ed $\{h, k\}$ non consecutivi del ciclo e sostituendoli con i lati $\{i, h\}$ e $\{j, k\}$, se esistono. Questa operazione è esemplificata in Figura 5.7.

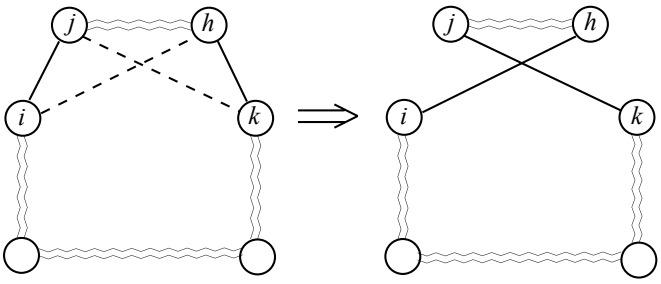


Figura 5.7: Un "2-scambio" per il (TSP)

Esempio 5.11. Ricerca locale per il (TSP)

Si consideri l'istanza in Figura 5.8(a) ed il suo ciclo Hamiltoniano $C = \{1, 2, 3, 4, 5\}$ (lati evidenziati), di costo $c = 14$. Per generare l'intorno basato sul 2-scambio di C è sufficiente enumerare tutte le coppie di lati $\{i, j\}$ ed $\{h, k\}$ non consecutive su C; in questo caso si ottengono i tre cicli Hamiltoniani mostrati in Figura 5.8(b), (c) e (d), ciascuno col corrispondente costo e con indicati i lati di C sostituiti. Quindi, C non è un ottimo locale rispetto all'intorno basato sui 2-scambi (si noti che, per definizione, C appartiene all'intorno): il ciclo $C' = \{1, 2, 3, 5, 4\}$ ha costo minore.

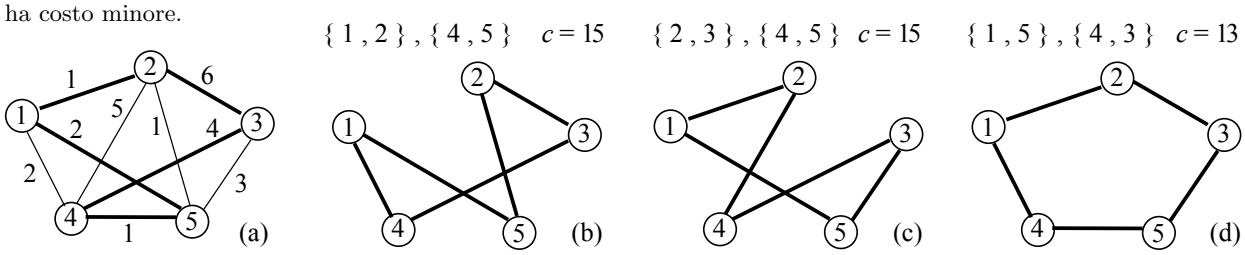


Figura 5.8: Ricerca locale basata sui "2-scambi" per il (TSP)

Il costo di valutare la funzione σ corrispondente a questo intorno è $O(n^2)$, poichè tante sono le coppie di lati non consecutivi del ciclo, e la valutazione del costo del ciclo ottenuto da un 2-scambio può essere fatta in $O(1)$ (conoscendo il costo del ciclo originario). Gli algoritmi di ricerca locale per il TSP attualmente ritenuti più efficienti fanno uso di operazioni di 2-scambio o simili.

5.2.1.4 Il problema del Constrained MST

Si consideri il problema del Constrained MST definito nell'Esempio 4.1. In questo caso le soluzioni ammissibili del problema sono alberi, e si può pensare ad operazioni analoghe a quelle viste in precedenza che coinvolgano lati dell'albero. È però necessario porre attenzione al vincolo sul massimo peso dei sottoalberi della radice. Dato che ogni soluzione ammissibile x del problema è un albero di copertura radicato del grafo, possiamo rappresentarla mediante un vettore $p[\cdot]$ di predecessori. Indicheremo inoltre con $T(i)$ il sottoalbero di radice i della soluzione (se i è una foglia, $T(i) = \{i\}$) e

con $Q(i) = \sum_{h \in T(i)} q_h$ il suo peso: siccome la soluzione è ammissibile si avrà sicuramente $Q(i) \leq Q$ per ogni $i \neq r$, dove r è la radice dell'albero,

Un primo esempio di mossa per il (CMST) è la cosiddetta "Cut & Paste", che consiste nel selezionare un nodo $i \neq r$ ed un nodo $j \notin T(i)$ tale che $j \neq p[i]$ e porre $p[i] = j$. Ciò corrisponde ad eliminare dall'albero il lato $\{p[i], i\}$ e sostituirlo con il lato $\{j, i\}$, ossia a "potare" il sottoalbero $T(i)$ dalla sua posizione ed "innestarlo sotto il nodo j", come mostrato in Figura 5.9(a). Naturalmente, la mossa può essere compiuta solamente se $Q(i) + Q(h) \leq Q$, dove h è il figlio della radice tale che $j \in T(h)$.

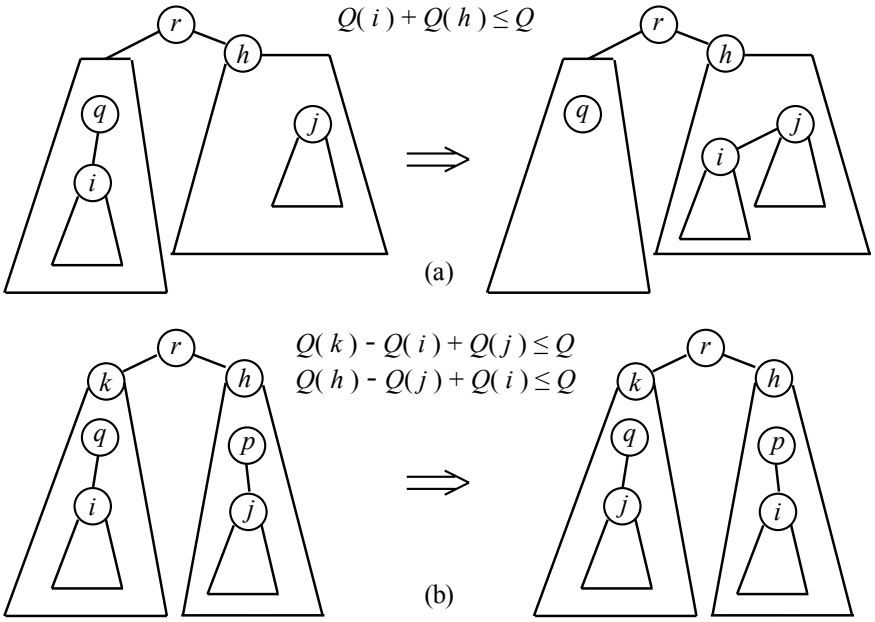


Figura 5.9: Operazioni di "Cut & Paste" e scambio per il (CMST)

È possibile implementare il calcolo della funzione σ in modo tale che il controllo di tutte le possibili mosse abbia costo $O(n)$; è necessario mantenere per ogni nodo i il valore di $Q(i)$ ed esaminare i nodi $j \neq i$ in ordine opportuno (ad esempio visitando l'albero a partire dalla radice) per disporre in $O(1)$ di $Q(h)$ e dell'informazione che j non appartiene a $T(i)$. Poichè la variazione del valore della funzione obiettivo corrispondente ad una mossa di questo tipo è $c_{ji} - c_{p[i]i}$, e quindi può essere calcolato in $O(1)$, il costo di valutare la funzione σ data da (5.8) per l'intorno che usa mosse di "Cut & Paste" è $O(n^2)$.

Esercizio 5.24. Si descriva formalmente, in pseudo-codice, una procedura che calcola la funzione σ data da (5.8) in $O(n^2)$ per la funzione intorno che usa mosse di "Cut & Paste".

Analogamente ai casi visti in precedenza, si possono realizzare mosse "più complesse" combinando opportunamente mosse "semplici", in questo caso quelle di "Cut & Paste". Ad esempio, è possibile pensare a mosse di scambio in cui si selezionano due nodi i e j tali che $i \notin T(j)$ e $j \notin T(i)$ e si scambiano i predecessori di i e j, purché ovviamente ciò non violi il vincolo sul massimo peso dei sottoalberi della radice: l'operazione è esemplificata in Figura 5.9(b). Siccome durante la mossa di scambio si tiene conto, nel valutare l'ammissibilità della soluzione ottenuta, del fatto che un insieme di nodi viene rimosso da ciascun sottoalbero della radice coinvolto mentre un altro sottoinsieme viene aggiunto, è chiaro che possono esistere ottimi locali per l'intorno basato sulle sole mosse di "Cut & Paste" che non sono ottimi locali per l'intorno che usa anche mosse di scambio, in quanto le due mosse di "Cut & Paste" corrispondenti ad una mossa di scambio potrebbero non essere eseguibili sequenzialmente per via del vincolo sul massimo peso dei sottoalberidella radice.

Esercizio 5.25. Si discuta la complessità di calcolare la funzione σ data da (5.8) per la funzione intorno che usa sia mosse di "Cut & Paste" che mosse di scambio.

È interessante notare che dopo mosse di ricerca locale è possibile effettuare un'operazione di "ottimizzazione globale" sui sottoalberi coinvolti. Infatti, il problema (CMST) sarebbe risolubile efficientemente

se si conoscessero i sottoinsiemi di nodi che formano ciascun sottoalbero della radice in una soluzione ottima: basterebbe applicare una procedura per l'(MST) a ciascun grafo parziale individuato da un sottoinsieme di nodi per determinare gli archi del sottoalbero ottimo. In generale, i sottoalberi ottenuti dopo una mossa di scambio possono non essere alberi di copertura di costo minimo per l'insieme di nodi che coprono, anche nel caso in cui lo fossero i sottoalberi di partenza, per cui applicando una procedura per determinare l'albero di copertura di costo minimo ai sottoalberi coinvolti dalla mossa si potrebbe ulteriormente migliorare la soluzione ottenuta. Questo è vero anche per il sottoalbero "destinazione" di una mossa di "Cut & Paste", ma chiaramente non per quello di "partenza". Poiché questo avrebbe un costo potenzialmente alto, una ragionevole strategia sarebbe quella di non compiere una tale ottimizzazione ad ogni iterazione dell'algoritmo di ricerca locale ma solo "occasionalmente"; ad esempio quando si sia raggiunto l'ottimo locale, in modo da provare a migliorarlo ulteriormente (il che, determinando una nuova soluzione non appartenente all'intorno, potrebbe in effetti anche permettere di riprendere la ricerca locale).

Esercizio 5.26. Si discuta come modificare le procedure note per l'(MST) in modo tale che risolvano in modo efficiente i problemi di (MST) generati da una mossa di "Cut & Paste" o di scambio.

Questo punto è rilevante perché mostra che non è necessario che le mosse coinvolgano "pochi" elementi della soluzione. Questa è stata la strategia seguita finora perché il numero di diverse configurazioni da provare per calcolare la funzione σ data da (5.8) tipicamente cresce in modo esponenziale con il numero di elementi di base coinvolti; pertanto mantenere molto limitato—$O(1)$—tale numero permette di mantenere polinomiale il numero di mosse necessario per esplorare completamente l'intorno, e quindi risolvere il problema (5.8) in tempo polinomiale. La parte critica è però la seconda: come abbiamo visto nella parte iniziale di queste dispense, esistono problemi "facili" per i quali è possibile trovare la soluzione ottima—anche all'interno di un insieme di soluzioni ammissibili di cardinalità esponenziale o peggio—in tempo polinomiale. La conoscenza di tali algoritmi permette quindi di sfruttare strutture presenti nel problema per eseguire in tempo polinomiale l'esplorazione di intorni che coinvolgono "molti" degli elementi della soluzione. Il prossimo paragrafo fornisce un ulteriore esempio di questa tecnica.

5.2.1.5 Dislocazione ottima di impianti

Il problema della dislocazione ottima di impianti definito all'Esempio 1.23 ha due tipi diversi di variabili binarie: le y_i, $i \in I$, per rappresentare la scelta relativa agli uffici da aprire, e le x_{ij}, $i \in I$, $j \in J$, per assegnare i clienti agli uffici. Per analogia con gli esempi precedenti si possono definire mosse che riguardano poche variabili: ad esempio, sono mosse possibili l'apertura o chiusura di un singolo impianto (fissare ad 1 o a 0 una data variabile y_i), lo scambio tra un impianto chiuso ed uno aperto, l'assegnazione di un cliente ad un diverso impianto e così via. In questo caso è bene però tener presente che esiste una chiara "gerarchia" tra le variabili: le y rappresentano le decisioni "principali", mentre le x rappresentano decisioni "secondarie". Ciò deriva dal fatto che, una volta fissato il valore delle y, il valore ottimo delle x può essere facilmente ottenuto risolvendo un problema di flusso di costo minimo.

Esercizio 5.27. Si descriva come istanza di un problema di (MCF) il problema di determinare, data una soluzione y, l'assegnamento ottimo dei clienti agli impianti aperti; in particolare, si discuta cosa accade nel caso in cui non siano aperti abbastanza impianti per servire tutti i clienti.

Quindi, in linea di principio le mosse potrebbero essere effettuate solamente sulle variabili y: una volta effettuata una mossa, applicando un algoritmo per (MCF) è possibile determinare il miglior valore possibile per le variabili x data la nuova scelta di y.

Un possibile svantaggio di operare in questo modo consiste nel dover risolvere un problema di (MCF) per valutare qualsiasi mossa: anchelimitandosi a mosse semplici, quali l'apertura e chiusura di un singolo impianto, ciò richiede la soluzione di n problemi di (MCF) per valutare la funzione σ ad ogni passo dell'algoritmo di ricerca locale. Sono possibili diverse strategie per cercare di ridurre il costo computazionale del calcolo di σ. Ad esempio, si potrebbero specializzare gli algoritmi noti per (MCF) alla particolare forma delle istanze da risolvere in questo caso, analogamente a quanto fatto nel §2.6 per i problemi di accoppiamento. Alternativamente, si potrebbero determinare inizialmente soluzioni

approssimate del (MCF), ossia costruire (velocemente) un assegnamento non necessariamente ottimo dei clienti agli impianti aperti. Ad esempio, nel caso di chiusura di un impianto i si potrebbero distribuire gli utenti j precedentemente assegnati a quell'impianto con un semplice criterio greedy basato sui costi, esaminandoli in ordine arbitrario ed assegnando ciascuno all'impianto aperto h con costo c_{hj} minimo tra quelli che hanno ancora capacità residua; analogamente, nel caso di apertura di un impianto i si potrebbero assegnare ad i utenti j, attualmente assegnati ad un diverso impianto h, per cui risulti $c_{ij} < c_{hj}$, partendo da quelli in cui la $c_{hj} - c_{ij}$ è maggiore. Se la nuova soluzione (x , y) così ottenuta è migliore della soluzione corrente allora sicuramente lo sarà anche quella ottenuta risolvendo il (MCF); si può quindi selezionare la mossa da compiere sulla base di questa valutazione approssimata, risolvendo il (MCF) una sola volta per iterazione. Se invece così facendo non si determina nessuna soluzione migliore di quella corrente si possono riesaminare le mosse risolvendo esattamente il (MCF), in quanto la soluzione approssimata può aver portato a non effettuare mosse che invece risultano convenienti. Infine, invece che valutazioni superiori del costo del(MCF) si potrebbero utilizzare valutazioni inferiori, sfruttando informazione duale analogamente a quanto mostrato nel §3.3.4. È in generale difficile valutare a priori quale di queste strategie possa risultare più conveniente; sono necessari a tal proposito esperimenti su un adeguato insieme di istanze "campione" che confrontino la qualità delle soluzioni ottenute e lo sforzo computazionale richiesto dall'algoritmo di ricerca locale con le diverse varianti di funzione intorno.

I due esempi precedenti mostrano come lo studio della struttura dei problemi sia fondamentale per costruire funzioni intorno opportune. In particolare, in molti casi emergono "gerarchie" tra gruppi di variabili del problema, in quanto è possibile utilizzare algoritmi efficienti per determinare il valore ottimo di un gruppo di variabili una volta fissato il valore delle altre. Questo è un caso in cui si mostra ancora una volta l'importanza della conoscenza degli algoritmi per i problemi "facili" per costruire algoritmi efficienti per problemi "difficili".

5.2.2 Intorni di grande dimensione

Il costo computazionale di un algoritmo di ricerca locale dipende fondamentalmente dalla complessità della trasformazione σ, che a sua volta dipende tipicamente dalla dimensione e dalla struttura dell'intorno $I(x)$. Quest'ultima è anche collegata alla qualità degli ottimi locali che si determinano: intorni "più grandi" tipicamente forniscono ottimi locali di migliore qualità. È necessario quindi operare un attento bilanciamento tra la complessità della funzione σ e la qualità delle soluzioni determinate; come esempio estremo, la (5.8) con $I(x) = X$ fornisce sicuramente la soluzione ottima, ma richiede di risolvere all'ottimo il problema originario, e quindi è di fatto inutile. In molti casi risulta comunque conveniente utilizzare intorni "di grande dimensione", corrispondenti a "mosse complesse". Ciò è spesso legato alla possibilità di implementare la funzione σ in modo efficiente, ad esempio utilizzando algoritmi per problemi di ottimizzazione per calcolare (5.8) senza dover esaminare esplicitamente tutte le soluzioni nell'intorno $I(x)$. Discutiamo nel seguito alcuni esempi di intorni di questo tipo.

5.2.2.1 Il problema del commesso viaggiatore

Una famiglia di funzioni intorno di grande dimensione che generalizza quella discussa nel §5.2.1.3 è quella basata sui "k-scambi": in un k-scambio si selezionano e rimuovono k archi del ciclo e si costruiscono tutti i possibili cicli Hamiltoniani che è possibile ottenere combinando i sottocammini rimanenti. Non è difficile vedere che la complessità di calcolare σ quando I è definita mediante operazioni di "k-scambio" cresce grosso modo come n^k: quindi, per valori di k superiori a 2 o 3 determinare la migliore delle soluzioni in $I(x)$ può diventare molto oneroso dal punto di vista computazionale. D'altra parte, gli ottimi locali che si trovano usando operazioni di "3-scambio" sono usualmente migliori di quelli che si trovano usando operazioni di "2-scambio", e così via.

Non essendo noti modi per calcolare (5.8) che non richiedano di esaminare sostanzialmente tutte le soluzioni nell'intorno, si possono utilizzare schemi di ricerca locale in cui la dimensione dell'intorno (il numero k di scambi) varia dinamicamente. Si può ad esempio utilizzare preferibilmente 2-scambi, passando ai 3-scambi solamente se il punto corrente si rivela essere un ottimo locale per l'intorno basato sui 2-scambi; analogamente, se il punto corrente si rivela essere un ottimo locale anche per l'intorno basato sui 3-scambi si può passare ai 4-scambi e così via, fino ad un qualche valore limite di

k fissato a priori. In questo caso risulta spesso conveniente ritornare immediatamente ai 2-scambi non appena si sia effettuata una mossa di ricerca locale con un k-scambio per $k > 2$, in quanto il k-scambio potrebbe aver generato una soluzione che non è un ottimo locale per il 2-scambio. Inoltre, quando k è "grande" può essere conveniente evitare di calcolare esattamente (5.8), terminando la computazione non appena si determina una qualunque soluzione nell'intorno che migliori "abbastanza" il valore della funzione obiettivo.

5.2.2.2 Il problema del Constrained MST

Come abbiamo già rilevato, il problema del (CMST) consiste di fatto nel determinare la partizione ottima dell'insieme dei nodi in un numero opportuno (non noto a priori) di sottoinsiemi disgiunti, in ciascuno dei quali la somma dei pesi dei nodi non sia superiore alla soglia Q; nota la partizione ottima, il problema è facilmente risolubile. Quindi, il (CMST) è un esempio di problema in cui le soluzioni ammissibili sono identificabili con una partizione di un insieme base in un certo numero di sottoinsiemi disgiunti con opportune proprietà: altri problemi con questa struttura sono ad esempio i problemi di ordinamento di lavori su macchine (ogni macchina corrisponde ad un insieme).

Un modo abbastanza generale per costruire intorni di grande dimensione per problemi con questo tipo di struttura è quello dello "scambio ciclico". Per il caso del (CMST), ciò corrisponde a selezionare un insieme di nodi i_1, i_2, ..., i_k, appartenenti ciascuno a un diverso sottoalbero della radice, ed effettuare una mossa di scambio simultanea che coinvolge tutti i nodi: porre $p[\,i_1\,] = p[\,i_2\,]$, $p[\,i_2\,] = p[\,i_3\,]$, ..., $p[\,i_{k-1}\,] = p[\,i_k\,]$ e $p[\,i_k\,] = p[\,i_1\,]$. Naturalmente, la mossa è possibile solo se la soluzione così ottenuta non viola il vincolo di capacità sui sottoalberi. È chiaro come queste mosse generalizzino quella di scambio vista al §5.2.1.4, e quindi che permettano potenzialmente di non rimanere "intrappolati" in soluzioni che siano ottimi locali per queste ultime.

Implementare un'algoritmo di ricerca locale basato su mosse di scambio ciclico richiede però di determinare in modo efficiente, ad ogni iterazione, un insieme di nodi i_1, i_2, ..., i_k a cui corrisponda una mossa di scambio ciclico ammissibile e che migliori il valore della funzione obiettivo. Può risultare utile esprimere tale problema come un problema di decisione su grafi (su un grafo diverso da quello del problema originale). Sia data infatti una soluzione ammissibile di (CMST), e si consideri il grafo orientato che ha per nodi quelli del grafo originale, tranne la radice, e nel quale esiste l'arco $(\,i\,,\,j\,)$, di costo $c_{p[\,j\,]\,i} - c_{p[\,i\,]\,i}$, se e solo se i e j appartengono a sottoalberi della radice diversi ed è possibile "innestare" i sotto il predecessore di j senza violare il vincolo di capacità, purchè nel contempo j sia "potato" (ed innestato sotto un diverso sottoalbero, non importa quale). È facile verificare che un ciclo orientato di costo negativo su questo grafo, in cui tutti i nodi appartengano a sottoalberi diversi, rappresenta una mossa di scambio ciclico che migliora il valore della funzione obiettivo, e, viceversa, qualsiasi mossa di scambio ciclico corrisponde ad un ciclo di questo tipo. Il problema di determinare una mossa di scambio ciclico può quindi essere quindi formulato nel seguente modo: dato un grafo orientato $G = (\,N\,,\,A\,)$ con costi associati agli archi, in cui l'insieme dei nodi N sia partizionato in un certo numero di sottoinsiemi disgiunti N_1, ..., N_k, determinare se esiste nel grafo un ciclo orientato di costo negativo che contenga al più un nodo per ciascuno dei sottoinsiemi N_i. Risolvere questo problema di decisione corrisponde a determinare se esiste oppure no una mossa di scambio ciclico che determina una soluzione migliore di quella corrente; la funzione (5.8) richiederebbe invece di determinare il ciclo di costo minimo tra tutti quelli aventi le caratteristiche desiderate. Come abbiamo visto nel §2.2.5, il problema di determinare se esiste un ciclo orientato di costo negativo in un grafo ha complessità polinomiale; invece, determinare il ciclo orientato di costo minimo è $\mathcal{NP}$-arduo. Il problema in esame richiede però non di determinare un qualsiasi ciclo di costo negativo, ma un ciclo di costo negativo che contenga al più un nodo per ciascuno dei sottoinsiemi N_i; come spesso accade, pur essendo una variante apparentemente minore di un problema polinomiale, questo problema è $\mathcal{NP}$-completo. È però possibile costruire euristiche per questo problema che cerchino di determinare un ciclo di costo negativo con le caratteristiche richieste, pur non garantendo di determinarlo anche nel caso in cui esista; se il ciclo viene trovato è possibile effettuare la mossa di scambio ciclico corrispondente, altrimenti l'algoritmo di ricerca locale si ferma dichiarando—forse erroneamente—che la soluzione corrente è un ottimo locale per l'intorno basato sullo scambio ciclico. Utilizzando euristiche

opportune per la determinazione del ciclo di costo negativo si possono ottenere algoritmi di ricerca locale in grado di determinare efficientemente, in pratica, soluzioni di buona qualità per il problema.

Esercizio 5.28. Si discuta come modificare l'algoritmo basato su *SPT.L* per la determinazione dei cicli orientati di costo negativo in un grafo in modo da ottenere un algoritmo euristico per determinare mosse di scambio ciclico.

Questo esempio mostra come tecniche di ottimizzazione, esatte o euristiche, possano essere utilizzate per implementare in modo efficiente la funzione σ anche per intorni di grandi dimensioni. Si noti come la conoscenza di un algoritmo esatto per risolvere un problema di decisione "facile", in questo caso il problema di determinare l'esistenza di un ciclo orientato di costo negativo in un grafo, suggerisca algoritmi euristici per un problema simile, che possono a loro volta essere utilizzati per implementare algoritmi euristici per un problema molto diverso. Questo mostra il livello di complessità e sofisticazione che è a volte necessario affrontare per realizzare algoritmi efficienti, esatti o euristici, per problemi di *OC*, e quindi l'importanza della conoscenza degli algoritmi per problemi di ottimizzazione "facili".

Esercizio 5.29. Si proponga una funzione intorno basata su mosse di scambio ciclico per il problema di ordinamento di lavori su macchine con minimizzazione del tempo di completamento (MMMS), discutendo l'implementazione della procedura σ.

5.3 Metaeuristiche

Anche utilizzando intorni di grande dimensione, quali quelli visti nel paragrafo precedente, tutti gli algoritmi di ricerca locale per problemi di *OC* si arrestano per aver determinato un ottimo locale rispetto alla funzione intorno utilizzata, o, più in generale, perchè non sono in grado di trovare "mosse" che generino soluzioni migliori della soluzione corrente. Qualora ci sia motivo per credere che la soluzione corrente non sia un ottimo globale, si pone quindi il problema di cercare di determinare un diverso, e possibilmente migliore, ottimo locale. In questo paragrafo discuteremo brevemente alcune possibili strategie che permettono di fare questo, cercando di illustrare alcune delle loro principali caratteristiche e potenziali limitazioni. Queste strategie sono chiamate *metaeuristiche* perchè non sono algoritmi specifici per un dato problema, ma metodi generali che possono essere applicati per tentare di migliorarne le prestazioni di molti diversi algoritmi di ricerca locale.

5.3.1 Multistart

In generale, la qualità dell'ottimo locale determinato da un algoritmo di ricerca locale dipende da due fattori: l'intorno utilizzato e la soluzione ammissibile iniziale da cui la ricerca parte. Nei paragrafi precedenti ci siamo soffermati principalmente sul primo fattore, assumendo che la soluzione iniziale venisse determinata con qualche euristica, ad esempio di tipo greedy. Come abbiamo visto in molti degli esempi discussi nel §5.1.1, spesso è possibile definire più di un algoritmo greedy per lo stesso problema, ed in molti casi gli algoritmi sono anche molto simili, differendo solamente per l'ordine in cui sono compiute alcune scelte, per cui non è irragionevole pensare di avere a disposizione più di un algoritmo in grado di produrre soluzioni iniziali. In questo caso, le soluzioni prodotte saranno normalmente diverse; inoltre, non è detto che l'ottimo locale determinato eseguendo l'algoritmo di ricerca locale a partire dalla migliore delle soluzioni così ottenute sia necessariamente il migliore degli ottimi locali ottenibili eseguendo l'algoritmo di ricerca locale a partire da ciascuna delle soluzioni separatamente. Tutto ciò suggerisce un'ovvia estensione dell'algoritmo di ricerca locale: generare più soluzioni iniziali, eseguire l'algoritmo di ricerca locale a partire da ciascuna di esse, quindi selezionare la migliore delle soluzioni così ottenute.

Questo procedimento è particolarmente attraente quando sia possibile, tipicamente attraverso l'uso di tecniche randomizzate, generare facilmente un insieme arbitrariamente numeroso di soluzioni iniziali potenzialmente diverse. Si pensi ad esempio agli algoritmi greedy di tipo *list scheduling* per (MMMS) presentati al §5.1.1.4: è facile costruire un'*euristica randomizzata* per il problema semplicemente costruendo un ordinamento pseudo-casuale dei lavori e poi assegnando i lavori alle macchine (selezionando ogni volta la macchina "più scarica") in quell'ordine. In questo modo è chiaramente

possibile produrre un gran numero di soluzioni iniziali diverse, a partire da ciascuna delle quali si può poi eseguire un algoritmo di ricerca locale. Poiché però è noto che alcuni ordinamenti sono in generale "più efficaci" di altri, si può operare in modo da rendere più probabile che l'ordinamento sia "del tipo promettente". Per (MMMS), ad esempio, l'ordinamento LPT normalmente produce soluzioni migliori di altri; pertanto si può pensare di costruire ordinamenti casuali in cui però la probabilità che un lavoro sia tra i primi dell'ordinamento sia in qualche modo positivamente correlata alla sua lunghezza.

Esercizio 5.30. Si discutano diversi modi per costruire ordinamenti pseudo-casuali per i lavori in (MMMS) tali che lavori con durata maggiore abbiamo maggiore probabilità di apparire prima nell'ordinamento.

Mediante l'uso di numeri pseudo-casuali è di solito facile trasformare euristiche greedy deterministiche per un problema di ottimizzazione in euristiche greedy randomizzate.

Esercizio 5.31. Si discuta se e come sia possibile trasformare ciascuno degli algoritmi greedy visti nel §5.1.1 in euristiche greedy randomizzate.

La combinazione di un algoritmo di ricerca locale e di un'euristica randomizzata viene denominato *metodo multistart*; quando, come spesso accade, l'euristica randomizzata è di tipo greedy, si parla di *GRASP* (greedy randomized adaptive search procedure). Una volta sviluppata una qualsiasi euristica randomizzata per determinare la soluzione iniziale ed un qualsiasi algoritmo di ricerca locale per un dato problema, è praticamente immediato combinare le due per costruire un metodo multistart.

Sotto opportune ipotesi tecniche è possibile dimostrare che ripetendo un numero sufficientemente alto di volte la procedura, ossia generando un numero sufficientemente alto di volte soluzioni iniziali, si è praticamente certi di determinare una soluzione ottima del problema. Si noti che, al limite, non è neppure necessaria la fase di ricerca locale: un'euristica randomizzata ripetuta più volte è comunque un modo per generare un insieme di soluzioni ammissibili, e se l'insieme è abbastanza ampio è molto probabile che contenga anche la soluzione ottima. Questo risultato non deve illudere sulla reale possibilità di giungere ad una (quasi) certezza di ottimalità utilizzando un metodo di questo tipo: in generale, il numero di ripetizioni necessarie può essere enorme, in modo tale che risulterebbe comunque più conveniente enumerare tutte le soluzioni del problema (il che è tipicamente impossibile).

Le euristiche multistart forniscono quindi un modo semplice, ma non particolarmente efficiente, per cercare di migliorare la qualità delle soluzioni determinate da un algoritmo di ricerca locale. Anche se non esistono regole che valgano per qualsiasi problema, è possibile enunciare alcune linee guida che si sono dimostrate valide per molti problemi diversi. In generale, il meccanismo della ripartenza da un punto casuale non fornisce un sostituto efficiente di una ricerca locale ben congegnata: se si confrontano le prestazioni di un metodo multistart che esegue molte ricerche locali con intorni "piccoli" e di uno che esegue poche ricerche locali con intorni "grandi", è di solito il secondo a fornire soluzioni migliori a parità di tempo totale utilizzato. Questo comportamento è spiegato dal fatto che una ripartenza da un punto pseudo-casuale "cancella la storia" dell'algoritmo: l'evoluzione successiva è completamente indipendente da tutto quello che è accaduto prima della ripartenza. In altri termini, il metodo multistart non è in grado di sfruttare in alcun modo l'informazione generata durante la ricerche locali precedenti per "guidare" la ricerca locale corrente. Ad esempio, è noto che per molti problemi le soluzioni di buona qualità sono normalmente abbastanza "vicine" le une alle altre (in termini di numero di mosse degli intorni utilizzati); quindi, l'aver determinato una "buona" soluzione fornisce una qualche forma di informazione che il metodo multistart non tiene in alcun modo in considerazione. Al contrario, la ricerca locale ha un qualche tipo di informazione sulla "storia" dell'algoritmo, in particolare data dalla soluzione corrente. La capacità di sfruttare l'informazione contenuta nelle soluzioni precedentemente generate è in effetti uno degli elementi che contraddistinguono le metaeuristiche più efficienti, quali quelle discusse nel seguito.

5.3.2 Simulated annealing

L'idea di base del *Simulated Annealing* (SA) è quella di modificare l'algoritmo di ricerca locale sostituendo i criteri deterministici di selezione del nuovo punto nell'intorno corrente e di accettazione della mossa con un criteri randomizzati. Uno schema generale di algoritmo di tipo SA (per un problema di

minimo) è il seguente:

```
procedure x = Simulated-Annealing( F , x )  {
  x* = x; c = InitTemp();
  while( c ≥ c̄ ) do {
    for i = 1 to k( c ) do
      ⟨ seleziona x' ∈ I( x ) in modo pseudocasuale ⟩;
      if( c( x' ) < c( x* ) ) then x* = x';
      if( c( x' ) < c( x ) )  then x = x';                          // downhill
                              else { r = random( 0 , 1 );
                                     if( r < e^{-(c(x')-c(x))/c} ) then x = x';   // uphill
                                     }
    ⟨ decrementa c ⟩;
    }
  x = x*;
  }
```

Procedura 5.5: Algoritmo *Simulated Annealing*

L'algoritmo prende in input una soluzione iniziale ammissibile ed inizia determinando un opportuno valore del parametro "temperatura" c, che controlla l'accettazione di mosse che peggiorano il valore della funzione obiettivo. L'algoritmo esegue quindi un certo numero di "fasi", all'interno della quali la "temperatura" è costante. Ad ogni iterazione si seleziona in modo pseudo-casuale un punto nell'intorno del punto corrente: se questo punto ha un miglior valore della funzione obiettivo (si tratta cioè di una normale mossa di ricerca locale, o "downhill") viene certamente accettato come il punto corrente. In caso contrario il punto può essere ugualmente accettato come punto corrente (mossa "uphill"), basandosi sull'estrazione di un numero pseudo-casuale: la probabilità che il punto sia accettato dipende dal peggioramento della funzione obiettivo e dal valore di c. Una fase in cui la temperatura è costante termina quando il sistema raggiunge uno "stato stazionario", ossia quando si suppone che il valore della funzione obiettivo della soluzione corrente x sia sufficientemente "vicino"—usando c come "unità di misura"—a quello della soluzione ottima; in pratica si fissa il numero di iterazioni di ogni fase secondo regole opportune. La successione dei passi compiuti dall'algoritmo quando c è costante è infatti un processo stocastico noto come *Catena di Markov*, del quale sono note molte importanti proprietà statistiche; in particolare, dopo un numero opportuno di passi il sistema tende a raggiungere uno stato stazionario indipendentemente dal punto di partenza. Alla fine di ogni fase la temperatura viene diminuita, rendendo meno probabile l'accettazione di mosse che peggiorino sensibilmente il valore della funzione obiettivo, e quindi concentrando la ricerca su soluzioni più "vicine" a quella corrente. L'algoritmo termina quando la temperatura è sufficientemente bassa (il sistema è "congelato"), riportando la migliore delle soluzioni trovate: nell'ultima fase di fatto l'algoritmo si comporta in modo simile ad un normale algoritmo di ricerca locale.

L'idea alla base dell'algoritmo è quella di permettere consistenti peggioramenti del valore della funzione obiettivo nelle fasi iniziali dell'esecuzione, in modo da evitare di rimanere intrappolati in ottimi locali molto "lontani" dall'ottimo globale. Dopo un numero sufficiente di iterazioni l'algoritmo dovrebbe aver raggiunto una parte dello spazio delle soluzioni "vicina" all'ottimo globale: a quel punto la temperatura viene diminuita per raffinare la ricerca. L'algoritmo prende spunto da un metodo usato in pratica per produrre cristalli con elevato grado di regolarità: si scalda il materiale in questione per rompere i legami chimici non desiderati (di più alta energia), si lascia ad una certa temperatura per un tempo sufficientemente lungo affinchè si creino con alta probabilità i legami chimici desiderati (di più bassa energia), quindi si raffredda il materiale per rendere più stabili i legami chimici di bassa energia, ripetendo il tutto finchè, sperabilmente, il materiale contiene solo legami di più bassa energia (si ha un cristallo regolare).

Come per il caso del multistart, anche il SA possiede, sotto opportune ipotesi tecniche, interessanti proprietà teoriche. In particolare, si può dimostrare che esiste una costante C tale che se la temperatura decresce non più rapidamente di $C/\log(k)$, dove k è il numero di iterazioni compiute, allora l'algoritmo determina una soluzione ottima con probabilità pari a uno. Come nel caso del multistart, però, queste

proprietà teoriche non hanno grande utilità in pratica: la *regola di raffreddamento* sopra indicata corrisponde ad una diminuzione molto lenta della temperatura, e quindi ad un numero di iterazioni talmente grande da rendere più conveniente l'enumerazione di tutte le soluzioni ammissibili. In pratica si usano quindi *regole di raffreddamento esponenziali* in cui la temperatura viene moltiplicata per un fattore $\theta < 1$ dopo un numero fissato di iterazioni, e la temperatura iniziale c viene scelta come la maggior differenza di costo possibile tra due soluzioni x ed x' tali che $x' \in I(x)$ (in modo tale da rendere possibile, almeno inizialmente, qualsiasi mossa). In questo modo non si ha alcuna certezza, neanche in senso stocastico, di determinare una soluzione ottima, ma si possono ottenere euristiche di buona efficienza in pratica.

Gli algoritmi di tipo SA hanno avuto un buon successo in alcuni campi di applicazioni dell'Ottimizzazione Combinatoria, ad esempio nella progettazione di circuiti VLSI. La ragione di questo successo risiede nel fatto che sono relativamente semplici da implementare, poco dipendenti dalla struttura del problema e quindi facilmente adattabili a problemi diversi, ed abbastanza "robusti", nel senso che, se lasciati in esecuzione sufficientemente a lungo, solitamente producono soluzioni di buona qualità in quasi tutti i problemi in cui sono applicati. Un vantaggio in questo senso è il fatto che contengano relativamente pochi parametri, ossia la temperatura iniziale, il valore di θ e la durata di ciascuna fase. Ogniqualvolta il comportamento di un algoritmo dipende da parametri che possono essere scelti in modo arbitrario (sia pur seguendo certe regole) si pone infatti il problema di determinare un valore dei parametri che produce soluzioni di buona qualità quanto più rapidamente possibile per le istanze che si intende risolvere. Per fare questo è normalmente necessario eseguire l'algoritmo molte volte su un nutrito insieme di istanze di test con combinazioni diverse dei parametri, in modo da ottenere informazione sull'impatto dei diversi parametri sull'efficacia ed efficienza dell'algoritmo. Questo processo può richiedere molto tempo, specialmente se l'algoritmo è poco "robusto", ossia il suo comportamento varia considerevolmente per piccole variazioni dei parametri o, con gli stessi parametri, per istanze simili. Il SA risulta piuttosto "robusto" da questo punto di vista, anche per la disponibilità di linee guida generali per l'impostazione dei parametri che si sono rivelate valide in molti diversi contesti. In linea generale, si può affermare che normalmente la tecnica del SA permette di migliorare la qualità delle soluzioni fornite da un algoritmo di ricerca locale con una ragionevole efficienza complessiva. Il metodo risulta molto spesso più efficiente dei metodi multistart basati sullo stesso intorno, poiché mantiene una maggiore quantità di informazione sulla storia dell'algoritmo—non solo la soluzione corrente, ma anche la temperatura—che in qualche modo "guida" la ricerca di una soluzione ottima. Questo tipo di tecnica risulta particolarmente adatta a problemi molto complessi, di cui sia difficile sfruttare la struttura combinatoria, e nel caso in cui l'efficacia del metodo (la qualità della soluzione determinata) sia più importante della sua efficienza (la rapidità con cui la soluzione è determinata). In casi diversi altri tipi di tecniche, come quelle discusse nel seguito, possono risultare preferibili; questo è giustificato dal fatto che l'informazione sulla storia dell'algoritmo mantenuta da un algoritmo di tipo SA è molto "aggregata", e quindi non particolarmente efficiente nel guidare la ricerca. Inoltre, il SA si basa fondamentalmente su decisioni di tipo pseudo-casuale, il che tipicamente produce un comportamento "medio" affidabile ma difficilmente è più efficiente di decisioni deterministiche guidate da criteri opportuni. Infine, la generalità del metodo, ossia il fatto che l'algoritmo sia largamente indipendente dal problema, implica che non viene sfruttata appieno la struttura del problema; la capacità di sfruttare quanto più possibile tale struttura è uno dei fattori fondamentali che caratterizzano gli algoritmi più efficienti.

5.3.3 Ricerca Taboo

Il problema fondamentale che deve essere superato quando si intende modificare un algoritmo di ricerca locale consiste nel fatto che accettare mosse che peggiorano il valore della funzione obiettivo ("uphill") pone ad immediato rischio di ritornare sulla precedente soluzione corrente, e quindi di entrare in un ciclo in cui si ripetono sempre le stesse soluzioni. Infatti, normalmente accade che $x' \in I(x)$ implica che $x \in I(x')$; di conseguenza, qualora si abbia $c(x') > c(x)$ (in un problema di minimo) e si accetti ugualmente di porre x' come punto corrente, all'iterazione successiva si avrà la soluzione x nell'intorno del punto corrente con un valore minore della funzione obiettivo, e quindi sarà possibile (e

probabile) effettuare una mossa "downhill" ritornando su x. Le tecniche di tipo SA non sono immuni da questo problema, ma sono basate su decisioni stocastiche, per cui la ripetizione del punto corrente non causa necessariamente l'entrata in un ciclo. Infatti, nella situazione precedente il nuovo punto corrente viene scelto in modo casuale nell'intorno di x', per cui anche se $c(x)$ fosse l'unico punto in $I(x')$ con valore migliore della funzione obiettivo non sarebbe necessariamente scelto; anche qualora ciò accadesse, comunque, il nuovo punto x'' scelto nell'intorno di x sarebbe con alta probabilità diverso da x', e quindi la probabilità di ritornare un numero molto alto di volte sullo stesso punto corrente x è estremamente bassa.

Qualora si vogliano implementare algoritmi deterministici questo tipo di considerazione non è più valida, ed è quindi necessario porre in effetto tecniche che impediscano, o comunque rendano altamente improbabile, l'entrata in un ciclo. La più diffusa da queste tecniche è quella delle *mosse taboo*, che caratterizza un'ampia famiglia di algoritmi detti di *ricerca taboo* (TS, da Taboo Search). Questi algoritmi sono, in prima approssimazione, normali algoritmi di ricerca locale, in cui cioè si compiono mosse "downhill" fin quando ciò sia possibile; quando però il punto corrente è un ottimo locale per l'intorno utilizzato, e quindi un normale algoritmo di ricerca locale terminerebbe, un algoritmo TS seleziona comunque una soluzione $x' \in I(x)$ secondo criteri opportuni e compie una mossa "uphill". Per evitare i cicli, l'algoritmo mantiene una *lista taboo* che contiene una descrizione delle mosse "uphill"; ad ogni iterazione, nel cercare una soluzione $x' \in I(x)$ con $c(x') < c(x)$, l'algoritmo controlla la lista taboo, e scarta tutte le soluzioni che sono generate da una mossa taboo. Siccome l'algoritmo permette mosse "uphill", non è garantito che la soluzione corrente al momento in cui l'algoritmo termina sia la migliore determinata; come nel caso del SA, si mantiene quindi, oltre alla soluzione corrente, la miglior soluzione x^* tra tutte quelle determinate. In prima approssimazione si potrebbe pensare che la lista taboo contenga tutte le soluzioni precedentemente trovate con valore della funzione obiettivo migliore di quello della soluzione corrente; questo chiaramente eviterebbe i cicli, ma normalmente risulta eccessivamente costoso, sia in termini di memoria richiesta che in termini del costo di verificare che una data soluzione appartenga alla lista. In generale si preferisce quindi mantenere nella lista taboo una descrizione, anche parziale delle mosse (che definiscono la funzione intorno) che hanno generato la soluzione corrispondente. Questa descrizione dipende quindi dalla particolare funzione intorno utilizzata, e deve essere "compatibile" con la funzione σ, ossia non deve rendere "eccessivamente più costosa" la determinazione della nuova soluzione.

Si consideri ad esempio il problema del (TSP) e la funzione intorno basata su mosse di 2-scambio descritta nel §5.2.1.3, ossia sullo scambio della coppia di lati $\{\{i, j\}, \{h, k\}\}$ con $\{\{i, k\}, \{h, j\}\}$. Una possibile implementazione della lista taboo per questo caso potrebbe contenere semplicemente la coppia $\{\{i, k\}, \{h, j\}\}$ che caratterizza la mossa: una mossa di 2-scambio sarebbe dichiarata taboo—e quindi scartata—se coinvolgesse esattamente quei due lati. Si noti che questa scelta può impedire di generare molte soluzioni oltre a quella che originariamente ha causato l'inserzione di una data coppia di lati nella lista taboo. Infatti, se x ed x' sono rispettivamente il punto corrente ed il nuovo punto corrispondenti ad una mossa "uphill", ed $x'' \neq x$, $x'' \in I(x')$ è il punto accettato all'iterazione successiva, allora effettuare la mossa taboo (qualora possibile) non genererebbe x, e quindi non causerebbe necessariamente l'entrata in un ciclo. Quindi, questa implementazione della lista taboo è in qualche modo "eccessiva" rispetto al compito di evitare i cicli; ciononostante può risultare opportuna perchè può essere facilmente ed efficientemente integrata nel calcolo della funzione σ.

In effetti, per semplificare il controllo della lista taboo non è infrequente che si implementi una lista taboo che contiene una *descrizione parziale* delle mosse utilizzate. Si consideri ad esempio il problema (MMMS) e la funzione intorno basata su mosse di "spostamento" e "scambio" descritta nel §5.2.1.2. Una possibile implementazione della lista taboo per questo caso potrebbe contenere semplicemente coppie $\{d, i\}$ dove d è una durata di lavoro ed i è una macchina: è considerata una mossa taboo assegnare alla macchina i un lavoro di durata d. Questa è ovviamente una descrizione parziale sia di una mossa di spostamento che di scambio: se si pone $\{d, i\}$ nella lista taboo ogniqualvolta si elimina un lavoro di durata d dalla lista di quelli assegnati alla macchina i, si evitano sicuramente i cicli. Analogamente a quello che abbiamo visto per il (TSP), questa implementazione impedisce di generare molte altre soluzioni oltre a quelle già visitate dall'algoritmo.

Inserire mosse nella lista taboo diminuisce il numero di soluzioni considerate ad ogni passo come possibili nuove soluzioni correnti: questo può seriamente limitare la capacità dell'algoritmo di scoprire soluzioni migliori. Per questo sono necessari dei meccanismi che limitino l'effetto della lista taboo. Il primo meccanismo, usato in tutti gli algoritmi di TS, consiste semplicemente nel limitare la massima lunghezza della lista taboo ad una costante fissata: quando la lista ha raggiunto tale lunghezza ed una nuova mossa deve essere resa taboo, la più "vecchia" delle mosse nella lista viene nuovamente resa ammissibile. In generale non è facile determinare valori della massima lunghezza della lista delle mosse taboo che garantiscano che l'algoritmo non entri in un ciclo; in pratica, però, gli algoritmi di tipo TS non entrano in ciclo se la lista taboo contiene anche solo poche decine di mosse. Ciò dipende dal fatto che il numero di possibili soluzioni che possono essere visitate con k mosse di ricerca locale aumenta esponenzialmente in k, e quindi è molto poco probabile che l'algoritmo visiti esattamente, tra tutte quelle possibili, una delle soluzioni già generate. Limitare la lunghezza della lista taboo ha anche, chiaramente, un effetto positivo sulle prestazioni dell'algoritmo: normalmente, il costo di verificare che una mossa non sia taboo cresce con la lunghezza della lista, e quindi ciò potrebbe risultare molto costoso qualora la liste divenisse molto lunga.

Un secondo metodo per limitare gli effetti negativi della lista taboo è quello di inserire un cosiddetto *criterio di aspirazione*, ossia un insieme di condizioni logiche tale per cui se la soluzione $x' \in I(x)$ generata da una mossa soddisfa almeno una di queste condizioni, allora x' può essere considerata tra i possibili candidati a divenire la prossima soluzione corrente anche se la mossa appartiene alla lista taboo. Ovviamente, il criterio di aspirazione deve essere scelto in modo tale da garantire—o rendere molto probabile—che l'algoritmo non entri in un ciclo: un possibile criterio di aspirazione è ad esempio $c(x') < c(x^*)$, dove x^* è la migliore delle soluzioni generate, dato che in questo caso x' non può sicuramente essere stata visitata in precedenza. Sono comunque possibili anche altri criteri di aspirazione: come il nome suggerisce, e l'esempio precedente mostra, i criteri di aspirazione corrispondono spesso a condizioni che richiedono che x' sia sotto qualche aspetto una "buona" soluzione. Ad esempio, nel problema (MMMS) si potrebbe permettere di assegnare un lavoro di durata d ad una data macchina i anche se $\{d, i\}$ fa parte della lista taboo purchè il tempo di completamento della macchina i dopo l'assegnamento sia minore del tempo di completamento che la macchina aveva al momento in cui $\{d, i\}$ è stata inserita nella lista delle mosse taboo (questo richiederebbe chiaramente di memorizzare anche tale durata nella lista). Chiaramente questo evita i cicli in quanto la soluzione ottenuta dalla mossa è "localmente", ossia limitatamente alla macchina i, migliore di quella che c'era al momento in cui la mossa taboo è stata effettuata.

Oltre a queste componenti base, gli algoritmi TS spesso contengono altre strategie che risultano utili per rendere il processo di ricerca più efficiente. Ad esempio, per quei problemi in cui le "buone" soluzioni sono spesso "vicine" sono spesso utilizzate tecniche di *intensificazione*, che concentrano temporaneamente la ricerca "vicino" alla soluzione corrente. Questo può essere ottenuto ad esempio restringendo la dimensione dell'intorno, oppure modificando la funzione obiettivo con un termine che penalizza leggermente le soluzioni dell'intorno più "lontane" da quella corrente. L'intensificazione viene solitamente effettuata quando si determina una nuova miglior soluzione x^*, e mantenuta per un numero limitato di iterazioni, nella speranza che vicino alla migliore soluzione determinata fino a quel momento ci siano altre soluzioni ancora migliori. Una strategia in un certo senso opposta è quella della *diversificazione*, che cerca di indirizzare la ricerca verso aree dello spazio delle soluzioni diverse da quelle esplorate fino a quel momento. Modi possibili per implementare strategie di diversificazione sono ad esempio penalizzare leggermente le soluzioni dell'intorno "vicine" alla soluzione corrente x, allargare la dimensione dell'intorno, effettuare in una sola iterazione combinazioni di molte mosse, ad esempio selezionandole in modo randomizzate, per "saltare" in una zona dello spazio delle soluzioni "lontana" da x, o persino effettuare una ripartenza pseudo-casuale come nel metodo multistart. Normalmente, vengono effettuate mosse di diversificazione quando per "molte" iterazioni successive la migliore soluzione trovata x^* non cambia, suggerendo che l'area dello spazio delle soluzioni in cui si trova la soluzione corrente sia già stata sostanzialmente esplorata, e che sia quindi necessario visitarne una diversa. Nell'implementare tecniche di intensificazione e diversificazione risulta spesso utile sfruttare l'informazione contenuta nella sequenza di soluzioni generate, ad esempio associando a ciascun

costituente elementare delle soluzioni un qualche valore numerico che ne rappresenti in qualche modo "l'importanza". Si pensi ad esempio di applicare un qualche algoritmo di ricerca locale per risolvere il (TSP), e di mantenere per ogni lato un contatore che indichi di quante soluzioni correnti il lato ha fatto parte fino a quel momento: lati con un valore alto del contatore sono in qualche senso "più rilevanti" di lati con un valore basso del contatore. In una procedura di diversificazione potrebbe essere ragionevole concentrarsi su lati con valore alto del contatore, in modo da esplorare soluzioni "diverse" da quelle visitate fino a quel momento. Alternativamente si potrebbe contare per ogni arco di quante delle migliori k soluzioni generate ha fatto parte, dove k è un valore fissato. Più in generale, si potrebbe mantenere per ogni lato un valore numerico calcolato a partire dal costo delle soluzioni di cui ha fatto parte, in modo tale che il valore sia "alto" per lati cha fanno "spesso" parte di soluzioni "buone". Con un'opportuna scelta della funzione e dei parametri, questo tipo di informazione sulle soluzioni generate dall'algoritmo può rivelarsi utile per guidare la ricerca.

Infine, una strategia che talvolta si rivela utile è quella di permettere che la soluzione corrente x diventi inammissibile per un numero limitato di iterazioni, se questo serve a migliorare sensibilmente il valore della soluzione obiettivo. Questo viene fatto per i problemi in cui sia particolarmente difficile costruire soluzioni ammissibili: le "poche" soluzioni ammissibili del problema possono quindi essere "lontane", ed è necessario permettere mosse in cui si perde l'ammissibilità, seguite da una successione di mosse in cui si cerca di riguadagnarla.

È chiaro come, rispetto ad un algoritmo di tipo SA, un algoritmo TS sia più complesso da realizzare. L'implementazione della lista taboo, e spesso anche dei criteri di aspirazione, è fortemente dipendente dalla scelta della funzione intorno, e quindi l'algoritmo deve essere progettato tenendo in conto di tutti questi aspetti contemporaneamente. Normalmente, anche le strategie di intensificazione e diversificazione dipendono in qualche modo dall'intorno utilizzato, per cui anche questi aspetti devono essere considerati insieme agli altri. Per tutte le strategie sopra elencate è normalmente necessario determinare sperimentalmente diversi parametri, quali la lunghezza della lista taboo, il numero di iterazioni per l'intensificazione, il numero di iterazioni per la diversificazione, e così via: il numero complessivo di parametri può essere alto, il che può rendere difficoltosa la determinazione di "buoni" valori per tutti i parametri. La grande flessibilità data dai molti parametri, ed il fatto che ogni aspetto di un algoritmo TS debba essere adattato allo specifico problema di OC risolto, sono però anche le caratteristiche che possono rendere questi algoritmi molto efficienti in pratica: un algoritmo TS ben congegnato ed in cui i parametri siano scelti in modo opportuno si rivela spesso più efficiente, ad esempio, di un algoritmo di tipo SA per lo stesso problema che usi la stessa funzione intorno, nel senso che produce soluzioni di migliore qualità a parità di tempo. Qualora l'efficienza sia almeno altrettanto importante della qualità delle soluzioni fornite, quindi, può essere ragionevole investire le risorse necessarie a realizzare un algoritmo TS.

5.3.4 Algoritmi genetici

Uno dei principali fattori che determina l'efficienza di un algoritmo di ricerca locale è la capacità di sfruttare l'informazione contenuta nella sequenza di soluzioni generate per guidare la ricerca di soluzioni migliori. Come abbiamo visto, le diverse strategie algoritmiche ottengono questo risultato in gradi diversi, e con diversi accorgimenti. Esiste una classe di algoritmi euristici, che in linea di principio *non* sono algoritmi di ricerca locale, che portano questo concetto alle estreme conseguenze: si tratta degli algoritmi *population based*, i quali non mantengono una sola soluzione corrente, come visto negli esempi precedenti, ma un insieme di soluzioni diverse. Il primo e più famoso esempio di questa classe sono gli *algoritmi genetici*, che cercano di riprodurre alcuni dei meccanismi che si ritiene stiano alla base dell'evoluzione delle forme di vita.

L'algoritmo mantiene una *popolazione* di soluzioni e procede per fasi, corrispondenti a "generazioni" nella popolazione. In ogni fase vengono ripetute un numero opportuno di volte le seguenti operazioni:

- all'interno della popolazione vengono selezionate in modo pseudo-casuale due (o più) soluzioni "genitrici";

- a partire da tali soluzioni vengono generate in modo pseudo-casuale un certo numero di soluzioni "discendenti", ottenute "mescolando" le caratteristiche di tutte le soluzioni genitrici;
- a ciascuna soluzione così generata vengono applicate alcune "mutazioni casuali" che cercano di introdurre nella popolazione caratteristiche altrimenti non presenti.

Alla fine di ogni fase si procede alla *selezione*: a ciascuna delle soluzioni, sia quelle presenti nella popolazione all'inizio della fase che quelle generate durante la fase, viene associato un *valore di adattamento* (fitness), ad esempio il valore della funzione obiettivo, e vengono mantenute nella popolazione (sopravvivono alla generazione successiva) solo le soluzioni con miglior fitness, mantenendo costante la dimensione della popolazione. Alternativamente, le soluzioni sopravvissute sono selezionate in modo pseudo-casuale con probabilità dipendente dal fitness. Sono state poi proposte molte varianti a questo schema base, ad esempio suddividendo la popolazione in sotto-popolazioni più piccole con scambi limitati (in modo da simulare le barriere geografiche presenti in natura) o imponendo la "morte" delle soluzioni dopo un numero massimo di generazioni indipendentemente dal loro valore di fitness.

Tutte le operazioni di un algoritmo genetico devono essere specializzate per il problema da risolvere, anche se per alcune esistono implementazioni abbastanza standard. Ad esempio, la selezione delle soluzioni genitrici viene normalmente effettuata semplicemente estraendo in modo casuale dalla popolazione; la probabililità di essere estratta può essere uniforme oppure dipendere dal valore di fitness. Analogamente, la selezione alla fine di ogni fase è normalmente effettuata, in modo deterministico o pseudo-casuale, semplicemente privilegiando le soluzioni con miglior fitness. Per quanto riguarda la definizione del valore di fitness, spesso si usa semplicemente il valore della funzione obiettivo. In alcuni casi può risultare opportuno modificare la funzione obiettivo per premiare soluzioni che, a parità di funzione obiettivo, appaiano più desiderabili: ad esempio, per il problema (MMMS) si può assegnare un valore di fitness leggermente più alto, a parità di makespan, a quelle soluzioni in cui le macchine che non contribuiscono a determinare il makespan sono "bilanciate", ossia hanno tutte tempo di completamento simile. Per i problemi in cui sia particolarmente difficile costruire soluzioni ammissibili è possibile ammettere nella popolazione soluzioni non ammissibili, penalizzando opportunamente la non ammissibilità nel corrispondente valore di fitness. La generazione di soluzioni e le mutazioni casuali, invece, sono strettamente dipendenti dal problema. Questo non è stato inizialmente ben compreso: dato che qualsiasi soluzione ammissibile rappresentata in un computer può essere vista come una stringa di bits, si è sostenuto che le operazioni di generazione e mutazione potessero essere implementate in modo generico, ad esempio rimpiazzando nel "patrimonio genetico" di un genitore una o più sottostringhe di bits con quelle prelevate dalle corrispondenti locazioni nel "patrimonio genetico" dell'altro genitore, e cambiando il valore di un piccolo numero dei bits scelti in modo pseudo-casuale. Questo risulta in effetti possibile per problemi con "pochi vincoli", in cui sia molto facile costruire una soluzione ammissibile. Si pensi ad esempio a (MMMS): una soluzione ammissibile è data da un qualsiasi assegnamento dei lavori alle macchine, ossia da qualsiasi vettore $s[\cdot]$ di n componenti in cui ogni elemento $s[\,i\,]$ appartenga all'insieme $\{\,1\,,\,\ldots\,,\,m\,\}$, col significato che $s[\,i\,] = h$ se e solo se il lavoro i è assegnato alla macchina h. Dati due vettori $s_1[\cdot]$ ed $s_2[\cdot]$ di questo tipo, è molto facile produrre un nuovo vettore $s_3[\cdot]$ possibilmente diverso da entrambe: ad esempio si possono selezionare in modo pseudo-casuale due numeri $1 \leq i \leq j \leq n$ e porre $s_3[\,h\,] = s_1[\,h\,]$ per $i \leq h \leq j$ e $s_3[\,h\,] = s_2[\,h\,]$ per tutti gli altri indici h. Le mutazioni possono poi essere ottenute semplicemente cambiando in modo pseudo-casuale il valore di un piccolo numero di elementi di $s_3[\cdot]$, scelti in modo pseudo-casuale. Nella maggioranza dei casi questo però non risulta effettivamente possibile: per la maggior parte dei problemi, operando in modo analogo a quanto appena visto si ottengono quasi sempre soluzioni non ammissibili. È quindi necessario sviluppare euristiche specifiche che costruiscano soluzioni ammissibili cercando di replicare, per quanto possibile, le caratteristiche di entrambe i genitori.

Si consideri ad esempio il problema del (TSP): dati due cicli Hamiltoniani diversi, si devono produrre uno o più cicli che "combinino" le caratteristiche dei due genitori. Un possibile modo di procedere è quello di considerare fissati, nei cicli "figli", tutti gli archi che sono comuni ad entrambe i genitori, e poi cercare di completare questa soluzione parziale fino a formare un ciclo Hamiltoniano mediante un'euristica costruttiva randomizzata analoga alla "Nearest Neighbour" descritta nel §5.1.1.3. Definito

un nodo "corrente" iniziale i, l'euristica prosegue sicuramente sul nodo j se il lato $\{ i , j \}$ appartiene ad entrambe i genitori. Se invece il "successore" di i nei due cicli Hamiltoniani è diverso, l'euristica seleziona uno dei due successori che non sia ancora stato visitato secondo un qualche criterio opportuno; ad esempio, quello a cui corrisponde il lato di costo minore, oppure in modo pseudo casuale con probabilità uniforme, oppure con probabilità dipendente dal costo del lato, oppure ancora con probabilità dipendente dal valore della funzione obiettivo del "genitore" a cui il lato corrispondente appartiene (favorendo le scelte del "genitore migliore"). Qualora entrambe i successori siano stati già visitati l'euristica selezionerà un diverso nodo secondo altri criteri (ad esempio quello "più vicino); se tutti i nodi raggiungibili da i sono già visitati (e non si è formato un ciclo) l'euristica fallirà, "abortendo" la generazione della nuova soluzione.

Una procedura in qualche modo analoga si può utilizzare nel caso di (CMST). Le due soluzioni "genitrici" sono alberi di copertura diversi dello stesso grafo: esse avranno quindi un certo numero di sottoalberi comuni, che risulterebbero quindi sicuramente ammissibili se fossero utilizzate come sottoalberi della radice (al limite i sottoalberi potranno essere formati da un solo nodo). In ciascun sottoalbero, il predecessore di tutti i nodi tranne la radice è identico nelle due soluzioni, mentre la radice ha due predecessori diversi, uno per ciascuna soluzione. Si può quindi realizzare una procedura euristica che tenti di combinare questi sottoalberi per formare una soluzione ammissibile: selezionata la radice i di un sottoalbero, l'euristica seleziona uno dei due predecessori secondo un qualche criterio opportuno, ad esempio quello a cui corrisponde l'arco di costo minore, oppure in modo pseudo-casuale con probabilità uniforme, oppure con probabilità dipendente dal costo dell'arco, oppure ancora con probabilità dipendente dal valore della funzione obiettivo del "genitore" a cui l'arco corrispondente appartiene. Naturalmente un predecessore può essere selezionato solamente se ciò non crea un sottoalbero di peso superiore alla soglia massima fissata: se ciò accade per entrambe i predecessori corrispondenti alle soluzioni genitrici l'euristica tenterà di selezionare un diverso predecessore—al limite, la radice dell'albero—mediante criteri opportuni. Se nessun predecessore può essere selezionato senza violare i vincoli l'euristica fallirà, "abortendo" la generazione della nuova soluzione.

Si noti che euristiche simili possono essere utilizzate anche per problemi in cui sia "facile" produrre soluzioni ammissibili. Nel caso di (MMMS), ad esempio, si potrebbe utilizzare un approccio simile alle euristiche list scheduling viste al §5.1.1.4: ordinati i lavori secondo un qualche criterio, si potrebbe assegnare ciascun lavoro h ad una delle due macchine $s_1[\, h \,]$ ed $s_2[\, h \,]$, scegliendo quella che ha il tempo di completamento minimo oppure con un qualche criterio pseudo-casuale analogo a quelli visti precedentemente (i lavori che siano assegnati alla stessa macchina in entrambe le soluzioni genitrici sarebbero quindi sicuramente assegnati a quella macchina in tutte le soluzioni discendenti).

Le mutazioni casuali sono normalmente più facili da realizzare, soprattutto qualora si disponga già di un algoritmo di ricerca locale per il problema. Infatti, un modo tipico per realizzare mutazioni è quello di effettuare un piccolo numero di mosse scelte in modo pseudo-casuale, a partire dalla soluzione generata. Ad esempio, per il (TSP) si potrebbero effettuare un piccolo numero di 2-scambi tra coppie $\{ i , j \}$ e $\{ h , k \}$ di lati del ciclo selezionate in modo pseudo-casuale; analogamente, per (CMST) si potrebbero effettuare un piccolo numero di mosse di "Cut & Paste" tra coppie di nodi i e j scelte in modo pseudo-casuale. Si noti che nel caso del (CMST) le mosse di "Cut & Paste" potrebbero produrre una soluzione non ammissibile: in questo caso si ha la scelta tra non effettuare la mossa oppure "abortire" la soluzione così generata e passare a generarne un'altra.

Infine, qualsiasi algoritmo genetico richiede una prima fase in cui viene generata la popolazione iniziale. Ciò può essere ottenuto in vari modi: ad esempio, si può utilizzare un'euristica randomizzata analogamente a quanto visto per il metodo multistart. Alternativamente si può utilizzare un algoritmo di ricerca locale, eventualmente con multistart, tenendo traccia di un opportuno sottoinsieme delle soluzioni visitate, che costituiranno a terminazione la popolazione iniziale per l'algoritmo genetico.

Anche gli algoritmi genetici, come quelli di TS, dipendono da un numero di parametri il cui valore può non essere facile da fissare. Un parametro molto importante è ad esempio la cardinalità della popolazione: popolazioni troppo piccole non permettono di diversificare a sufficienza la ricerca, che può rimanere "intrappolata" in una zona dello spazio delle soluzioni, mentre una popolazione troppo

numerosa può rendere poco efficiente l'algoritmo. Altri parametri importanti riguardano il numero di nuove soluzioni costruite ad ogni generazione, i parametri per la scelta dei genitori e la selezione e così via.

Gli algoritmi genetici "puri", ossia che seguono lo schema precedente, possono rivelarsi efficienti ed efficaci; spesso, però algoritmi basati sulla ricerca locale risultano maggiormente efficienti. In effetti, gli algoritmi genetici di maggior successo sono normalmente quelli che integrano entrambe le tecniche: ad ogni soluzione generata, dopo la fase di mutazione, viene applicata una procedura di ricerca locale per tentare di migliorarne il valore di fitness. Un algoritmo di questo tipo può anche essere considerato un metodo "multistart con memoria", in cui cioè le soluzioni di partenza del metodo multistart non sono selezionate in modo completamente pseudo-casuale, ma si cerca di utilizzare l'informazione precedentemente generata. La combinazione degli algoritmi genetici e degli algoritmi di ricerca locale può assumere molte forme diverse. I casi estremi corrispondono all'algoritmo genetico "puro", in cui cioè non si effettua nessuna ricerca locale, ed all'algoritmo di ricerca locale "puro", in cui cioè la popolazione è formata da un solo individuo. I metodi ibridi si differenziano fondamentalmente per la quantità di risorse che vengono spese nell'uno e nell'altro tipo di operazione: un algoritmo in cui la ricerca locale sia molto semplice, ed eventualmente eseguita comunque per al più un numero fissato di mosse, avrà un comportamento più simile a quello di un algoritmo genetico "puro", mentre un algoritmo in cui la ricerca locale sia effettuata in modo estensivo—-ad esempio con tecniche TS e con intorni di grande dimensione—avrà un comportamento più simile a quello di un algoritmo di ricerca locale "puro" con multistart. Esiste inoltre la possibilità di eseguire sequenzialmente un algoritmo genetico ed uno di ricerca locale, eventualmente più volte: l'algoritmo di ricerca locale fornisce la popolazione iniziale per quello genetico e questi fornisce una nuova soluzione iniziale per la ricerca locale una volta che questa si sia fermata in un ottimo locale. Dato che esistono moltissimi modi per combinare i due approcci, è molto difficile fornire linee guida generali; ciò è particolarmente vero in quando in un approccio ibrido è necessario determinare il valore di molti parametri (per entrambe gli algoritmi e per la loro combinazione), rendendo la fase di messa a punto dell'algoritmo potenzialmente lunga e complessa. Solo l'esperienza può quindi fornire indicazioni sulla migliore strategia da utilizzare per un determinato problema di OC. Si può però affermare che un'opportuna combinazione delle tecniche descritte in questo Capitolo può permettere di determinare efficientemente soluzioni ammissibili di buona qualità per moltissimi problemi di OC, per quanto al costo di un consistente investimento nello sviluppo e nella messa a punto di approcci potenzialmente assai complessi.

Riferimenti Bibliografici

E. Aarts and J.K. Lenstra (eds.) "Local Search in Combinatorial Optimization" Wiley, 1997

F. Maffioli "Elementi di Programmazione Matematica" Casa Editrice Ambrosiana, 2000

V. Varzirani "Approximation Algorithms" Springer-Verlag, 2001

Capitolo 6

Tecniche di rilassamento

Come abbiamo osservato nei Capitoli 4 e 5, uno dei passi fondamentali nel processo di risoluzione di un problema di *OC* consiste nell'individuazione di valutazioni superiori (se il problema è di massimo, inferiori se è di minimo) del suo valore ottimo. Ciò permette di certificare l'ottimalità, o almeno stimare la qualità, delle soluzioni ammissibili di cui si disponga, ad esempio avendole costruite attraverso euristiche come quelle descritte nel Capitolo 5. Se la qualità della soluzione disponibile non risulta soddisfacente, le valutazioni superiori sono uno degli elementi fondamentali per costruire algoritmi in grado di determinare, in linea di principio, soluzioni ottime o comunque con qualsiasi grado di accuratezza desiderato, come vedremo nel Capitolo 7. In effetti, come già discusso nel §4.3, si può ritenere che la difficoltà di risolvere un problema di *OC* consista proprio nel calcolare (o stimare in modo sufficientemente accurato) il valore ottimo della funzione obiettivo del problema.

La tecnica più comunemente utilizzata per produrre valutazioni superiori per un problema di *OC* è risolvere un *rilassamento* del problema. Il concetto di rilassamento è già stato introdotto nel Capitolo 1 e discusso nei Capitoli 4 e 5. In questo capitolo descriveremo alcune delle principali tecniche utilizzate per definire rilassamenti per problemi di *OC*, sfruttando spesso le loro formulazioni in termini di *PLI*, ed in particolare:

- il rilassamento continuo;
- il rilassamento per eliminazione di vincoli (o combinatorio);
- il rilassamento Lagrangiano;
- il rilassamento surrogato.

Esistono altri modi generali per costruire rilassamenti di problemi di *OC*, quali i rilassamenti semidefiniti positivi, e per molti problemi specifici possono essere sviluppati rilassamenti ad-hoc. Le tecniche che presenteremo sono comunque tra le più generali ed utilizzate, oltre ad essere relativamente facili da comprendere. Alla descrizione di ciascuna di queste tecniche, con le relative esemplificazioni per problemi specifici, premettiamo alcune semplici considerazioni generali sull'uso dei rilassamenti, nella pratica, per la soluzione di problemi di *OC*.

Nella costruzione di un rilassamento per un problema "difficile" di *OC* occorre sempre tenere conto del bilanciamento tra due obiettivi necessariamente contrastanti: l'*efficacia* del rilassamento, ossia la qualità della valutazione superiore da esso prodotta, e l'*efficienza* del rilassamento, ossia il tempo necessario a determinare tale valutazione per l'istanza in esame. L'efficacia di un rilassamento (P') di un dato problema (P) si misura solitamente attraverso il corrispondente *gap assoluto o relativo*, ossia (assumendo $z(P) > 0$)

$$G_A = z(P') - z(P) \qquad\qquad G_R = \frac{z(P') - z(P)}{z(P)} \ ;$$

un rilassamento sarà tanto più efficace quanto più il gap corrispondente è piccolo. È intuitivo come questi due obiettivi siano fondamentalmente in contrasto tra loro; un'esemplificazione estrema si può ottenere considerando i due rilassamenti "estremi" consistenti nel riportare sempre la valutazione

"$+\infty$" e nel risolvere il problema all'ottimo, rispettivamente. È chiaro come il primo sia un rilassamento valido a costo nullo (costante), che tuttavia non produce di fatto nessuna informazione utile sull'istanza in oggetto; per contro, il secondo produce informazione sommamente utile, nel contesto della soluzione del problema, ma ad un costo che potenzialmente può non essere ragionevole pagare. In generale, per un dato problema di ottimizzazione si possono costruire molti rilassamenti diversi, che avranno quindi efficienza ed efficacia diverse; si pone quindi il problema di confrontarli tra loro. Dati due rilassamenti (P') e (P'') di uno stesso problema (P), non è in generale ragionevole esprimere preferenze tra (P') e (P'') tranne nel caso in cui—ad esempio—(P') *domini* (P''), ossia per ogni istanza del problema (o per un sottoinsieme di istanze rilevanti) (P') determini una valutazione non superiore (e quindi un gap non peggiore) di quella determinata da (P'') in un tempo non superiore. In tutti gli altri casi, ossia quando uno dei due rilassamenti fornisca gap migliori in un tempo superiore, almeno per insiemi rilevanti di istanze, la scelta tra (P') e (P'') dipenderà dalla specifica applicazione, oltreché possibilmente dall'istanza a cui il procedimento si applica. Ad esempio, se la valutazione viene computata una sola volta per stimare la qualità di una soluzione ottenuta con l'applicazione di un'euristica particolarmente costosa (ad esempio una metaeuristica), e se da tale stima dipendono decisioni potenzialmente critiche (ad esempio far proseguire ancora la metaeuristica, pagando un ulteriore sostanziale costo), allora presumibilmente l'efficacia sarà maggiormente rilevante dell'efficienza. Se invece la valutazione deve essere computata molte volte, come accade all'interno degli algoritmi enumerativi descritti nel Capitolo 7, l'efficienza può assumere un'importanza preponderante.

Nella pratica, comunque, si ritiene più spesso che l'efficacia di un rilassamento abbia importanza superiore all'efficienza; capita cioè sovente che rilassamenti più "costosi", ma in grado di fornire informazione molto accurata sul problema, risultino più utili di rilassamenti meno costosi ma meno accurati, anche nell'ambito di approcci enumerativi; ciò sarà discusso più in dettaglio nel Capitolo 7.

Per terminare queste brevi riflessioni generali aggiungiamo che il ruolo del rilassamento è quello di estrarre dall'istanza del problema da risolvere "informazione" che possa poi essere utilizzata all'interno del processo risolutivo. Questa informazione comprende sicuramente, e fondamentalmente, la valutazione superiore, ma può essere anche più ricca. Si consideri ad esempio il problema dello zaino e la valutazione superiore ottenuta risolvendone il rilassamento continuo come mostrato nel §5.1.2.1; oltre alla valutazione superiore tale rilassamento restituisce anche una soluzione primale frazionaria. Questa soluzione è un'ulteriore fonte di informazione: ad esempio, come abbiamo visto, mediante una semplice tecnica di arrotondamento da essa si deriva una soluzione ammissibile per il problema. L'utilità di tale informazione è particolarmente evidente nel caso in cui la soluzione del rilassamento continuo sia intera: il rilassamento ha di fatto risolto il problema. Ulteriori esempi dell'utilità di informazione prodotta da rilassamenti saranno discussi nel seguito.

6.1 Rilassamento continuo

Il rilassamento continuo è già stato introdotto e discusso nel §4.2.1. Data una formulazione *PLI* di un problema di *OC*

$$(P) \qquad \max\{\, cx \,:\, Ax \leq b \,,\; x \in \mathbb{Z}^n \,\}$$

il suo rilassamento continuo

$$(\bar{P}) \qquad \max\{\, cx \,:\, Ax \leq b \,\}$$

si ottiene semplicemente rimuovendo il *vincolo di integralità* sulle variabili. È quindi immediato costruire il rilassamento continuo di un problema di *OC* una volta che lo si sia formulato come *PLI*. In generale si avranno quindi molti possibili rilassamenti continui per uno stesso problema di *OC*, uno per ogni diversa formulazione *PLI* del problema. Tra questi uno è "ottimo", come discusso nel §4.2.1, ma la formulazione corrispondente non è in generale disponibile. Il fondamentale pregio del rilassamento continuo è che, essendo un problema di *PL*, abbiamo a disposizione algoritmi efficienti per risolverlo.

6.1.1 Efficacia del rilassamento continuo

Il rilassamento continuo può quindi essere considerato efficiente; in alcuni casi è anche ragionevolmente efficace. Ad esempio, molte delle dimostrazioni di garanzia delle prestazioni di algoritmi greedy riportate nel §5.1.2 possono essere "rilette" come *dimostrazioni di efficacia* di rilassamenti continui. Infatti, si consideri ad esempio l'algoritmo greedy per il Weighted Vertex Cover (cf. §5.1.2.4); la relazione

$$\sum_{\{ i , j \} \in E} y_{ij} \leq z(\underline{\text{WVC}}) \leq z(\text{WVC}) \leq 2 \sum_{\{ i , j \} \in E} y_{ij}$$

mostra come il gap relativo ottenuto dal rilassamento continuo della formulazione di *PLI* "naturale" del problema non sia mai superiore al 100%.

Esercizio 6.1. Si ripeta lo stesso ragionamento per derivare valutazioni al caso pessimo per i gaps dei rilassamenti continui di (KP) e (MMMS); per quest'ultimo si mostri prima che il valore ottimo della funzione obiettivo del rilassamento continuo è esattamente pari a L.

Nella pratica, una valutazione del valore ottimo della funzione obiettivo affetta da un gap del 100% difficilmente può essere considerata accurata; usualmente si considerano tali, a seconda del problema in esame, valutazioni con un gap di pochi punti percentuali. Essendo questa una stima dell'errore nel caso pessimo, naturalmente, è possibile che le valutazioni prodotte siano considerevolmente più precise; si ha cioè anche per i rilassamenti la distinzione tra comportamento nel caso pessimo e comportamento "medio" su una classe di istanze, come già discusso nel §5.1.2 per le euristiche. Ad esempio, per il caso di (MMMS) il gap dal rilassamento continuo su istanze generate in modo pseudo-casuale è molto spesso una piccola frazione dell'1%, ossia molto migliore dal 33% che si potrebbe desumere dai risultati del §5.1.2.2. Tuttavia, in molti casi la valutazione del valore ottimo prodotta dal rilassamento continuo è di qualità fortemente scadente, come mostrato dai due esempi seguenti.

Esempio 6.1. Assegnamento di frequenze

Si consideri il problema dell'assegnamento di frequenze descritto nell'Esempio 1.11: è facile dimostrare che, qualsiasi siano la dimensione e la densità del grafo, la valutazione inferiore sul valore ottimo prodotta dal rilassamento continuo della formulazione *PLI* fornita del problema, ossia

$$(\underline{\text{GC}}) \qquad \begin{array}{lllll} \min & \sum_{f \in F} y_f & & & \\ & \sum_{f \in F} x_{if} = 1 & i \in S & & \\ & x_{if} + x_{jf} \leq 1 & \{ i , j \} \in E & f \in F \\ & y_f \geq x_{if} \geq 0 & i \in S & f \in F \\ & y_f \in [0 , 1] & & f \in F \end{array}$$

è non superiore a 1. Per questo si consideri la soluzione (frazionaria, poiché ovviamente $m = | F | > 1$) che pone $y_f = x_{if} = 1 / m$ tutti gli $f \in F$ e $i \in S$; in altre parole, in questa soluzione si assegna $1 / m$ di ciascuna frequenza a ciascun vertice. Tale soluzione è ammissibile per ($\underline{\text{GC}}$) in quanto $x_{if} + x_{jf} = 2 / m \leq 1$ e $\sum_{f \in F} x_{if} = 1$; il corrispondente valore della funzione obiettivo è 1, e quindi $z(\underline{\text{GC}}) \leq 1$. Si noti che questo è valido anche per un grafo completo, che richiede un numero di frequenze esattamente pari al numero dei vertici, di qualsiasi dimensione; quindi, la valutazione inferiore sul valore ottimo della funzione obiettivo fornita da ($\underline{\text{GC}}$) può essere affetta da un gap arbitrariamente grande. Il problema in questo caso è che, passando al rilassamento continuo, il vincolo numerico "$x_{if} + x_{jf} \leq 1$" perde completamente il suo originario significato logico "al massimo uno tra i due valori x_{if} e x_{jf} può essere diverso da zero".

Esempio 6.2. Il problema (CMST)

Per poter presentare un rilassamento continuo di (CMST) dobbiamo in primo luogo introdurne una formulazione in termini di *PLI*. Come abbiamo visto nell'Esempio 4.6 sarebbe possibile, per costruirne una, partire dalla formulazione "per tagli" dell'Esempio 4.4 separando iterativamente le cutset inequalities (1.12).

Esercizio 6.2. Si proponga una formulazione per (CMST) basata sulle cutset inequalities.

Supponiamo però che si decida di non procedere per questa via, ad esempio per via della complessità dell'implementazione; potrebbe essere allora preferito partire dalla formulazione "compatta" dell'Esempio 4.4. Questa in effetti può essere abbastanza facilmente modificata introducendo, come per le variabili di flusso, due copie x_{ij} e x_{ji} delle variabili binarie per ciascun lato $\{ i , j \}$, con lo stesso costo (ciò può essere evitato per i lati incidenti in r ma

evitiamo di discutere questi raffinamenti per semplicità). La formulazione risultante è quindi

$$
\text{(CMST)} \qquad
\begin{array}{lll}
\min & \sum_{(i,j)\in A} c_{ij} x_{ij} & \\
& \sum_{(j,i)\in BS(i)} f_{ji} - \sum_{(i,j)\in FS(i)} f_{ij} = \begin{cases} -(n-1) & i = 1 \\ 1 & i \neq 1 \end{cases} & i \in V \\
& 0 \leq f_{ij} \leq Q x_{ij} & (i,j) \in A \\
& \sum_{(i,j)\in BS(j)} x_{ij} \leq 1 & j \in V \setminus \{r\} \\
& x_{ij} \in \{0,1\} & (i,j) \in A
\end{array}
$$

Sostituendo $n-1$ con Q nel vincolo che lega le x_{ij} con le f_{ij} assicura che nessun (arco corrispondente ad un lato) (i,j) porti più di Q unità di flusso, e quindi che nessun sottoalbero di radice j (per nessun j, e quindi in particolare per quelli adiacenti alla radice) possa avere cardinalità superiore a Q. Per questo però occorre assicurarsi anche che in j non possa entrare flusso—proveniente dalla radice—da più di un arco, il che è assicurato dal vincolo corrispondente, che però a sua volta richiede di duplicare le variabili x_{ij} per poterne distinguere l'orientamento. Si noti che non può mai essere conveniente avere flusso su entrambi gli archi (i,j) e (j,i), in quanto questo porterebbe a pagare due volte il lato $\{i,j\}$; pertanto non è necessario imporre esplicitamente un vincolo di esclusione $x_{ij} + x_{ji} \leq 1$.

Passare al rilassamento continuo di tale formulazione ha due effetti sulla struttura delle soluzioni ammissibili. Il primo è che, analogamente a quanto visto nell'esempio precedente, possono assumere valore diverso da zero più variabili x corrispondenti ad archi che entrano nello stesso nodo (ancorché la loro somma non possa superare 1); quindi, l'insieme degli archi corrispondenti a variabili $x_{ij} > 0$ può contenere strettamente un albero di copertura, il che non permette più di definire i sottoalberi della radice (il flusso che soddisfa la richiesta di un qualche nodo i può passare su più di un arco uscente dalla radice). Inoltre, è facile vedere che se f^* è il flusso corrispondente ad una soluzione ottima del rilassamento continuo, allora il valore delle corrispondenti variabili x è

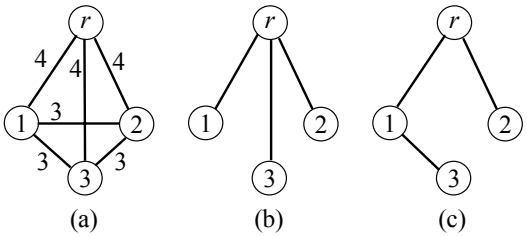


Figura 6.1: Un'istanza di CMST (a) e due soluzioni ammissibili

$$f^*_{ij} = x^*_{ij} / Q \quad (i,j) \in A \ .$$

In altre parole, il costo che viene "visto" dal rilassamento continuo per utilizzare un lato $\{i,j\}$ non è necessariamente il costo originale c_{ij}, ma una sua frazione che dipende da quanto flusso passa sull'arco. Quindi, anche se le variabili x individuassero un albero di copertura per G, che sarebbe quindi ammissibile per i vincoli relativi al massimo peso dei sottoalberi, il valore ottimo della funzione obiettivo sarebbe una valutazione inferiore, potenzialmente molto distante, del reale costo di quell'albero. Si consideri ad esempio l'istanza di (CMST) mostrata in Figura 6.1(a), in cui tutti i vertici hanno peso unitario e $Q = 2$. È facile verificare che la soluzione ottima del rilassamento continuo si ottiene ponendo $f_{r1} = f_{r2} = f_{r3} = 1$, $x_{r1} = x_{r2} = x_{r3} = 1/2$, ed a zero tutte le altre variabili. Essa corrisponde (arrotondando le variabili x) all'albero mostrato in Figura 6.1(b): il valore della funzione obiettivo del rilassamento continuo è 6, mentre il costo dell'albero è 12 (ciascun lato viene "pagato", nel rilassamento continuo, esattamente la metà del costo originale perché ci passa solo una delle due unità di flusso che potrebbero passarci). Tale albero non è ottimo, infatti esiste un albero di costo 11 mostrato in Figura 6.1(c); ma la migliore soluzione del rilassamento continuo compatibile con tale albero è $f_{r1} = 2$, $f_{r2} = x_{13} = 1$, $x_{r1} = 2$, $x_{r2} = x_{13} = 1/2$, di costo 7.5. Quindi, il rilassamento continuo non "indirizza" verso la soluzione intera ottima.

Esercizio 6.3. Si discuta come sia possibile migliorare la valutazione inferiore ottenuta dal rilassamento continuo modificando la formulazione *PLI*. In particolare, nei vincoli tra f_{ij} ed x_{ij} per gli archi (i,j) che non escono direttamente dalla radice Q può essere sostituito con un valore inferiore, che pertanto—restringendo la regione ammissibile—può far aumentare il valore ottimo del rilassamento.

In questo caso, quindi, il costo di utilizzare un certo lato all'interno di una soluzione può essere fortemente sottostimato rispetto al costo originale. Ragionamenti analoghi possono essere fatti per problemi quali il progetto di rete (cf. Esempio 1.15), la dislocazione ottima di impianti (cf. Esempio 1.23) e, più in generale, i problemi in cui siano presenti variabili semicontinue, ad esempio per modellare funzioni lineari a tratti (cf. §1.2.8) o vincoli disgiuntivi (cf. §1.2.10). Questi modelli contengono vincoli numerici del tipo "$x \leq My$", dove M può avere un valore numerico "grande" (vengono infatti spesso denominati come vincoli "big-M"): passando al rilassamento continuo tali vincoli perdono completamente il loro originario significato logico "y è pari a 1 se x è maggiore di zero". Infatti, nel rilassamento continuo la risorsa rappresentata da y (in assenza di altri vincoli) può essere costruita—-e quindi pagata—solo limitatamente alla frazione x / M strettamente necessaria a permettere il dato valore di x. In altri termini, sia ad esempio $c > 0$ il coefficiente di y in funzione obiettivo: il costo di x non è, come nelle intenzioni originarie, una funzione a carico fisso in cui si paga c non appena $x > 0$

(e zero se $x = 0$), ma la sua approssimazione inferiore (convessa) $(c\,/\,M)x$. Tale approssimazione coincide con la funzione originaria solo per $x = 0$ ed $x = M$, ma ha un valore potenzialmente molto inferiore (in particolare se, appunto, M è "grande") per i valori intermedi, soprattutto quelli vicini a 0. Questo può portare a sottostimare drasticamente il reale costo di una soluzione.

Gli esempi precedenti mostrano il principale limite del rilassamento continuo. In un modello *PLI* di un problema di *OC* le variabili intere vengono utilizzate principalmente per esprimere condizioni logiche; il rilassamento del vincolo di integralità può distruggere quindi la struttura fondamentale del modello, risultando in un problema di ottimizzazione le cui soluzioni ottime possono avere poco o nulla a che fare con quelle del problema originario. Nella rappresentazione geometrica discussa nel §4.2.2, la regione ammissibile del rilassamento continuo è "troppo più grande" di quella dell'inviluppo convesso delle soluzioni intere e quindi contiene soluzioni frazionarie con un valore molto minore di quella ottima intera. Ciò giustifica intuitivamente il fatto che, spesso, il rilassamento continuo risulti scarsamente efficace, anche se esistono tecniche—che verranno illustrate in seguito—che possono permettere di diminuire il gap corrispondente.

6.1.2 Informazione generata dal rilassamento continuo

Una volta risolto il rilassamento continuo di un problema di *OC*, ad esempio utilizzando il metodo del simplesso, si ha a disposizione, oltre ad una valutazione superiore del valore ottimo della funzione obiettivo, molta informazione che può essere sfruttata all'interno del processo risolutivo del problema. In particolare si hanno a disposizione una soluzione primale ottima x^* ed una corrispondente soluzione duale ottima y^*; nel seguito illustreremo brevemente alcuni possibili usi di tale informazione.

6.1.2.1 Uso dell'informazione primale

Come enunciato attraverso il Lemma 4.1, e ricordato per il problema dello zaino, l'informazione primale prodotta dal rilassamento continuo è di grande utilità qualora risulti intera: infatti, in questo caso è certamente una soluzione ottima per il problema *OC* che si vuole risolvere. Qualora questo non avvenga, ossia x^* abbia componenti frazionarie, è spesso possibile utilizzare la soluzione frazionaria per guidare la costruzione di "buone" soluzioni intere attraverso opportune euristiche, le più semplici delle quali sono le *tecniche di arrotondamento*. Abbiamo già di fatto visto all'opera una tecnica di arrotondamento nel §5.1.2.1 per il caso del problema dello zaino, in cui sappiamo che al più una variabile può essere frazionaria. L'euristica non si limita però semplicemente ad arrotondare la soluzione frazionaria (ossia ad escludere dallo zaino l'elemento "critico"), ma utilizza la soluzione ammissibile così ottenuta come punto di partenza per raffinarla ulteriormente (aggiungendo altri oggetti nello spazio rimasto disponibile).

Discutiamo adesso un caso più complesso ed interessante relativo al Problema di Copertura presentato nel §1.2.5. Consideriamo il rilassamento continuo

$$(\underline{\text{SC}}) \qquad \min \left\{ \sum_{j\in F}^{m} c_j x_j \;:\; \sum_{j\,:\,i\in F_j} x_j \geq 1 \quad i \in E \;\;,\;\; x \in [\,0\,,\,1\,]^m \right\}$$

e sia x^* una sua soluzione ottima. Un modo ovvio di determinare una soluzione ammissibile per (PC) è quello di selezionare tutti gli insiemi F_j tali che $x^*_j > 0$; ciò però tipicamente produce soluzioni di scarsa qualità. Un modo migliore per costruire una soluzione ammissibile consiste nel selezionare solo gli insiemi che hanno x^*_j "sufficientemente grande". A tal fine, definiamo la *frequenza* f_i di ciascun elemento $i \in E$ come il numero di sottoinsiemi F_j di cui i fa parte, e sia $f = \max\{\, f_i \;:\; i \in E \,\}$ la massima delle frequenze. Costruiamo quindi una soluzione S per (SC) selezionando tutti gli insiemi F_j tali che $x^*_j \geq 1\,/\,f$. S è sicuramente una soluzione ammissibile per (SC): infatti, dato un qualsiasi elemento $i \in E$, esiste almeno un sottoinsieme F_j che lo contiene e che ha $x^*_j \geq 1\,/\,f$ (nel caso peggiore tutti gli insiemi j che contengono i hanno $x^*_j = 1\,/\,f$, analogamente a quanto accade nell'Esempio 6.1). È facile dimostrare che questa soluzione ha un errore relativo al più $f - 1$. Infatti, si consideri la soluzione intera $\bar{x}$ ad essa corrispondente: per ogni j tale che $\bar{x}_j = 1$ si ha $x^*_j \geq 1\,/\,f$, e quindi $\bar{x}_j \leq f x^*_j$. Quindi il costo di S, $c\bar{x}$, è minore o uguale di fcx^*, ma $cx^* \leq z(PC)$, da cui

$$R_{arr} = \frac{c\bar{x} - z(PC)}{z(PC)} \leq \frac{fcx^* - cx^*}{cx^*} = f - 1\,.$$

Sono state proposte tecniche di arrotondamento per molti altri problemi di *OC*; spesso per queste tecniche è anche possibile dimostrare garanzie sulle prestazioni, come il caso di (SC) mostra. Per ulteriori dettagli su questi temi si rimanda alla letteratura citata.

Più complesso, ma comunque possibile ed interessante, è il caso in cui si vogliano sviluppare tecniche di arrotondamento che possano funzionare per "qualsiasi" *PLI*. In generale, non appena la matrice dei coefficienti A ha elementi di segno opposto per la stessa variabile j, ossia esistono due indici tali che $A_{ij} < 0$ e $A_{hj} > 0$—cosa che non capitava negli esempi di (KP) e (SC))—arrotondare una soluzione frazionaria ammissibile x^* (tale che $Ax^* \leq b$, ed assumiamo per semplicità di discussione che sia anche $x^* \geq 0$) ottenendo una soluzione intera ammissibile è in generale difficile: arrotondare x^*_i all'intero superiore rende "più ammissibile" il vincolo i-esimo ma può far perdere l'ammissibilità nell'h-esimo, mentre arrotondarla all'intero inferiore ha l'effetto opposto. Quando questo capiti per molte variabili (frazionarie) e vincoli diventa quindi complesso anche solo determinare se esiste una direzione di arrotondamento per ciascuna di esse che permette di costruire una soluzione ammissibile: del resto non potrebbe essere diversamente visto che la sola ammissibilità della *PLI* è $\mathcal{NP}$-hard (cf. §1.9). Non ci si può quindi aspettare di costruire approcci che funzionino sempre, né che approcci che funzionano in molti casi possano essere basati su regole semplici. Esistono però alcuni approcci iterativi generali e ragionevolmente facili da descrivere che hanno avuto un buon successo nella pratica: le idee fondamentali di uno di essi sono descritte nello pseudo-codice seguente.

```
procedure x = FeasibilityPump( A , b , c ) {
  x = argmin{ cx : Ax ≤ b };
  while( x ∉ ℤ^n ) {
    x̄ = ⌊ x ⌉;
    if( Ax̄ ≤ b ) then x = x̄;
                 else x = argmin{ ‖ z − x̄ ‖_∞ : Az ≤ b };
    }
  }
```

Procedura 6.1: Feasibility Pump

La procedura inizia calcolando la soluzione ottima del rilassamento continuo, e termina non appena (se) ha determinato una soluzione ammissibile intera. Se la soluzione corrente x, che è ammissibile, non è intera si provvede ad arrotondarla all'intero più vicino, ottenendo una soluzione $\bar{x} \in \mathbb{Z}^n$. Se tale soluzione risulta intera l'algoritmo termina, altrimenti determina la soluzione ammissibile (continua) più vicina a $\bar{x}$ in una "norma lineare" quale ad esempio quella L_∞ (di modo tale che il problema corrispondente possa essere scritto come *PL*, cf. §1.19) ed itera. L'intuizione dietro tale procedura è che l'operazione di arrotondamento "pompi integralità nella soluzione frazionaria x", mentre la soluzione del problema di *PL* "pompi ammissibilità nella soluzione intera $\bar{x}$" (da cui il nome di *feasibility pump* dato all'approccio), di modo tale che gradualmente si vada verso una soluzione che ha entrambe le proprietà; si è però visto che l'approccio può essere interpretato in diversi modi interessanti che giustificano in modo più formale il buon successo computazionale che ha in pratica. Le implementazioni reali di questa procedura sono significativamente più complesse di quanto brevemente descritto qui; in particolare, non è affatto ovvio che l'algoritmo determini una soluzione ammissibile (dato che potrebbe non esservene alcuna) e quindi termini. In effetti capita tipicamente che l'algoritmo determini la stessa $\bar{x}$ in due iterazioni successive, il che lo farebbe entrare in un ciclo infinito: se ciò accade o la procedura viene terminata oppure si applica una ripartenza (cf. §5.3), tipicamente ottenuta perturbando casualmente la soluzione $\bar{x}$ in un certo numero di variabili. Altre varianti cercano di tenere in conto anche del valore della funzione obiettivo mentre risolvono il problema di determinare la nuova soluzione continua ammissibile, ad esempio aggiungendo alla funzione obiettivo un termine $(\varepsilon c)x$ per un "piccolo" ε. Da questa idea iniziale sono state sviluppate molte versioni, per le quali si rimanda alla letteratura citata; algoritmi di questo tipo sono attualmente implementati in tutti i più efficienti solutori di *PLI* e permettono spesso di determinare—anche se ad un costo potenzialmente elevato, data la necessità di risolvere molti problemi di *PL*—soluzioni ammissibili per molte istanze di *PLI* sulle quali in precedenza le tecniche di arrotondamento più basilari fallivano sovente.

L'informazione primale generata dal rilassamento continuo ha anche altri usi importanti, ad esempio all'interno delle *regole di separazione* degli algoritmi di enumerazione implicita che verranno discussi nel Capitolo 7.

6.1.2.2 Uso dell'informazione duale

La soluzione duale y^* del rilassamento continuo di (una formulazione *PLI* di) un problema di *OC* può essere interpretata come un'indicazione dell'"importanza" relativa dei diversi vincoli lineari della formulazione. Abbiamo infatti visto nel §3.2.3 come le variabili duali ottime di un problema di *PL* forniscano i *prezzi o costi ombra* delle risorse associate a ciascun vincolo; in particolare, le condizioni degli scarti complementari assicurano che siano diverse da zero solo variabili duali corrispondenti a vincoli soddisfatti come uguaglianza, ossia a risorse completamente sfruttate. Quindi, il fatto che $y_i^* = 0$ indica che il corrispondente vincolo primale $A_i x \leq b_i$ è irrilevante ai fini della determinazione della soluzione ottima del rilassamento continuo. Si può quindi pensare che le variabili duali contengano una qualche indicazione su quali dei vincoli del problema siano più o meno rilevanti ai fini della determinazione della soluzione ottima del problema di *OC* originario. Questa indicazione è comunque da considerarsi euristica, in quanto è valida per la soluzione ottima del rilassamento continuo, che, come abbiamo visto, può essere fortemente scorrelata dalla soluzione ottima del problema di *OC*. Se però il rilassamento continuo fornisce una "buona approssimazione" del problema di *OC*, si può pensare che la soluzione del rilassamento continuo fornisca informazione rilevante per la soluzione del problema di *OC*; questa è in effetti la giustificazione delle tecniche di arrotondamento che abbiamo visto nel paragrafo precedente. In questo caso si può pensare che la soluzione duale contenga anch'essa informazione rilevante per la soluzione ottima del problema. Anche qualora il rilassamento continuo sia molto accurato, questa informazione è da considerarsi comunque "approssimata". Pur senza entrare nel dettaglio, notiamo a questo proposito che il numero di variabili duali diverse da zero, ossia di vincoli "importanti", in un problema di *PL* nella forma di (P) da noi comunemente utilizzata, è al più pari al numero n delle variabili; viceversa, si può dimostrare che il numero di vincoli di una formulazione *PLI* la cui rimozione cambia la soluzione ottima del problema—e che quindi sono decisamente "importanti"—può essere anche $O(\,2^n\,)$. Quindi, l'informazione duale fornita dal rilassamento continuo non può essere corretta in tutti i casi; esistono però problemi per i quali tale informazione risulta comunque utile.

Descriviamo adesso un possibile modo per sfruttare l'informazione duale prodotta dal rilassamento continuo, che viene comunemente indicato come *fissaggio basato sui costi ridotti*. Questo approccio si applica a problemi di *PLI* in cui le variabili abbiano limitazioni superiori e/o inferiori; per semplicità ci limiteremo ad esporlo per il caso della Programmazione 0/1. Sia quindi dato il problema

$$(P) \qquad \max\{\, cx \,:\, Ax \leq b \,,\; x \in \{\,0\,,\,1\,\}^n \,\}$$

ed il suo rilassamento continuo

$$(\bar{P}) \qquad \max\{\, cx \,:\, Ax \leq b \,,\; x \in [\,0\,,\,1\,]^n \,\} \;\;;$$

il duale di $(\bar{P})$ è

$$(\bar{D}) \qquad \min\{\, yb + ze \,:\, yA + z - w = c \,,\; y \geq 0 \,,\; z \geq 0 \,,\; w \geq 0 \,\}$$

dove e è il vettore avente tutte le componenti pari ad 1. Siano adesso x^* ed $[\,y^*\,,\,z^*\,,\,w^*\,]$ una coppia di soluzioni ottime rispettivamente per $(\bar{P})$ e $(\bar{D})$; indichiamo con

$$c_i^* = c_i - y^* A^i = z_i^* - w_i^*$$

il *costo ridotto* della variabile x_i. Dalle condizioni degli scarti complementari (si vedano ad esempio le discussioni fatte nei §5.1.2.1, 5.1.2.4 e 5.1.3) si ottengono facilmente le seguenti relazioni:

$$\begin{array}{lcl} z_i^* > 0 \implies w_i^* = 0 & \text{e} & w_i^* > 0 \implies z_i^* = 0 \\ x_i^* < 1 \implies c_i^* \leq 0 & \text{e} & x_i^* > 0 \implies c_i^* \geq 0 \\ c_i^* > 0 \implies x_i^* = 1 & \text{e} & c_i^* < 0 \implies x_i^* = 0 \end{array} \quad .$$

Il costo ridotto di una variabile è quindi dato dai prezzi ombra dei vincoli $x_i \geq 0$ e $x_i \leq 1$; in particolare, se il costo ridotto è positivo allora il vincolo $x_i \leq 1$ è soddisfatto come uguaglianza

($x_i = 1$), mentre se il costo ridotto è negativo allora il vincolo $x_i \geq 0$ è soddisfatto come uguaglianza ($x_i = 0$). Se disponiamo di una valutazione inferiore $\underline{z}$ su $z(P)$, tipicamente data dal costo di una soluzione ammissibile, possiamo utilizzare i costi ridotti per cercare di determinare se alcune delle variabili che hanno valore intero (0 o 1) in x^* debbano avere tale valore anche nella soluzione ottima di (P). Supponiamo ad esempio che sia $x_i^* = 0$ e $c_i^* = -w_i^* < 0$, e consideriamo il problema di *PL* $(\bar{P}')$ identico a $(\bar{P})$ tranne per il fatto che il vincolo $x_i \geq 0$ è rimpiazzato col vincolo $x_i \geq 1$ (nella forma standard, il vincolo, $-x_i \leq 0$ diviene $-x_i \leq -1$, ossia il lato destro del vincolo è *diminuito* di un'unità); come abbiamo visto nel paragrafo 3.2.3, si ha che

$$z(\bar{P}') \leq z(\bar{P}) + (-1)w_i^* = z(\bar{P}) + c_i^* \ .$$

Se quindi accade che $z(\bar{P}) + c_i^* < \underline{z}$, allora è possibile fissare $x_i = 0$, perchè tale deve essere il valore della variabile in tutte le soluzioni ottime di (P). Infatti, $(\bar{P}')$ è il rilassamento continuo del problema di *PLI* (P') identico a (P) tranne per il fatto che x_i è fissata al valore 1 (sono presenti i vincoli $1 \leq x_i \leq 1$); si ha quindi

$$z(P') \leq z(\bar{P}') \leq z(\bar{P}) + c_i^* < \underline{z} \leq z(P) \ ,$$

ossia fissare ad 1 la variabile x_i produce certamente un peggioramento del valore ottimo della funzione obiettivo di (P), e quindi tale variabile può essere fissata ad 0. Si noti che, per essere in grado di fissare la variabile, deve risultare

$$z(\bar{P}) - z(P) \leq z(\bar{P}) - \underline{z} \leq -c_i^* \ ,$$

ossia il costo ridotto deve essere (in valore assoluto) più grande del gap assoluto corrispondente al rilassamento continuo. Quindi, sarà tanto più facile fissare variabili in base ai costi ridotti quanto più il gap del rilassamento continuo è piccolo e $\underline{z}$ è una valutazione accurata di $z(P)$.

Esercizio 6.4. Si ripeta il ragionamento precedente per il caso in cui risulti $c_i^* > 0$ e $x_i^* = 1$.

Esercizio 6.5. Si estenda il procedimento al caso più generale di un problema di *PLI* in cui le variabili hanno limiti superiori ed inferiori $l_i \leq x_i \leq u_i$; si può ancora parlare di "fissaggio" di variabili?

Esercizio 6.6. Si discutano le relazioni tra il procedimento appena visto e l'Esempio 4.8.

L'informazione duale prodotta dal rilassamento continuo può quindi essere usata per dedurre proprietà importanti della soluzione ottima. Comunque, vedremo un ulteriore possibile utilizzo di tale informazione duale al termine di questo capitolo.

6.2 Eliminazione di vincoli

Come abbiamo visto nel paragrafo precedente, rilassare il vincolo di integralità può distruggere quasi completamente la struttura di un problema, rendendo inefficace il rilassamento. In questo caso è quindi necessario sviluppare metodologie alternative per costruire rilassamenti efficaci. Moltissimi problemi di *PLI* posseggono un qualche tipo di "struttura" che può essere sfruttata per costruire rilassamenti. Un possibile modo generale per descrivere la struttura di un problema è quello di considerare formulazioni del tipo

$$(P) \qquad \max\{\, cx \,:\, Ax \leq b \,,\ Ex \leq d \,,\ x \in \mathbb{Z}^n, \} \tag{6.1}$$

in cui i vincoli $Ex \leq d$ sono "facili", ossia la rimozione dei *vincoli complicanti* $Ax \leq b$ trasformerebbe (P) in un problema "facile". In altre parole, l'introduzione dei vincoli $Ax \leq b$ distrugge la "struttura" presente nei vincoli $Ex \leq d$, che invece permetterebbe di usare algoritmi specifici, più efficienti, per la soluzione del problema. Alternativamente, o in aggiunta a questo, i vincoli $Ex \leq d$ possono essere *separabili*, ossia il problema si decomporrebbe in sottoproblemi indipendenti di dimensione inferiore se non fosse per la presenza dei vincoli $Ax \leq b$, che "legano insieme" le variabili dei singoli sottoproblemi; in questo caso i vincoli $Ax \leq b$ sono anche detti *vincoli accoppianti*. Chiaramente, in tale situazione un possibile rilassamento di (P) si ottiene rimuovendo dal problema i vincoli complicanti, ossia considerando il problema

$$(P') \qquad \max\{\, cx \,:\, Ex \leq d \,,\ x \in \mathbb{Z}^n \,\} \ .$$

(P') viene detto *rilassamento per eliminazione di vincoli* di (P) rispetto ai vincoli $Ax \leq b$. Poichè (P') è un problema di *PLI*, e quindi di *OC*, (P') viene anche detto *rilassamento combinatorio* per distinguerlo dal rilassamento continuo. Chiedere che (P') sia un rilassamento efficiente corrisponde ad assumere che la rimozione dei vincoli $Ax \leq b$ permetta di ottenere un problema "facile"; è bene però sottolineare che, in questo contesto, "facile" non significa necessariamente polinomiale, ma solamente sostanzialmente più facile da risolvere di (P), in pratica e per le dimensioni delle istanze in esame, ossia tale che sia possibile calcolare $z(P')$ in modo notevolmente più efficiente di quanto non sia possibile calcolare $z(P)$. È anche utile far presente che, in questo contesto, "risolvere" (P') significa calcolare *esattamente* $z(P')$, o al limite una sua (buona) valutazione superiore: infatti, una soluzione euristica di (P') non necessariamente fornisce una valutazione superiore di $z(P)$.

6.2.1 Esempi di rilassamenti per eliminazione di vincoli

In generale, dato un problema di *OC*, ed una sua formulazione *PLI*, possono esistere più modi per costruire rilassamenti per eliminazione di vincoli, in quanto possono esistere più blocchi di vincoli la cui rimozione consenta di ottenere un problema "facile"; quindi, esiste spesso una molteplicità di rilassamenti per eliminazione di vincoli corrispondenti ad uno stesso problema di *OC*. Si noti che il rilassamento continuo potrebbe essere considerato una caso speciale del rilassamento per eliminazione di vincoli, in quanto corrisponde all'eliminazione dei vincoli di integralità. Inoltre, può capitare che formulazioni *PLI* diverse dello stesso problema suggeriscano rilassamenti diversi, in quanto evidenzino parti diverse della struttura del problema. Discutiamo adesso alcuni rilassamenti per eliminazione di vincoli per un certo numero di problemi di *OC*.

Esempio 6.3. Il problema (CMST)

Come è stato osservato in precedenza, molti problemi di *OC* "difficili" sono varianti apparentemente minori di problemi "facili". (CMST) è un tipico esempio. In casi come questi è solitamente facile individuare rilassamenti efficienti per eliminazione di vincoli; è sufficiente eliminare i vincoli corrispondenti alle condizioni "in più" rispetto a quelle presenti nel problema "facile". Per (CMST), ad esempio, questo significa rimuovere il vincolo sul massimo peso dei sottoalberi: (MST) è chiaramente un rilassamento efficiente per eliminazione di vincoli di (CMST). Tipicamente, l'efficacia di un tale rilassamento dipenderà dal valore della costante Q, che è completamente ignorata in (MST): se la costante è "grande" si può sperare che solo "pochi" sottoalberi della radice nella soluzione ottima di (MST) violino il vincolo, e quindi che il gap relativo al rilassamento sia basso (esiste certamente un valore di Q sufficientemente grande per cui il gap è nullo), mentre al diminuire di Q è presumibile che il gap cresca. Può essere interessante notare come la formulazione *PLI* compatta dell'Esempio 6.2 suggerisca un diverso rilassamento. In particolare, in questo caso possono essere considerati "complicanti" i vincoli $f_{ij} \leq Qx_{ij}$, che sono gli unici a "collegare"le variabili f e le variabili x (sono quindi "accoppianti"). Il problema ottenuto dalla rimozione di tali vincoli è quindi separabile in due sottoproblemi indipendenti: un problema di flusso sulle variabili f, ed il problema di selezionare, per ciascun nodo (tranne la radice), l'arco entrante di costo minimo, che è a sua volta separabile in $n-1$ sottoproblemi indipendenti. Poichè in effetti la soluzione del problema di flusso non ha alcuna influenza sul valore della funzione obiettivo (le variabili f hanno tutte coefficienti nulli), questo rilassamento può essere risolto in $O(m)$ semplicemente scandendo la lista degli archi del grafo. È facile verificare come questo rilassamento sia a sua volta un rilassamento di (MST): in un albero di copertura (orientato) ciascun nodo (tranne la radice) ha un arco entrante. Quindi, il rilassamento basato su (MST) ha sicuramente un gap inferiore, ma ha anche una complessità superiore. È interessante notare come questo rilassamento potrebbe essere stato ideato riflettendo sul problema di *OC* originale, ma venga facilmente suggerito dall'esame di una formulazione *PLI* del problema. Ciò mostra come formulazioni diverse delle stesso problema possano essere utili per suggerire rilassamenti diversi.

Esercizio 6.7. Si dimostri con due esempi che non esiste nessuna relazione di dominanza tra il gap fornito dal rilassamento continuo della formulazione compatta dell'Esempio 6.2 e quello basato su (MST), ossia che esistono istanze in cui l'uno è migliore dell'altro ed altre istanze in cui accade il contrario.

Esercizio 6.8. Si esaminino tutti i possibili rilassamenti per eliminazione di vincoli ottenibili dalla formulazione compatta dell'Esempio 6.2, discutendo per ciascuno di essi se possa essere considerato "facile" e se si possano individuare relazioni di dominanza rispetto al gap relativo ai rilassamenti discussi finora.

Esempio 6.4. Il problema del cammino minimo vincolato

Un caso analogo al precedente si ha per il problema del *cammino minimo vincolato* (CSP, da Constrained Shortest Path). Sia $G = (N, A)$ un grafo orientato e pesato dove ad ogni arco $(i, j) \in A$ è associato un costo $c_{ij} \in \mathbb{R}_+$ ed una lunghezza $l_{ij} \in \mathbb{R}_+$, e siano dati i due nodi r e t: si vuole determinare un cammino di costo minimo tra tutti i quelli da r a t di lunghezza inferiore o uguale ad una data soglia L. Una formulazione *PLI* per (CSP) può essere

ottenuta immediatamente da quella del problema del cammino minimo introdotta nel §2.2.1

$$\begin{array}{lll} \min \sum_{(i,j)\in A} c_{ij}x_{ij} & & \\ \sum_{(j,i)\in BS(i)} x_{ji} - \sum_{(i,j)\in FS(i)} x_{ij} = b_i & i \in N & (6.2) \\ \sum_{(i,j)\in A} l_{ij}x_{ij} \le L & & (6.3) \\ x_{ij} \in \{0,1\} & (i,j) \in A & \end{array}$$

dove $b_i = -1$ se $i = r$, $b_i = 1$ se $i = t$, e $b_i = 0$ altrimenti. Per questo problema, il singolo vincolo (6.3) risulta "complicante"; l'introduzione del vincolo rende il problema $\mathcal{NP}$-arduo, mentre la sua eliminazione consente di ottenere un problema polinomiale. Quindi, un ovvio ed efficiente rilassamento per eliminazione di vincoli di (CSP) si ottiene eliminando il vincolo (6.3) relativo alla massima lunghezza dei cammini. Anche in questo caso si può pensare che l'efficacia del rilassamento sia legata al valore della costante L: per valori "grandi" il vincolo è scarsamente rilevante (o addirittura irrilevante per valori sufficientemente grandi) ed è ragionevole attendersi gap relativamente bassi, che però tenderanno a crescere con il diminuire di L. In effetti, per valori sufficientemente piccoli di L (CSP) non ha alcuna soluzione ammissibile, ossia $z(CSP) = +\infty$, mentre il rilassamento ha soluzione (se t è raggiungibile da r): in questo caso il gap è infinito. In questo caso non sono facilmente costruibili altri rilassamenti per eliminazione di vincoli "ragionevoli"; rilassando i vincoli (6.2) si otterrebbe infatti un problema la cui soluzione ottima è nulla, in quanto nessun vincolo forza le variabili ad assumere valori diversi da zero.

Esercizio 6.9. Si discuta la struttura delle soluzioni ammissibili del rilassamento continuo della formulazione data di (CSP) e le eventuali relazioni di dominanza, in termini di gap, tra i due rilassamenti. Cosa si può dire in termini di efficienza?

Esempio 6.5. Il problema del flusso "multicommodity" indivisibile

Un problema in parte analogo al precedente si presenta, spesso come sottoproblema, durante la progettazione di reti di comunicazione. È dato un grafo $G = (N, A)$, che rappresenta ad esempio una rete di comunicazione, con costi di routing $c_{ij} \in \mathbb{R}_+$ e capacità $u_{ij} \in \mathbb{R}_{++}$ associate agli archi. Su questo grafo è individuato un insieme K di coppie origine/destinazione (o_h, d_h), ciascuna con un'associata domanda di comunicazione δ_h. Per ciascuna coppia (o_h, d_h), $h \in K$, si vuole selezionare un singolo cammino lungo il quale instradare tutte le δ_h unità di flusso che rappresentano la comunicazione tra l'origine e la destinazione date, rispettando i vincoli di capacità associati agli archi, e minimizzando il costo di routing complessivo, dato dalla somma pesata dei costi dei cammini utilizzati. Introducendo variabili di flusso $x^h = [x^h_{ij}]_{(i,j)\in A}$ per ciascuna coppia $h \in K$ (detta *commodity*), il problema del *flusso Multicommodity "indivisibile" di costo minimo* (UMMCF, da Unsplittable Multicommodity Min Cost Flow problem) può essere formulato come

$$\begin{array}{lll} \min \sum_{h\in K} \delta_h \sum_{(i,j)\in A} c_{ij}x^h_{ij} & & \\ \sum_{(j,i)\in BS(j)} x^h_{ji} - \sum_{(i,j)\in FS(i)} x^h_{ij} = b^h_i & i \in N \ , \ h \in K & (6.4) \\ \sum_{h\in K} \delta_h x^h_{ij} \le u_{ij} & (i,j) \in A & (6.5) \\ x^h_{ij} \in \{0,1\} & (i,j) \in A \ , \ h \in K & (6.6) \end{array}$$

dove $b^h_i = -1$ se $i = o_h$, $b^h_i = 1$ se $i = d_h$, e $b^h_i = 0$ altrimenti. I vincoli di capacità (6.5) risultano "complicanti" per due motivi: il primo è che se fossero rimossi il problema si decomporrebbe in $|K|$ sottoproblemi indipendenti, uno per ciascuna coppia origine/destinazione, e quindi i vincoli risultano "accoppianti". Il secondo è che ciascuno di tali sottoproblemi è un problema di cammino minimo, e quindi può essere risolto molto efficientemente. La presenza dei vincoli (6.5) rende invece (UMMCF) "difficile", sia per le grandi dimensioni sia per il fatto che la totale unimodularità dei vincoli di flusso (6.4) viene perduta, rendendo il problema $\mathcal{NP}$-arduo. In altri termini, come nei due casi precedenti, mentre la soluzione ottima del rilassamento per eliminazione dei vincoli (6.5) è certamente intera, la soluzione ottima del rilassamento continuo può non esserlo.

Esercizio 6.10. Si mostri attraverso un esempio che, anche se tutti i dati del problema sono interi, può esistere una soluzione ottima del rilassamento continuo di (UMMCF) non intera.

Come negli esempi precedenti, la rimozione dei vincoli di capacità congiunta (6.5) può essere poco rilevante se le capacità sono "grandi", ma il gap tenderà a crescere qualora le capacità diminuiscano. Come per il caso di (CSP), rimuovere i vincoli di conservazione di flusso (6.4) porterebbe invece ad un problema che ha soluzione ottima nulla.

Esempio 6.6. Il problema (TSP)

Come è già stato notato nei §1.7, 1.10 e 5.1.3, (TSP) può essere visto come un problema di ottimizzazione il cui insieme ammissibile è l'intersezione tra quello di un problema di accoppiamento e quello di un problema di albero

di copertura di costo minimo. In altri termini, nella formulazione *PLI*

$$\text{(TSP)} \qquad \begin{array}{lll} \min & \sum_{\{i,j\}\in E} c_{ij}x_{ij} & \\ & \sum_{i\in S, j\notin S} x_{ij} \geq 1 & \emptyset \subset S \subset V \\ & \sum_{\{i,j\}\in E} x_{ij} = 2 & i \in V \\ & x_{ij} \in \{0,1\} & \{i,j\} \in A \end{array}$$

sia il primo blocco di vincoli (di connessione) che il secondo blocco di vincoli (di copertura per cicli) possono essere considerati "complicanti". La rimozione di ciascun blocco di vincoli lascia un rilassamento efficiente, in quanto risolubile con un algoritmo polinomiale (sia pure di diversa complessità); l'efficacia relativa dei due approcci dipenderà tipicamente dalla singola istanza a cui vengono applicati. (TSP) è un tipico caso in cui all'interno dello stesso problema sono presenti più "strutture" che, prese singolarmente, sono "facili" da trattare, ma la cui intersezione caratterizza un problema difficile. Per questi problemi è quindi possibile definire una molteplicità di rilassamenti per eliminazione di vincoli "ragionevoli", in ciascuno dei quali vengono mantenuti solamente i vincoli che caratterizzano ciascuna delle strutture "facili".

Può essere utile notare che, in qualche caso, si riescono a gestire, all'interno di una delle strutture "facili" del problema, "forme più deboli" dell'altra struttura. Per il (TSP), ad esempio, è facile dimostrare che tutte le coperture per cicli del grafo hanno esattamente n lati, mentre gli alberi di copertura ne hanno $n-1$. In altri termini, alla formulazione di (MST) (ad esempio quella (1.13)–(1.12)) si può aggiungere il vincolo

$$\sum_{\{i,j\}\in A} x_{ij} = n \ ;$$

questo viene detto il problema dell'*1-albero di copertura di costo minimo* (MS1-T), ed è un rilassamento valido di (TSP), non peggiore di (MST) (se i costi sono non negativi) in quanto (MST) è un rilassamento di (MS1-T).

Esercizio 6.11. Si dimostri l'affermazione precedente mostrando che (MS1-T) è un rilassamento surrogato di (TSP) (si veda il §6.4).

Un 1-albero di copertura di un grafo G è un sottografo connesso di G con esattamente n lati. Si può dimostrare che l'1-albero di copertura di costo minimo di$G$ può essere calcolato efficientemente determinando un albero di copertura T di costo minimo di G, e poi aggiungendo a T il lato di costo minimo in G che non appartenga già a T.

Esercizio 6.12. Si dimostri l'affermazione precedente (suggerimento: la procedura appena descritta è una modifica dell'algoritmo di Kruskal, cf. §2.3.1).

Quindi, introducendo in (MST) una forma rilassata dei vincoli di copertura si ottiene un rilassamento non peggiore ed altrettanto facile da risolvere. Nel prossimo paragrafo presenteremo un modo diverso per fornire ai rilassamenti combinatori informazione sui vincoli rilassati.

Per terminare questa sezione sottolineiamo che non per tutti i problemi di *OC* sia facile costruire rilassamenti combinatori "ragionevoli". Si consideri ad esempio il problema dello zaino: questo problema ha un solo vincolo, a parte quelli di integralità. Rilassare questo vincolo porta ad un problema privo di vincoli, la cui soluzione ottima consiste nel prendere tutti gli oggetti. Per un caso più rilevante si consideri invece il problema (MMMS), ed in particolare la sua formulazione *PLI* presentata nell'Esempio 1.18. Rilassare ciascuno dei blocchi di vincoli del problema porta ad un problema la cui soluzione ottima non fornisce alcuna reale informazione sull'istanza in questione. Infatti, rilassare i vincoli di semiassegnamento elimina tutti i vincoli che costringono alcune delle variabili x ad essere non nulle; di conseguenza, la soluzione ottima del rilassamento corrisponde a non assegnare nessun lavoro a nessuna macchina, con un conseguente makespan pari a zero. Rilassare i vincoli che computano il makespan, ossia quelli che garantiscono che la singola variabile t sia maggiore o uguale del tempo di completamento di ogni macchina, fa sì che nel rilassamento non ci siano vincoli che "legano" t alle variabili x; di conseguenza, qualunque semiassegnamento scelto avrebbe un valore della funzione obiettivo pari a zero. In entrambi i casi il rilassamento fornisce una valutazione inferiore del valore ottimo del problema pari a zero, per qualsiasi istanza del problema.

Gli esempi precedenti mostrano che eliminare completamente alcuni vincoli da un problema di *OC* può permettere di costruire un rilassamento efficiente; spesso però tali rilassamenti possono essere scarsamente efficaci, in quanto i vincoli eliminati possono avere un profondo impatto sulla caratterizzazione della soluzione ottima, caratterizzazione che viene completamente perduta nel rilassamento. Sarebbe quindi utile una tecnica in grado di eliminare i vincoli "complicanti" da un problema "tenendone traccia" in qualche modo; questo è lo scopo del rilassamento Lagrangiano, descritto nel prossimo paragrafo.

6.3 Rilassamento Lagrangiano

Un modo più sofisticato per utilizzare la struttura presente in problemi con la forma (6.1) è quello di effettuare un *rilassamento Lagrangiano* di (P) rispetto agli m vincoli complicanti $Ax \leq b$, ossia considerare il problema

$$(P_y) \qquad \max\{\, cx + y(\, b - Ax\,) \,:\, Ex \leq d\,,\; x \in \mathbb{Z}^n \,\} \tag{6.7}$$

per un fissato vettore $y \in \mathbb{R}^m_+$ di *moltiplicatori Lagrangiani*. Per y fissato, (P_y) è equivalente al rilassamento per eliminazione di vincoli tranne per il diverso vettore dei costi $c_y = c - yA$ (detti *costi Lagrangiani*); come visto nel paragrafo precedente è quindi possibile che (P_y) sia più facile da risolvere in pratica di (P). È immediato verificare che, comunque si scelga y, (P_y) è un rilassamento di (P), ossia risulta $z(P_y) \geq z(P)$. Infatti la regione ammissibile di (P_y) contiene quella di (P) e si ha

$$cx + y(\, b - Ax\,) \geq cx$$

per ogni x ammissibile per (P), ossia tale che $b - Ax \geq 0$. Il termine aggiuntivo $y(\, b - Ax\,)$ nella funzione obiettivo ha il compito di "penalizzare" le soluzioni che non rispettano i vincoli rilassati $Ax \leq b$: infatti, se $\bar{x}$ rispetta tutti i vincoli rilassati allora $y(\, b - A\bar{x}\,) \geq 0$, mentre se $\bar{x}$ viola un dato vincolo $A_i x \leq b_i$, e $y_i > 0$, allora $y_i(\, b_i - A_i\bar{x}\,) < 0$. Quindi, nel rilassamento Lagrangiano si "tiene traccia" dei vincoli complicanti: nonostante se ne permetta la violazione, si aggiunge alla funzione obiettivo un termine che favorisce le soluzioni che non lo fanno. Ovviamente il rilassamento per eliminazione di vincoli è un caso particolare del rilassamento Lagrangiano, in quanto si ottiene semplicemente ponendo $y = 0$. Vedremo nel seguito che, in un certo senso, il rilassamento Lagrangiano generalizza anche il rilassamento continuo, ed i moltiplicatori Lagrangiani sono funzionalmente analoghi alle variabili duali, nella PL, dei vincoli rilassati.

Infatti, mentre la discussione verrà fatta per semplicità sul problema (P) in forma normale, è chiaramente possibile rilassare in modo Lagrangiano ogni tipo di vincolo lineare, e le regole che determinano il segno dei moltiplicatori Lagrangiani sono le stesse di quelle valide per la dualità lineare (cf. (3.18)): il moltiplicatore Lagrangiano di un vincolo complicante nella forma $A_i x \geq b_i$ è vincolato ad essere non positivo ($y_i \leq 0$), ma accadrebbe il contrario se (P) fosse un problema di minimo. Inoltre, il moltiplicatore Lagrangiano di un vincolo complicante nella forma $A_i x = b_i$ non è vincolato in segno, ed il duale Lagrangiano (definito nel seguito) è un problema di massimo se (P) è un problema di minimo.

Il rilassamento Lagrangiano di un problema di OC non è un singolo problema, ma piuttosto una famiglia infinita di problemi dipendenti dal vettore di parametri y. Ciascuno di questi problemi fornisce una, potenzialmente diversa, valutazione superiore di $z(P)$; ha quindi senso porsi il problema di determinare la migliore, ossia la minore, di tali valutazioni. Ciò corrisponde a risolvere il *duale Lagrangiano*

$$(D) \qquad \min\{\, z(P_y) \,:\, y \geq 0 \,\} \tag{6.8}$$

di (P) rispetto ai vincoli complicanti $Ax \leq b$. Siccome vale $z(P_y) \geq z(P)$ per ogni $y \in \mathbb{R}^m_+$, (D) è ancora un rilassamento di (P), ossia vale $z(D) \geq z(P)$.

Esempio 6.7. Duale Lagrangiano di (CSP)

Si consideri ad esempio l'istanza di (CSP) rappresentata in Figura 6.2(a); il suo rilassamento Lagrangiano rispetto al vincolo "complicante" (6.3) è il problema di cammino minimo, parametrico rispetto al singolo moltiplicatore Lagrangiano $y \leq 0$ in cui la funzione obiettivo è

$$Ly + \min \textstyle\sum_{(\,i\,,\,j\,)\in A}(\, c_{ij} - y l_{ij}\,)x_{ij} \;\;,$$

rappresentato in Figura 6.2(b). Le soluzioni ammissibili del rilassamento sono i quattro cammini $c_1, \ldots, c_4$ mostrati in Figura 6.3(a1)–(a4) con il corrispondente costo Lagrangiano. Al variare di y il costo Lagrangiano delle singole soluzioni varia in modo differente, e quindi per ciascun valore di y

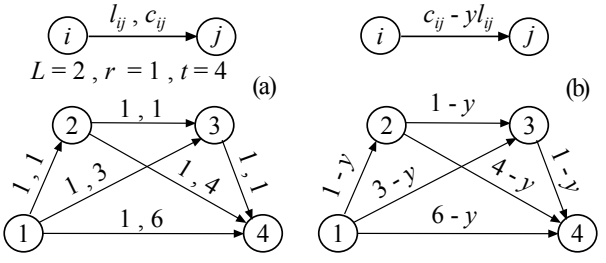


Figura 6.2: Un'istanza di CMST (a) ed il suo rilassamento Lagrangiano (b)

una (o più) soluzioni sarà quella di costo minimo. In particolare, come mostrato in Figura 6.3(b), il cammino c_1 ha il minimo costo Lagrangiano per $y \in [-1, 0]$, il cammino c_2 ha il minimo costo Lagrangiano per $y \in [-2, -1]$, il cammino c_4 ha il minimo costo Lagrangiano per $y \leq -2$ mentre il cammino c_3 non ha il minimo costo Lagrangiano per nessun valore di y; si noti che per $y = -1$ e $y = -2$ ci sono due soluzioni che hanno il medesimo costo Lagrangiano minimo (c_1 e c_2 nel primo caso, c_2 e c_4 nel secondo).

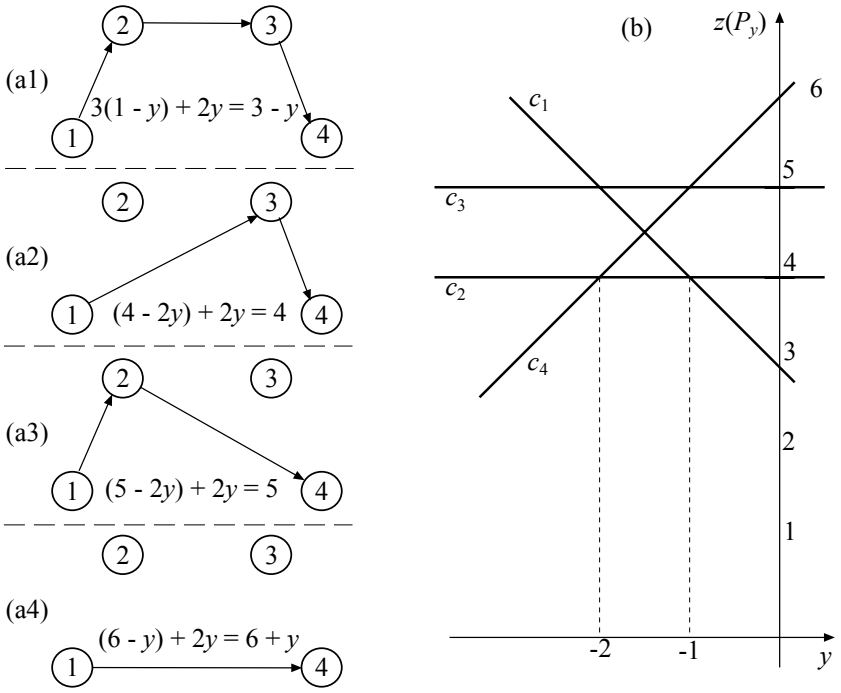


Figura 6.3: Soluzioni ammissibili del rilassamento Lagrangiano e duale Lagrangiano

L'esempio illustra chiaramente le caratteristiche del rilassamento Lagrangiano. Poichè tutti i coefficienti l_{ij} sono unitari, il costo Lagrangiano di un qualsiasi cammino p è pari al suo costo più il termine penalità $y(\,2 - l(p)\,)$, dove $l(p)$ è la lunghezza del cammino (numero di archi che lo compongono). Per $y = 0$ la lunghezza del cammino non viene tenuta in nessun conto, e la soluzione ottima del rilassamento Lagrangiano (e di quello per eliminazione di vincoli) è il cammino c_1, di costo 3 e lunghezza 3, quindi non ammissibile. Mano mano che y diminuisce il costo del cammino c_1, che viola il vincolo, aumenta, il costo dei cammini c_2 e c_3, che rispettano il vincolo come uguaglianza, resta inalterato, mentre il costo del cammino c_4, che rispetta il vincolo come stretta diseguaglianza, diminuisce. Per $y = -1$ il cammino c_2 ha lo stesso costo Lagrangiano di c_1, e per $y < -1$ ha un costo migliore. Per $y = -2$ il cammino c_4 ha lo stesso costo Lagrangiano di c_2, e per $y < -2$ ha un costo migliore; mano mano che y diminuisce i cammini "più corti" sono sempre più convenienti, in termini di costo Lagrangiano, rispetto a quelli "più lunghi". Si noti il ruolo importante del termine costante $Ly = 2y$: tale termine non dipende dal costo dei cammini, e quindi sarebbe ignorato quando si andasse ad utilizzare, ad esempio, l'algoritmo SPT per determinare il cammino minimo per un qualche y fissato, ma la sua presenza è cruciale per il funzionamento dell'approccio.

Per $y \in (\,-2\,, -1\,)$ l'unica soluzione del rilassamento Lagrangiano, ovvero il cammino c_2, rispetta il vincolo "complicante" come uguaglianza. Dalla Figura 6.3(b) è evidente che qualsiasi $y \in [\,-2\,, -1\,]$ è una soluzione ottima del duale Lagrangiano, ossia che $z(D) = 4$; questo è anche il costo nel problema (CSP) del cammino c_2, che infatti è la soluzione ottima del problema. In questo particolare caso il duale Lagrangiano fornisce una valutazione esatta di $z(P)$ (il gap è nullo): il termine di penalità $y(\,b - Ax\,)$, con un'opportuna scelta del moltiplicatore Lagrangiano y, "guida" il rilassamento Lagrangiano permettendogli di individuare la soluzione ottima del problema originario. Questo non sempre accade; in molti casi, come vedremo, il duale Lagrangiano ha un gap non nullo.

L'esempio precedente illustra la proprietà (ovvia) per cui il duale Lagrangiano fornisca una valutazione superiore non peggiore di quella fornita dal corrispondente rilassamento per eliminazione di vincoli, ovvero $z(D) \leq z(P_0)$; non potrebbe essere diversamente in quanto $y = 0$ è una delle soluzioni ammissibili di (D). Inoltre può accadere che il duale Lagrangiano individui una soluzione ottima di (P). Per questo non è sufficiente, come nel caso del rilassamento continuo (cf. il Lemma 4.1), che la soluzione del rilassamento Lagrangiano sia ammissibile per i vincoli rilassati, ma è necessaria anche la condizione

$$\bar{y}(\,b - A\bar{x}\,) = 0 \tag{6.9}$$

Lemma 6.1. Sia $\bar{x}$ una soluzione ottima di $P_{\bar{y}}$: se $\bar{x}$ è ammissibile per (P) ($A\bar{x} \leq b$) e vale (6.9) allora

$\bar{x}$ è una soluzione ottima per (P) e $\bar{y}$ è una soluzione ottima per (D).

Dimostrazione Si ha $c\bar{x} \leq z(P) \leq z(D) \leq z(P_{\bar{y}}) = c\bar{x} + \bar{y}(b - A\bar{x}) = c\bar{x}$. ◇

È interessante notare che (6.9) corrisponde alle condizioni degli scarti complementari introdotte nel caso della *PL* (cf. §3.2.4). Nel prossimo paragrafo mostreremo che questa corrispondenza non è casuale, e presenteremo alcuni risultati che consentono, in molti casi, di confrontare la qualità della valutazione superiore fornita da (D) con quella delle valutazioni fornite da rilassamenti diversi.

6.3.1 Teoria del rilassamento Lagrangiano

Nel caso della *PL* la dualità Lagrangiana coincide con quella lineare. Si consideri infatti il rilassamento continuo di (P)

$$(\bar{P}) \qquad \max\{\, cx \,:\, Ax \leq b \,,\; Ex \leq d \,\} \;, \tag{6.10}$$

il rilassamento Lagrangiano di $(\bar{P})$ rispetto ad $Ax \leq b$

$$(\bar{P}_y) \qquad \max\{\, cx + y(\, b - Ax \,) \,:\, Ex \leq d \,\} \tag{6.11}$$

ed il corrispondente duale Lagrangiano

$$(\bar{D}) \qquad \min\{\, z(\bar{P}_y) \,:\, y \geq 0 \,\} \;. \tag{6.12}$$

Iniziamo con l'escludere il caso in cui (6.11) è vuoto; questo non dipende da y si determina quindi facilmente ad esempio risolvendo il rilassamento per eliminazione di vincoli ($y = 0$), cosa che si assume essere "facile" e che quindi a maggior ragione implica che sia facile determinare se sia oppure no vuoto. Se ciò fosse il caso anche $(\bar{P})$ sarebbe banalmente vuoto e quindi lo avremmo risolto. Formalmente si ha che $z(\bar{P}_y) = -\infty$ per ogni y, quindi $z(D) = -\infty = z(P)$ come ci si attende. Il risultato è vero anche negli altri casi:

Teorema 6.1. $(\bar{D})$ è il duale—nel senso della *PL*—di $(\bar{P})$.

Dimostrazione Poiché $(\bar{P}_y)$ è un problema di *PL* il suo valore ottimo è lo stesso di quello del suo duale (tranne nel caso in cui siano entrambi vuoti, cf. Teorema 3.13, ma questo è stato precedentemente escluso). Si ha quindi

$$z(\bar{P}_y) = \min\{\, wd \,:\, wE = c - yA \,,\; w \geq 0 \,\} + yb \;,$$

e di conseguenza

$$\begin{aligned} z(\bar{D}) &= \min\{\, yb + \min\{\, wd \,:\, wE = c - yA \,,\; w \geq 0 \,\} \,:\, y \geq 0 \,\} \\ &= \min\{\, yb + wd \,:\, wE + yA = c \,,\; y \geq 0 \,,\; w \geq 0 \,\} \;; \end{aligned}$$

è immediato riconoscere nella precedente espressione il duale lineare di $(\bar{P})$. ◇

Quindi, nel caso della *PL* il duale Lagrangiano è in effetti il duale lineare. Si può dire in effetti che sia che sia un *duale parziale*: solo le variabili duali y corrispondenti ai vincoli "complicanti" sono esplicite, mentre le variabili duali w corrispondenti a quelli "facili" rimangono "nascoste" dentro il rilassamento Lagrangiano $(\bar{P}_y)$ (che è scritto in forma primale). Ciò giustifica le analogie tra i due duali precedentemente accennate.

Nel caso della *PLI* il rilassamento Lagrangiano (P_y) è ancora un problema di *PLI* e quindi non parrebbe avere un duale, il che sembrerebbe non permettere di estendere il ragionamento. Questo però è invece possibile, utilizzando i concetti sviluppati nel §4.2; a tal fine conviene esprimere il problema originale come

$$(P) \qquad \max\{\, cx \,:\, Ax \leq b \,,\; x \in X \,\} \tag{6.13}$$

dove $X = \{\, x \in \mathbb{Z}^n \,:\, Ex \leq d \,\}$. Come abbiamo visto nel §4.2.2, massimizzare una funzione lineare sull'insieme discreto X è equivalente a farlo sul suo *inviluppo convesso* $conv(\, X \,)$, ossia

$$z(P_y) = \max\{\, cx + y(\, b - Ax \,) \,:\, x \in conv(\, X \,) \,\}$$

per ogni $y \in \mathbb{R}^m_+$. Si può dimostrare che se la matrice E ed il vettore d hanno tutte componenti razionali, allora $conv(\, X \,)$ è un poliedro convesso, ossia esiste un insieme finito di vincoli lineari tali che

$$conv(\, X \,) = \{\, x \,:\, \tilde{E}x \leq \tilde{d} \,\} \;.$$

Definiamo quindi il *rilassamento convessificato* di (P) come

$$(\tilde{P}) \qquad \max\{\, cx \,:\, Ax \le b \,,\; x \in conv(\, X \,)\,\} \;. \tag{6.14}$$

$(\tilde{P})$ è quindi un problema di *PL*, per quanto in generale la descrizione poliedrale di $conv(X)$ non sia nota. È però a questo punto facile estendere il Teorema 6.1:

Teorema 6.2. $z(D) = z(\tilde{P})$.

Dimostrazione Dalla definizione si ha

$$z(\tilde{P}) = \max\{\, cx \,:\, Ax \le b \,,\; \tilde{E}x \le \tilde{d}\,\} \;.$$

Dalla dualità lineare si ottiene quindi

$$z(\tilde{P}) = \min\{\, yb + w\tilde{d} \,:\, yA + w\tilde{E} = c \,,\; y \ge 0 \,,\; w \ge 0\,\}$$

che può essere riscritto come

$$\min\{\, yb + \min\{\, w\tilde{d} \,:\, w\tilde{E} = c - yA \,,\; w \ge 0\,\} \,:\, y \ge 0\,\} \;.$$

Ancora per dualità lineare, applicata al problema interno, si ha

$$z(\tilde{P}) = \min\{\, yb + \max\{\, (\, c - yA\,)x \,:\, \tilde{E}x \le \tilde{d}\,\}\,\}$$

e quindi il teorema è dimostrato. ◇

Il Teorema 6.2 ha le seguenti importanti conseguenze.

Corollario 6.1. Il duale Lagrangiano fornisce una valutazione superiore non peggiore (non maggiore) di quella fornita dal rilassamento continuo, ossia $z(D) \le z(\bar{P})$; in più, se i vincoli $Ex \le d$ hanno la *proprietà di integralità* allora si ha che $z(D) = z(\tilde{P}) = z(\bar{P})$.

Dimostrazione La regione ammissibile di $(\tilde{P})$ è contenuta in quella di $(\bar{P})$, e le funzioni obiettivo coincidono. La proprietà di integralità (Definizione 4.1) è equivalente a $conv(\, X \,) = \{\, x \,:\, Ex \le d\,\}$, ossia $\tilde{E} = E$ e $\tilde{d} = d$. ◇

Il corollario precedente mostra che si ha un "principio di conservazione della difficoltà": se il rilassamento Lagrangiano è "facile", ossia il vincolo di integralità è soddisfatto automaticamente, come avviene ad esempio nei problemi di flusso, allora il duale Lagrangiano è equivalente al rilassamento continuo. Per ottenere una valutazione superiore strettamente migliore è necessario che il sottoproblema Lagrangiano sia "difficile", ossia che i vincoli $Ex \le d$ non forniscano una descrizione "esatta" di $conv(\, X \,)$.

Le osservazioni precedenti consentono in alcuni casi di valutare le prestazioni relative di rilassamenti Lagrangiani diversi dello stesso problema. Ad esempio, si consideri il duale Lagrangiano di (P) rispetto al secondo blocco di vincoli

$$(D') \qquad \min\{\, \max\{\, cx + w(\, d - Ex\,) \,:\, x \in X'\,\} \,:\, w \ge 0\,\} \;.$$

dove $X' = \{\, x \in \mathbb{Z}^n \,:\, Ax \le b\,\}$. Se sia i vincoli $Ax \le b$ che i vincoli $Ex \le d$ posseggono la proprietà di integralità allora $z(D') = z(D) = z(\bar{P})$; se invece i vincoli $Ex \le d$ posseggono la proprietà di integralità ed i vincoli $Ax \le b$ non la posseggono allora $z(D') \le z(D) = z(\bar{P})$, e la diseguaglianza può essere stretta.

Esempio 6.8. Rilassamenti Lagrangiani di (UMMCF)

Il rilassamento Lagrangiano di (UMMCF) rispetto ai vincoli di capacità (6.5) si decompone in $|\, K \,|$ problemi di cammino minimo, uno per ciascuna coppia $(\, o_h \,,\, d_h\,)$, rispetto ai costi Lagrangiani $c_{ij} - y_{ij}$ (c'è un moltiplicatore Lagrangiano y_{ij} per ciascun vincolo di capacità, e quindi per ciascun arco). Siccome i problemi di cammino minimo hanno la proprietà di integralità, il corrispondente duale Lagrangiano fornisce esattamente la stessa valutazione inferiore del rilassamento continuo. Invece, il rilassamento Lagrangiano rispetto ai vincoli di conservazione di flusso (6.4) ha la forma

$$\begin{array}{lll} \min & \sum_{h \in K} \sum_{(\, i \,,\, j\,) \in A} (\delta_h c_{ij} - w_i^h + w_j^h) x_{ij}^h & \\ & \sum_{h \in K} \delta_h x_{ij}^h \le u_{ij} & (\, i \,,\, j\,) \in A \\ & x_{ij}^h \in \{\, 0 \,,\, 1\,\} & (\, i \,,\, j\,) \in A \;,\; h \in K \end{array}$$

dato che esiste un moltiplicatore Lagrangiano w_i^h per la copia corrispondente alla commodity h del vincolo di conservazione di flusso relativo al nodo i. Il rilassamento si decompone in $|\, A \,|$ problemi indipendenti, uno per ogni

arco; quando $w = 0$ questi problemi hanno soluzione ottima identicamente nulla, ma per valori diversi del vettore dei moltiplicatori Lagrangiani alcuni dei costi Lagrangiani $\delta_h c_{ij} - w_i^h + w_j^h$ possono diventare negativi. Quindi, in generale, il rilassamento Lagrangiano richiede la soluzione di $|A|$ problemi di zaino, ciascuno con al più $|K|$ variabili (solo gli oggetti con costo Lagrangiano negativo possono essere inseriti nello zaino in una soluzione ottima), e quindi non possiede la proprietà di integralità. Di conseguenza, il corrispondente duale Lagrangiano può fornire una valutazione inferiore di $z(P)$ strettamente migliore di quella del rilassamento continuo.

Esempio 6.9. Rilassamenti Lagrangiani di (TSP)
Nel caso di (TSP), invece, sia il rilassamento Lagrangiano rispetto ai vincoli di connessione che quello rispetto ai vincoli di copertura per cicli posseggono la proprietà di integralità; di conseguenza i corrispondenti duali Lagrangiani forniscono la stessa valutazione inferiore, che è anche la stessa fornita dal rilassamento continuo. Si noti che, in questo caso, risolvere direttamente il rilassamento continuo non è banale dato il numero esponenziale di vincoli della formulazione, ma è comunque possibile con le tecniche viste nel §4.2.3 (cf. in particolare l'Esempio 4.6). Comunque, anche risolvere un duale Lagrangiano non è in principio banale; alcune tecniche risolutive saranno discusse nel prossimo paragrafo.

Esercizio 6.13. Si discutano le relazioni tra la valutazione fornita dal rilassamento continuo e quella fornita da tutti i possibili duali Lagrangiani per i problemi (KP), (MMMS), (CMST) e (CSP).

Negli esempi precedenti si è sempre dato il caso in cui una delle due alternative sia la migliore, oppure che siano tutte equivalenti. È facile vedere che non sempre questo è il caso.

Esempio 6.10. Il problema del doppio zaino
Un ovvio esempio è il problema "del doppio zaino"

$$(2K) \quad \max\{ \sum_{i\in I} c_i x_i \,:\, \sum_{i\in I} l_i x_i \le L \,,\, \sum_{i\in I} w_i x_i \le W \,,\, x \in \{0\,,\,1\}^n \}$$

in cui ogni oggetto ha una "lunghezza" l_i ed una "larghezza" w_i e si vuole che sia la lunghezza totale che la larghezza totale siano minori di costanti date; questo è, tra le altre cose, un'approssimazione dei problemi di *zaino bidimensionale* in cui ciascun oggetto è un rettangolo $l_i \times w_i$ che va inserito in (o ritagliato da) un rettangolo $L \times W$.

Esercizio 6.14. Si dimostri con un esempio come mai il doppio zaino non sia equivalente allo zaino bidimensionale.

È evidente che (2K) ammette due diversi rilassamenti Lagrangiani, uno per ciascuno dei due diversi vincoli di zaino. È ovvio come nessuno dei due ammetta la proprietà di integralità, e che quindi ciascuno dei due corrispondenti duali Lagrangiani possa dare una valutazione superiore migliore del rilassamento continuo. Il problema è però quello di determinare quale dei due rilassamenti (che computazionalmente appaiono uguali in termini di costo di soluzione) produca la valutazione superiore migliore. In generale non ci si può aspettare che le due valutazioni siano sempre equivalenti, ma non è ovvio sapere a priori, senza provarlo sperimentalmente, quale delle due sia preferibile in termini della valutazione superiore prodotta. La teoria sviluppata finora non è in grado, almeno direttamente, di fornire indicazioni rispetto alla migliore scelta tra i due diversi rilassamenti Lagrangiani.

L'esempio mostra che ci sono casi in cui sono disponibili rilassamenti Lagrangiani diversi tra i (corrispondenti duali Lagrangiani dei) quali non è facile scegliere. Esiste però un ingegnoso trucco modellistico che sostanzialmente permette di non scegliere, ottenendo sempre una valutazione superiore non peggiore (e possibilmente strettamente migliore) delle due valutazioni separatamente ottenibili con il normale rilassamento Lagrangiano. Questa è la cosiddetta *decomposizione Lagrangiana*, che si ottiene riformulando il problema nella forma equivalente (ma a prima vista apparentemente assurda)

$$(P) \quad \max\{ c(\,x + x'\,)/2 \,:\, Ax' \le b \,,\, x' \in \mathbb{Z}^n \,,\, Ex \le d \,,\, x \in \mathbb{Z}^n \,,\, x = x' \}$$

in cui si crea una copia delle variabili e si esprime metà dei vincoli del problema con riferimento alla copia, salvo poi imporre che le due copie siano identiche. La motivazione per questo approccio è che esso crea un nuovo insieme di vincoli, quelli appunto di copia $x = x'$, rispetto ai quali è possibile quindi fare il rilassamento Lagrangiano. Per il duale Lagrangiano rispetto a tali vincoli, ossia

$$(D'') \quad \min\{ \max\{ (\,c/2 - w\,)x \,:\, x \in X \} + \max\{ (\,c/2 + w\,)x' \,:\, x' \in X' \} \} \,.$$

il Teorema 6.2 mostra un importante risultato:

Corollario 6.2. $z(D'') = \max\{ cx \,:\, x \in conv(\,X\,) \cap conv(\,X'\,) \}$, e quindi $z(D'') \le \min\{ z(D')\,,\, z(D) \}$ $\le z(\bar{P})$.

Esercizio 6.15. Dimostrare il Corollario precedente.

In altri termini, la decomposizione Lagrangiana fornisce una valutazione superiore di $z(P)$ non peggiore (non maggiore) di quella fornita da ciascuno dei due rilassamenti Lagrangiani, e quindi non peggiore di quella fornita dal rilassamento continuo. In particolare, è facile verificare che $z(D'')$ coincide con $z(\bar{P})$ se sia i vincoli $Ax \leq b$ che i vincoli $Ex \leq d$ posseggono la proprietà di integralità, che $z(D'')$ coincide con la valutazione fornita dal migliore dei due rilassamenti Lagrangiani se uno solo dei due insiemi di vincoli possiede la proprietà di integralità, e che $z(D'')$ può essere strettamente minore di $\min\{\, z(D')\,,\, z(D)\,\}$ se nessuno dei due insiemi di vincoli possiede la proprietà di integralità.

Esercizio 6.16. Dimostrare le affermazioni precedenti (suggerimento: per l'ultima affermazione consultare l'esempio seguente).

Esempio 6.11. Rilassamenti Lagrangiani e decomposizione Lagrangiana
Si consideri il problema di *PLI*

$$(P) \qquad \max\{\, 3x_1 + x_2 \,:\, x_1 + x_2 \leq 3/2\,,\; x_1 \leq 1/2\,,\; [\,x_1\,,\,x_2\,] \in \mathbb{N}^2\,\}\ .$$

Indichiamo con $Ax \leq b$ il primo vincolo ($x_1 + x_2 \leq 3/2$) e con $Ex \leq d$ il secondo vincolo ($x_1 \leq 1/2$). Si noti che sono presenti, nella formulazione del problema, anche i vincoli $x_1 \geq 0$ ed $x_2 \geq 0$, che non verranno mai rilassati, e quindi che verranno sempre implicitamente considerati come facenti parte dell'"altro" blocco di vincoli, così come quelli di integralità.

Il problema è illustrato geometricamente in Figura 6.4(a); in particolare, nella figura sono evidenziati i vincoli lineari, l'insieme delle soluzioni ammissibili del problema (i punti $[\,0\,,\,0\,]$ e $[\,0\,,\,1\,]$) e l'insieme ammissibile del rilassamento continuo (zona ombreggiata). Dalla figura si deduce facilmente che la soluzione ottima del problema è $[\,0\,,\,1\,]$, e quindi che $z(P) = 1$, mentre la soluzione ottima del rilassamento continuo ($\bar{P}$) è $[\,1/2\,,\,1\,]$, e quindi $z(\bar{P}) = 5/2$. Consideriamo adesso il rilassamento Lagrangiano rispetto al primo vincolo ($Ax \leq b$)

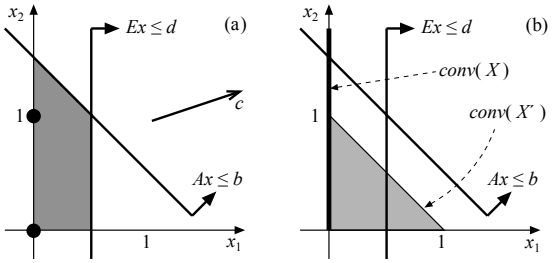


Figura 6.4: Un esempio di rilassamento Lagrangiano (1)

$$\max\{\, (\,3 - y\,)x_1 + (\,1 - y\,)x_2 \,:\, x_1 \leq 1/2\,,\; [\,x_1\,,\,x_2\,] \in \mathbb{N}^2\,\} + (3/2)y\ ,$$

ed il corrispondente duale Lagrangiano. Dal Teorema 6.2 abbiamo che il duale Lagrangiano fornisce la stessa valutazione superiore del rilassamento convessificato, la cui regione ammissibile è evidenziata in Figura 6.5(a). Infatti, l'insieme ammissibile X del rilassamento Lagrangiano contiene tutti i punti $[\,0\,,\,x_2\,]$ con $x_2 \in \mathbb{N}$, e quindi $conv(\,X\,)$ è il semiasse positivo di x_2 (si veda la Figura 6.4(b)); la sua intersezione con $Ax \leq b$ restituisce il segmento di estremi $[\,0\,,\,0\,]$ e $[\,0\,,\,3/2\,]$. La soluzione ottima del rilassamento convessificato è quindi il punto $[\,0\,,\,3/2\,]$; ci attendiamo pertanto che sia $z(D) = 3/2$. Possiamo verificare che ciò sia vero disegnando la *funzione Lagrangiana* $\varphi(\,y\,) = z(P_y)$ per tutti i valori di $y \geq 0$; a tale scopo, in Figura 6.5(b) vengono mostrati i vettori dei costi Lagrangiani $c_y = [\,3 - y\,,\,1 - y\,]$ corrispondenti ai valori $y = 0$, $y = 1$, $y = 2$, $y = 3$, ed il limite del vettore dei costi Lagrangiani per $y \to \infty$. Si ha che:

- il rilassamento Lagrangiano è superiormente illimitato ($z(P_y) = +\infty$) per $y < 1$;
- tutte le soluzioni ammissibili del rilassamento Lagrangiano sono ottime per $y = 1$, avendo tutte costo Lagrangiano $3/2$;
- $[\,0\,,\,0\,]$ è l'unica soluzione ottima del rilassamento Lagrangiano per $y > 1$, con costo Lagrangiano $(3/2)y$.

La funzione Lagrangiana è quindi quella rappresentata in Figura 6.5(c) (nella zona ombreggiata la funzione ha valore $+\infty$), ed ha minimo per $y = 1$ con $\varphi(\,1\,) = 3/2 = z(D)$.

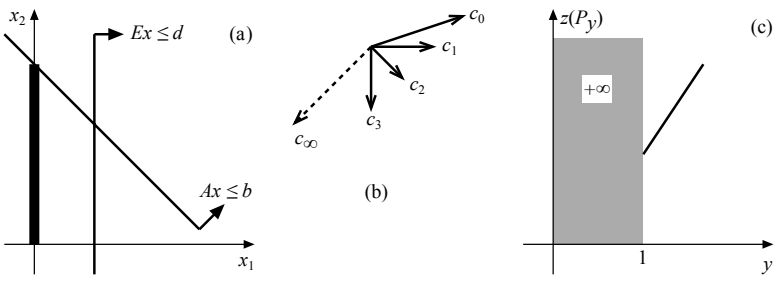


Figura 6.5: Un esempio di rilassamento Lagrangiano (2)

Consideriamo ora il rilassamento Lagrangiano rispetto al secondo vincolo ($Ex \leq d$)

$$\max\{\,(3-y)x_1 + x_2 \,:\, x_1 + x_2 \leq 3/2\,,\ [\,x_1\,,\,x_2\,] \in \mathbb{N}^2\,\} + (1/2)y\ \ ,$$

ed il corrispondente duale Lagrangiano. L'insieme ammissibile X' del rilassamento Lagrangiano contiene i punti $[\,0\,,\,0\,]$, $[\,0\,,\,1\,]$ e $[\,1\,,\,0\,]$, e quindi $conv(\,X'\,)$ è il triangolo avente quei punti come vertici; la sua intersezione con $Ex \leq d$ restituisce la regione ammissibile del corrispondente rilassamento convessificato, mostrata in Figura 6.6(a) (zona in ombreggiata più scura). La soluzione ottima del rilassamento convessificato è quindi il punto $[\,1/2\,,\,1/2\,]$; ci attendiamo pertanto che sia $z(D) = 2$. Possiamo verificare che ciò sia vero disegnando la funzione Lagrangiana $\varphi(\,y\,)$ per tutti i valori di $y \geq 0$; a tale scopo, in Figura 6.6(b) vengono mostrati i vettori dei costi Lagrangiani $c_y = [\,3-y\,,\,1\,]$ corrispondenti ai valori $y = 0$, $y = 2$, $y = 3$, $y = 4$, ed il limite del vettore dei costi Lagrangiani per $y \to \infty$. Si ha che:

- per $0 \leq y \leq 2$, il punto $[\,1\,,\,0\,]$ è soluzione ottima del rilassamento Lagrangiano (unica se $y < 2$), con costo Lagrangiano $3 - y/2$;
- per $y \geq 2$, il punto $[\,0\,,\,1\,]$ è soluzione ottima del rilassamento Lagrangiano (unica se $y > 2$), con costo Lagrangiano $2 + y/2$.

La funzione Lagrangiana è quindi quella rappresentata in Figura 6.6(c), ed ha minimo per $y = 2$ con $\varphi(\,2\,) = 2 = z(D')$.

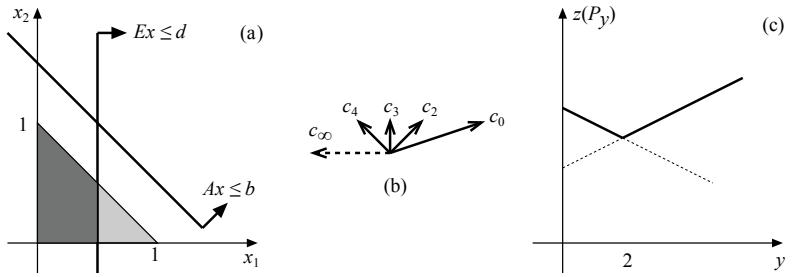


Figura 6.6: Un esempio di rilassamento Lagrangiano (3)

Consideriamo infine la decomposizione Lagrangiana di (P) corrispondente ad entrambi i blocchi $Ax \leq b$ ed $Ex \leq d$, ossia

$$\begin{array}{c} \max\{\,(3/2-y_1)x_1 + (1/2-y_2)x_2 \,:\, x_1 \leq 1/2\,,\ [\,x_1\,,\,x_2\,] \in \mathbb{N}^2\,\} \\ + \\ \max\{\,(3/2+y_1)x_1 + (1/2+y_2)x_2 \,:\, x_1 + x_2 \leq 3/2\,,\ [\,x_1\,,\,x_2\,] \in \mathbb{N}^2\,\} \end{array} .$$

L'insieme ammissibile del corrispondente rilassamento convessificato è l'intersezione tra $conv(\,X\,)$ e $conv(\,X'\,)$, ossia il segmento di estremi $[\,0\,,\,0\,]$ e $[\,0\,,\,1\,]$; pertanto la sua soluzione ottima è $[\,0\,,\,1\,]$ e si ha $z(D'') = z(P) = 1$. Per verificare che ciò sia vero si consideri il vettore di moltiplicatori Lagrangiani $[\,\bar{y}_1\,,\,\bar{y}_2\,] = [\,-1/2\,,\,1/2\,]$. In corrispondenza a tali moltiplicatori, il primo problema della decomposizione Lagrangiana ha soluzione ottima $[\,0\,,\,0\,]$ ed il secondo ha soluzione ottima $[\,1\,,\,0\,]$, da cui $\varphi(\,\bar{y}\,) = 1$. Poichè $1 \leq z(P) \leq z(D'') \leq \varphi(\,\bar{y}\,) = 1$, $\bar{y}$ è una soluzione ottima del corrispondente duale Lagrangiano, che ha effettivamente gap nullo.

Esercizio 6.17. Si caratterizzi la funzione Lagrangiana della decomposizione Lagrangiana nell'ultimo caso dell'esempio precedente, disegnandone le curve di livello.

Esercizio 6.18. Costruire un esempio di problema *PLI* con due variabili e due vincoli in cui la decomposizione Lagrangiana fornisca una valutazione superiore non esatta (con gap non nullo) del valore ottimo della funzione obiettivo.

È però ovvio che la decomposizione Lagrangiana richiede, per fissati moltiplicatori Lagrangiani, la soluzione di *due* problemi combinatori invece che uno solo; inoltre, affinché la valutazione superiore possa essere strettamente migliore (della migliore) di quelle dei due rilassamenti, entrambi i due problemi devono essere "difficili". Non è quindi scontato che la decomposizione Lagrangiana sia computazionalmente preferibile ai rilassamenti, anche qualora produca valutazioni superiori migliori. Questo dipende anche dallo sforzo necessario per risolvere il duale Lagrangiano, che può essere molto diverso. Nel caso del doppio zaino, ad esempio, ciascuno dei due rilassamenti ha un solo moltiplicatore Lagrangiano, mentre la decomposizione ne ha n; è intuitivo che questo possa rendere il corrispondente duale più costoso da risolvere, anche senza tenere in conto il diverso costo di calcolare i rilassamenti. Per sostanziare questa discussione occorre però discutere come sia possibile risolvere un duale Lagrangiano.

6.3.2 Algoritmi per il rilassamento Lagrangiano

Per semplificare la trattazione assumeremo temporaneamente che l'insieme X sia *compatto*; nel seguito indicheremo poi come l'assunzione possa essere eliminata al costo di complicare leggermente la notazione e gli algoritmi. La conseguenza di questa assunzione è che la funzione Lagrangiana

$$\varphi(\,y\,) = z(P_y) = \max\{\, cx + y(\,b - Ax\,) \,:\, x \in X\,\}$$

è finita ovunque (cosa che altrimenti può non accadere, si veda ad esempio la Figura 6.5(c)). Essendo il massimo di (un numero finito di) funzioni lineari, φ è convessa (sarebbe concava se (P) fosse un problema di minimo). Dato che X è un insieme discreto, come abbiamo visto negli esempi, φ è una funzione lineare a tratti quando $m = 1$; in generale φ è *poliedrale*. Infatti, il suo *epigrafo*

$$Epi(\,\varphi\,) = \{\,(\,z\,,\,y\,) \,:\, z \geq \varphi(\,y\,)\,\} = \{\,(\,z\,,\,y\,) \,:\, z \geq cx + y(\,b - Ax\,)\,,\; x \in X\,\}$$

è un poliedro; ad ogni elemento x di X che risolve (P_y) per un qualche y è associato un vincolo lineare che definisce una delle facce di $Epi(\,\varphi\,)$. È quindi possibile riscrivere (D) come un problema di PL; questo non dovrebbe stupire, in quanto (D) è il duale lineare di $(\tilde{P})$. Infatti, un modo equivalente di formulare (D) è

$$(D) \qquad \min\{\, z \,:\, z \geq cx + y(\,b - Ax\,)\;\; x \in X\,,\; y \geq 0\,\}\;. \tag{6.15}$$

In (6.15), ciascun elemento di X definisce un vincolo del problema, ossia possibilmente definisce una faccia della regione ammissibile, che altro non è che $Epi(\,\varphi\,)$. Si noti quindi che "$x \in X$" in quel contesto non indica un vincolo sulle variabili del problema, che sono z e y; piuttosto, x rappresenta l'indice dell'insieme dei vincoli. Ciò indica immediatamente che il fatto che sia possibile scrivere (D) come un problema di PL non implica che (D) sia di "facile" risoluzione: il numero di vincoli del problema, corrispondente al numero di elementi di X, può essere enorme. È però vero che non tutti i vincoli di (D) sono necessari per la determinazione della soluzione ottima; al limite, sarebbero sufficienti gli $m+1$ vincoli corrispondenti ad una base ottima del problema. Ciò suggerisce un approccio di *generazione di vincoli*, ossia sostanzialmente ancora una volta il *Simplesso Duale Meccanizzato* (cf. Algoritmo 4.2.3) che è stato discusso nel §4.2.3. È però importante notare che l'algoritmo opera adesso nello spazio $[\,z\,,\,y\,]$ del duale. Adattato a questo contesto si ottiene il seguente *algoritmo dei piani di taglio* (di Kelley):

```
procedure y* = Cutting_Planes( A , b , c , X ) {
    ⟨ inizializza B ⟩
    do { ( z* , y* ) = argmin{ z : z ≥ cx + y( b − Ax )  x ∈ B , y ≥ 0 };   // (D_B)
         x̄ = argmax{ ( c − y*A )x : x ∈ X };                                // (P_y*)
         φ( y* ) = cx̄ + y*( b − Ax̄ ); B = B ∪ { x̄ };
         } while( z* < φ( y* ) );
    }
```

Procedura 6.2: Algoritmo dei Piani di Taglio

Ad ogni iterazione si risolve il *Problema Master* (Duale)

$$(D_{\mathcal{B}}) \qquad \min\{\, z \,:\, z \geq cx + y(\,b - Ax\,)\;\; x \in \mathcal{B}\,,\; y \geq 0\,\} \tag{6.16}$$

ove $\mathcal{B} \subset X$ è un "piccolo" sottoinsieme delle soluzioni ammissibili di (P_y); in altri termini si considera solamente un "piccolo" sottoinsieme dei vincoli di (D) (ancora, "$x \in \mathcal{B}$" denota un indice, non un vincolo). Ciò corrisponde a risolvere

$$\min\{\, \varphi_{\mathcal{B}}(\,y\,) = \max\{\, cx + y(\,b - Ax\,) \,:\, x \in \mathcal{B}\,\}\,\}\;\;,$$

ossia a minimizzare la funzione convessa poliedrale $\varphi_{\mathcal{B}}$ tale che $\varphi_{\mathcal{B}}(\,y\,) \leq \varphi(\,y\,)$ comunque scelto $y \geq 0$, al posto di φ; si noti che nel problema di calcolare $\varphi_{\mathcal{B}}$ (per fissato y) "$x \in \mathcal{B}$" denota invece il vincolo sulle variabili del problema. $\varphi_{\mathcal{B}}$ è detta *modello* di φ, in quanto ne rappresenta un'approssimazione "semplificata". La soluzione ottima $(\,z^*\,,\,y^*\,)$ di $(D_{\mathcal{B}})$, dove $z^* = \varphi_{\mathcal{B}}(\,y^*\,)$, può quindi essere usata per generare un ulteriore vincolo, se necessario, semplicemente risolvendo il rilassamento Lagrangiano (P_{y^*}), ossia calcolando $\varphi(\,y^*\,)$; in questo contesto (P_{y^*}) è il *problema (oracolo) di separazione*. Se infatti si ha $z^* < \varphi(\,y^*\,)$, allora una qualsiasi soluzione ottima $\bar{x}$ di (P_{y^*}) fornisce un vincolo di (D)

violato dalla soluzione corrente (z^* , y^*), che può quindi essere aggiunto a $\mathcal{B}$. Altrimenti, ed è facile verificare che in questo caso risulta $z^* = \varphi(y^*)$, (z^* , y^*) rispetta tutti i vincoli in (D), anche quelli non esplicitamente rappresentati in $\mathcal{B}$, e quindi è ottima per (D).

L'algoritmo determina quindi ad ogni passo una valutazione inferiore ed una superiore di $z(D)$, in quanto $z^* \leq z(D) \leq \varphi(y^*)$, e termina in un numero finito di passi (al limite $\mathcal{B} = X$) quando le due coincidono. Occorre solamente assicurarsi che l'insieme di vincoli $\mathcal{B}$ determinato dalla fase di inizializzazione sia sufficiente ad assicurare che $(D_\mathcal{B})$ abbia soluzione ottima finita; un modo in cui questo può essere ottenuto è ponendo $\mathcal{B} = \{ \hat{x} \}$, dove $\hat{x}$ è una soluzione ammissibile per (P) tale che $A\hat{x} = b$, il che corrisponde ad inserire in $(D_\mathcal{B})$ il vincolo $z \geq c\hat{x}$.

L'algoritmo dei piani di taglio può essere "rivisitato" in notazione primale, nel qual caso prende il nome di *metodo di decomposizione di Dantzig-Wolfe*. Per questo occorre notare che $(D_\mathcal{B})$ ha un duale (lineare), il *Problema Master Primale*

$$(P_\mathcal{B}) \qquad \max \left\{ \; c \left(\textstyle\sum_{x \in \mathcal{B}} x\theta_x \right) \; : \; A \left(\textstyle\sum_{x \in \mathcal{B}} x\theta_x \right) \leq b \;\; , \;\; \theta \in \Theta \; \right\} \tag{6.17}$$

dove $\Theta = \{ \theta \geq 0 \; : \; \sum_{x \in \mathcal{B}} \theta_x = 1 \}$ è il simplesso unitario di dimensione opportuna. $(P_\mathcal{B})$ ha una variabile per ogni riga di $(D_\mathcal{B})$, ossia per ciascun elemento di $\mathcal{B}$. È interessante notare che questa "forma esplicita" del problema è equivalente alla "forma implicita"

$$(P_\mathcal{B}) \qquad \max\{ cx \; : \; Ax \leq b \;\; , \;\; x \in X_\mathcal{B} = conv(\mathcal{B}) \} \;\; . \tag{6.18}$$

Questo chiarisce la relazione tra l'algoritmo dei piani di taglio e $(\tilde{P})$; infatti, (6.17) con $\mathcal{B} = X$ è una formulazione di $(\tilde{P})$, in cui sono esplicitamente rappresentati i moltiplicatori convessi θ. In particolare, (6.17) con $\mathcal{B} = X$ è il duale lineare di (6.15), come anticipato dal Teorema 6.2; solamente, in questo caso $conv(X)$ è espresso mediante una rappresentazione *per punti*, piuttosto che mediante la più usuale rappresentazione *per facce* utilizzata nel teorema.

L'algoritmo dei piani di taglio può quindi essere "rivisitato" dal punto di vista primale, notando che la "forma esplicita" di $(\tilde{P})$ è un problema di *PL* con "molte" colonne, una per ciascun elemento di X. Questo suggerisce un approccio di *generazione di colonne* nel quale si risolve la restrizione di $(\tilde{P})$ rispetto al sottoinsieme di colonne $\mathcal{B}$, ottenendo una soluzione ottima θ^* a cui corrisponde la soluzione $x^* = \sum_{x \in \mathcal{B}} x\theta^*_x$ ammissibile per $(\tilde{P})$ (la soluzione ottima della "forma implicita" di $(P_\mathcal{B})$). Inoltre, dalla soluzione del Problema Master primale si ottiene il vettore di variabili duali ottime y^* dei vincoli $Ax \leq b$, le quali determinano il *costo ridotto* $(c - yA)x$ (si veda il §6.1.2.2) della variabile θ_x, ossia della colonna corrispondente. Dai vincoli del Problema Master duale si ha che $z^* - y^*b \geq (c - y^*A)x$, e le condizioni degli scarti complementari garantiscono che si abbia uguaglainza per ogni $x \in \mathcal{B}$ tale che $\theta_x > 0$. Si vuole quindi determinare se esiste oppure no una colonna $x \in X \setminus \mathcal{B}$ il cui costo ridotto sia maggiore di $z^* - y^*b$; questo viene fatto risolvendo (P_{y^*}), che determina la colonna $\bar{x}$ di costo ridotto massimo, e che, in questo contesto, viene detto *problema di pricing* (come nel Simplesso Duale Meccanizzato deve determinare la variabile entrante, ma in questo caso è una variabile primale θ_x). Se il costo ridotto massimo tra le colonne (corrispondenti alle soluzioni) di X è maggiore di $z^* - y^*b$, allora è stata generata una colonna "promettente", che può essere inserita in $\mathcal{B}$ per migliorare la soluzione x^*; altrimenti x^* è ottima per $(\tilde{P})$. Questi passi sono *esattamente* quelli effettuati dall'Algoritmo 6.3.2, in cui x^* non è stata esplicitata. Quindi l'algoritmo dei piani di taglio fornisce, al termine, una soluzione x^* ottima per $(\tilde{P})$.

Esercizio 6.19. Nell'algoritmo dei piani di taglio (metodo di decomposizione di Dantzig-Wolfe) vengono risolti una sequenza di problemi di *PL* correlati tra loro. Facendo riferimento al §3.3.4, si discuta quale algoritmo del Simplesso sia più conveniente utilizzare per la loro risoluzione.

Esempio 6.12. L'algoritmo dei piani di taglio
Si consideri il problema di *PLI*

$$(P) \qquad \max\{ x_1 + x_2 \; : \; x_2 \leq 1/2 \; , \; [x_1 , x_2] \in \{ 0 , 1 \}^2 \}$$

ed il suo rilassamento Lagrangiano rispetto al vincolo $x_2 \leq 1/2$

$$(P_y) \qquad \max\{\, x_1 + (1-y)x_2 \,:\, [\,x_1\,,\,x_2\,] \in \{\,0\,,\,1\,\}^2 \,\} + (1/2)y \ .$$

Applichiamo l'algoritmo dei piani di taglio a partire da $\mathcal{B} = \{\,[\,0\,,\,0\,]\,,\,[\,0\,,\,1\,]\,\}$. I Problemi Master duale e primale sono quindi

$$(D_{\mathcal{B}}) \quad \begin{cases} \min & z & & \\ & z & \geq & 0 + y(1/2) \\ & z & \geq & 1 - y(1/2) \\ & & & y \geq 0 \end{cases} \qquad (P_{\mathcal{B}}) \quad \begin{cases} \max & & \theta_{[\,0\,,\,1\,]} & & \\ & & \theta_{[\,0\,,\,1\,]} & \leq & 1/2 \\ & \theta_{[\,0\,,\,0\,]} & +\theta_{[\,0\,,\,1\,]} & = & 1 \\ & \theta_{[\,0\,,\,0\,]} & & \geq & 0 \\ & & \theta_{[\,0\,,\,1\,]} & \geq & 0 \end{cases} .$$

La soluzione ottima di $(D_{\mathcal{B}})$ è $y^* = 1$, $z^* = \varphi_{\mathcal{B}}(\,y^*\,) = 1/2$, a cui corrisponde un costo ridotto $c - y^*A = [\,1\,,\,0\,]$. La soluzione ottima di $(P_{\mathcal{B}})$ è $\theta^*_{[\,0\,,\,0\,]} = \theta^*_{[\,0\,,\,1\,]} = 1/2$, a cui corrisponde $x^* = [\,0\,,\,1/2\,]$. I punti $[\,1\,,\,0\,]$ e $[\,1\,,\,1\,]$ sono entrambi soluzioni ottime di (P_{y^*}); supponiamo che tra i due venga restituito $\bar{x} = [\,1\,,\,0\,]$. Siccome si ha $\varphi(\,y^*\,) = 3/2 > z^*$, l'algoritmo prosegue. Questa situazione è illustrata in Figura 6.7.

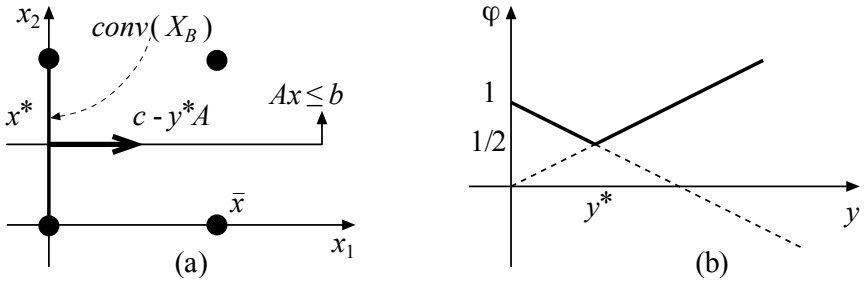


Figura 6.7: Applicazione dell'algoritmo dei piani di taglio (1)

Alla seconda iterazione si ha pertanto $\mathcal{B} = \{\,[\,0\,,\,0\,]\,,\,[\,0\,,\,1\,]\,,\,[\,1\,,\,0\,]\,\}$. I Problemi Master duale e primale sono quindi

$$(D_{\mathcal{B}}) \quad \begin{cases} \min & z & & \\ & z & \geq & 0 + y(1/2) \\ & z & \geq & 1 - y(1/2) \\ & z & \geq & 1 + y(1/2) \\ & & & y \geq 0 \end{cases} \qquad (P_{\mathcal{B}}) \quad \begin{cases} \max & & \theta_{[\,0\,,\,1\,]} & +\theta_{[\,1\,,\,0\,]} & & \\ & & \theta_{[\,0\,,\,1\,]} & & \leq & 1/2 \\ & \theta_{[\,0\,,\,0\,]} & +\theta_{[\,0\,,\,1\,]} & +\theta_{[\,1\,,\,0\,]} & = & 1 \\ & \theta_{[\,0\,,\,0\,]} & & & \geq & 0 \\ & & \theta_{[\,0\,,\,1\,]} & & \geq & 0 \\ & & & \theta_{[\,1\,,\,0\,]} & \geq & 0 \end{cases} .$$

La soluzione ottima di $(D_{\mathcal{B}})$ è $y^* = 0$, $z^* = \varphi_{\mathcal{B}}(\,y^*\,) = 1$, a cui corrisponde un costo ridotto $c - y^*A = [\,1\,,\,1\,]$. $(P_{\mathcal{B}})$ ha soluzioni ottime multiple, tra cui quelle estreme sono $\theta^*_{[\,0\,,\,0\,]} = 0$, $\theta^*_{[\,0\,,\,1\,]} = \theta^*_{[\,1\,,\,0\,]} = 1/2$ e $\theta^*_{[\,0\,,\,0\,]} = \theta^*_{[\,0\,,\,1\,]} = 0$, $\theta^*_{[\,1\,,\,0\,]} = 1$; queste soluzioni corrispondono a tutti i punti del segmento di estremi $[\,1/2\,,\,1/2\,]$ e $[\,1\,,\,0\,]$ nello spazio delle variabili x originarie. Il punto $\bar{x} = [\,1\,,\,1\,]$ è la soluzione ottima di (P_{y^*}). Siccome si ha $\varphi(\,y^*\,) = 2 > z^*$, l'algoritmo prosegue. Questa situazione è illustrata in Figura 6.8.

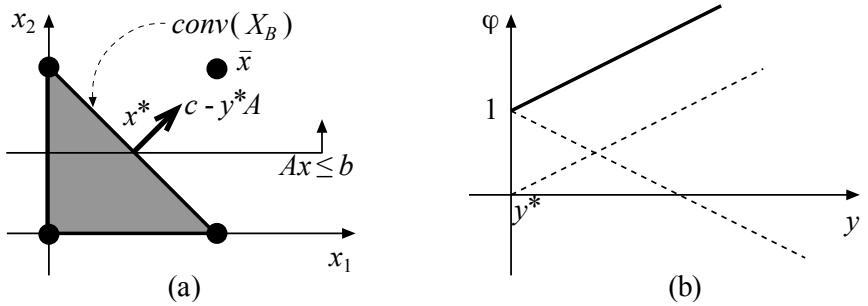


Figura 6.8: Applicazione dell'algoritmo dei piani di taglio (2)

Alla terza iterazione si ha dunque $\mathcal{B} = \{[0,0],[0,1],[1,0],[1,1]\} = X$. I Problemi Master duale (equivalente a (D)) e primale sono quindi

$$(D_{\mathcal{B}}) \quad \begin{cases} \min & z & & \\ & z & \geq & 0 + y(1/2) \\ & z & \geq & 1 - y(1/2) \\ & z & \geq & 1 + y(1/2) \\ & z & \geq & 2 - y(1/2) \\ & & & y \geq 0 \end{cases} \qquad (P_{\mathcal{B}}) \quad \begin{cases} \max & \theta_{[0,1]} & +\theta_{[1,0]} & +2\theta_{[1,1]} & & \\ & \theta_{[0,1]} & & +\theta_{[1,1]} & \leq & 1/2 \\ \theta_{[0,0]} & +\theta_{[0,1]} & +\theta_{[1,0]} & +\theta_{[1,1]} & = & 1 \\ \theta_{[0,0]} & & & & \geq & 0 \\ & \theta_{[0,1]} & & & \geq & 0 \\ & & \theta_{[1,0]} & & \geq & 0 \\ & & & \theta_{[1,1]} & \geq & 0 \end{cases}.$$

La soluzione ottima di $(D_{\mathcal{B}})$ è $y^* = 1$, $z^* = \varphi_{\mathcal{B}}(y^*) = 3/2$, a cui corrisponde un costo ridotto $c - y^*A = [1, 0]$. La soluzione ottima di $(P_{\mathcal{B}})$ è $\theta^*_{[0,0]} = \theta^*_{[0,1]} = 0$, $\theta^*_{[1,0]} = \theta^*_{[1,1]} = 1/2$, a cui corrisponde $x^* = [1, 1/2]$. I punti $[1, 0]$ e $[1, 1]$ sono entrambi soluzioni ottime di (P_{y^*}), e $\varphi(y^*) = 3/2 = z^*$: l'algoritmo quindi termina, avendo determinato la soluzione ottima $y^* = 1$ di (D) e la soluzione ottima $x^* = [1, 1/2]$ di $(\tilde{P})$. Questa situazione è illustrata in Figura 6.9.

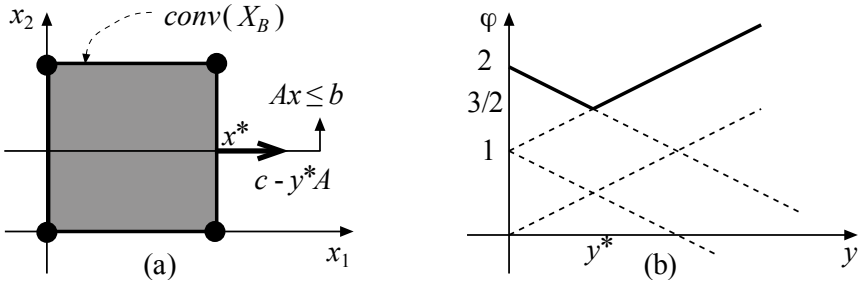


Figura 6.9: Applicazione dell'algoritmo dei piani di taglio (3)

Esercizio 6.20. Si discuta come sia possibile modificare l'algoritmo dei piani di taglio in modo che non sia necessario partire con un insieme di vincoli $\mathcal{B}$ sufficientemente grande da garantire che $(D_{\mathcal{B}})$ abbia ottimo finito (suggerimento: prendendo spunto dal Primale Ausiliario per il Simplesso Primale introdotto nel §3.3.1, si modifichi $(P_{\mathcal{B}})$ in modo tale che $(D_{\mathcal{B}})$ abbia sicuramente ottimo finito).

L'analisi precedente mostra che l'algoritmo dei piani di taglio risolve *contemporaneamente* (D) e $(\tilde{P})$, e suggerisce le seguenti considerazioni:

- Dal punto di vista primale, il Problema Master utilizza l'*approssimzaione interna* $X_{\mathcal{B}}$ di $conv(X)$, che viene espansa aggiungendo i punti generati dal problema di pricing (P_{y^*}) finché essa non contiene l'ottimo di $(\tilde{P})$. Dal punto di vista duale, il Problema Master utilizza l'*approssimazione esterna* $\varphi_{\mathcal{B}}$ di φ, e quindi della regione ammissibile di (D), che viene raffinata inserendo i vincoli generati dal problema di separazione finché non risulta "esatta" nell'ottimo di (D).
- La struttura del problema viene utilizzata per *generare efficientemente punti* (estremi) di $conv(X)$; in altri termini, questo procedimento è particolarmente attraente nel caso in cui ottenere una rappresentazione esplicita di $conv(X)$ sia significativamente più difficile che ottimizzare su X. In effetti, come mostrato nel paragrafo precedente, la soluzione del duale Lagrangiano di un problema di *PLI* è particolarmente attraente nel caso in cui i vincoli $Ex \leq d$ non rilassati *non* possiedano la proprietà di integralità, ossia non rappresentino esattamente $conv(X)$, ma si disponga comunque di un algoritmo "ragionevolmente efficiente" per risolvere il rilassamento Lagrangiano.

L'algoritmo dei piani di taglio può essere facilmente esteso al caso in cui X non sia compatto. Il poliedro $conv(X)$ può essere in generale espresso come la somma di un politopo P e di un cono finitamente generato C (si veda il §3.1.1), detto *cono delle direzioni* di $conv(X)$; quando X è compatto si ha $C = \{0\}$. Ogni vettore $\nu \in C$ costituisce una direzione ammissibile illimitata per $conv(X)$; ne consegue che se per un qualche y si ha $(c - yA)\nu > 0$, ossia la direzione è di crescita, allora il rilassamento Lagrangiano (P_y) è superiormente illimitato, ossia $\varphi(y) = +\infty$. In altri termini, a qualunque vettore $\nu \in C$ è associato un vincolo lineare $(c - yA)\nu \leq 0$ che è rispettato da tutti i punti y in cui $\varphi(y) < +\infty$. L'algoritmo dei piani di taglio si estende dunque al caso in cui X non sia compatto semplicemente mantenendo l'insieme $\mathcal{B} = \mathcal{B}^0 \cup \mathcal{B}^1$ in cui $\mathcal{B}^0 \subset C$ e $\mathcal{B}^1 \subset X$. Il Master

Problem Primale e Duale divengono rispettivamente

$$(P_{\mathcal{B}}) \quad \max \begin{cases} c\left(\sum_{x\in\mathcal{B}^1} x\theta_x + \sum_{\nu\in\mathcal{B}^0} \nu\theta_\nu\right) \\ A\left(\sum_{x\in\mathcal{B}^1} x\theta_x + \sum_{\nu\in\mathcal{B}^0} \nu\theta_\nu\right) \le b \\ \sum_{x\in\mathcal{B}^1} \theta_x = 1 \quad , \quad \theta \ge 0 \end{cases} \qquad (D_{\mathcal{B}}) \quad \min \begin{cases} yb + z \\ z \ge (c - yA)x \quad x \in \mathcal{B}^1 \\ 0 \ge (c - yA)\nu \quad \nu \in \mathcal{B}^0 \end{cases} .$$

Un modo equivalente di riscrivere i problemi è

$$(P_{\mathcal{B}}) \quad \max\{\, cx \,:\, Ax \le b \,,\, x \in conv(\mathcal{B}^1) + cono(\mathcal{B}^0)\,\} \qquad (D_{\mathcal{B}}) \quad \min\{\, \varphi_{\mathcal{B}^1}(y) \,:\, y \in Y_{\mathcal{B}}\,\} \,,$$

dove $Y_{\mathcal{B}} = \{\, y \,:\, (c - yA)\nu \le 0 \quad \nu \in \mathcal{B}^0 \,\}$ è un'approssimazione esterna dell'insieme dei punti y in cui $\varphi(y) < +\infty$. Ad ogni iterazione dell'algoritmo, la soluzione del rilassamento Lagrangiano riporta o una soluzione ottima $\bar{x} \in X$, che viene quindi aggiunta a $\mathcal{B}^1$, oppure una direzione di decrescita illimitata $\bar{\nu} \in C$, che viene quindi aggiunta a $\mathcal{B}^0$. Si noti che nel secondo caso si ha $\varphi(y^*) = +\infty$, e di conseguenza in questo tipo di iterazioni non si ha a disposizione una nuova valutazione superiore su $z(D)$.

Una diversa estensione dell'algoritmo dei piani di taglio si ha nel caso, molto frequente nelle applicazioni, in cui X è il prodotto cartesiano di k di insiemi (che assumiamo temporaneamente compatti) $X = X_1 \times X_2 \times \ldots \times X_k$, ossia il rilassamento Lagrangiano si decompone in k problemi indipendenti ed una soluzione ottima $\bar{x} = [\,\bar{x}^1\,,\,\bar{x}^2\,,\,\ldots\,,\,\bar{x}^k\,]$ del rilassamento Lagrangiano si ottiene semplicemente giustapponendo le soluzioni ottime dei k sottoproblemi; questo è ad esempio il caso di (UMMCF). In altri termini, la funzione Lagrangiana si decompone nella somma di k funzioni

$$\varphi(y) = yb + \sum_{h\in K} \left(\varphi^h(y) = \min\{\,(c^h - yA^h)x^h \,:\, x^h \in X^h\,\}\right)$$

($K = \{\,1\,,\,\ldots\,,\,k\,\}$, e si può pensare che la funzione lineare yb sia la $(k+1)$-esima funzione). Si può allora risolvere ad ogni iterazione, al posto di (6.17), il *Problema Master primale e duale disaggregato*

$$\begin{array}{ll} (P_{\mathcal{B}}) & \max\left\{\, \sum_{h\in K} c^h \sum_{x^h\in\mathcal{B}^h} x^h\theta^h_x \,:\, \sum_{h\in K} A^h \sum_{x^h\in\mathcal{B}^h} x^h\theta^h_x \le b \,,\, \theta^h \in \Theta \quad h \in K\,\right\} \\ (D_{\mathcal{B}}) & \min\left\{\, yb + \sum_{h\in K} z^h \,:\, z^h \ge (c^h - yA^h)x^h \quad x^h \in \mathcal{B}^h \quad h \in K\,\right\} \end{array} \tag{6.19}$$

in cui tutte le componenti h-esime delle soluzioni generate sono immagazzinate nell'insieme $\mathcal{B}^h$ ed hanno un moltiplicatore θ^h_x indipendente dalle altre componenti della stessa soluzione. I due problemi possono essere riscritti come

$$(P_{\mathcal{B}}) \quad \max \begin{cases} \sum_{h\in K} c^h x^h \\ \sum_{h\in K} A^h x^h \le b \\ x^h \in conv(\mathcal{B}^h) \quad h \in K \end{cases} \qquad (D_{\mathcal{B}}) \quad \min\left\{\, yb + \sum_{h\in K} \varphi_{\mathcal{B}^h}(y)\,\right\} \tag{6.20}$$

in cui $\varphi_{\mathcal{B}^h}$ è il modello dell'h-esima componente φ^h di φ. È facile verificare come, dato uno stesso insieme di soluzioni $\mathcal{B} \subset X$, l'insieme ammissibile di (6.20) contenga strettamente quello di (6.18); infatti, (6.17) è la restrizione di (6.19) in cui tutte le componenti x^h corrispondenti ad una stessa soluzione x sono forzate ad avere lo stesso moltiplicatore. In altri termini, $conv(\mathcal{B}^1) \times conv(\mathcal{B}^2) \times \ldots \times conv(\mathcal{B}^k)$ è una migliore approssimazione di $conv(X)$ rispetto a $conv(\mathcal{B})$; alternativamente, si può dire che la somma dei k modelli $\varphi_{\mathcal{B}^h}$ è una migliore approssimazione di φ rispetto al modello "aggregato" $\varphi_{\mathcal{B}}$. I Problemi Master disaggregati hanno dimensione maggiore di un fattore k rispetto a quelli aggregati (a parità di informazione raccolta), e sono quindi più costosi da risolvere; il miglior uso dell'informazione disponibile determina però spesso una convergenza sensibilmente più rapida dell'algoritmo dei piani di taglio (un minor numero di iterazioni), che può abbondantemente controbilanciare il maggior costo della soluzione del Problema Master. Nel caso poi in cui alcuni degli insiemi X^h non siano compatti l'algoritmo può essere esteso analogamente a quanto visto in precedenza.

Sono stati proposti molti altri algoritmi per la soluzione del duale Lagrangiano; alcuni sono basati sull'algoritmo dei piani di taglio e cercano di migliorarne le prestazioni evitando alcune delle sue limitazioni ("instabilità" e necessità di un opportuno insieme iniziale $\mathcal{B}$), mentre altri sono basati su idee diverse. Una classe di approcci che riscuote favore in ragione della sua semplicità è quella dei *metodi del subgradiente*, che sono basati sulla seguente osservazione: se $\bar{x}$ è una soluzione ottima di (P_y), il vettore $g = b - A\bar{x}$ (*subgradiente* della funzione Lagrangiana φ in y) "indica da che parte

la funzione φ cresce". Infatti, si consideri una qualsiasi direzione d ed i punti $y(\alpha) = y + \alpha d$, e si supponga che il passo $\alpha > 0$ sia scelto "sufficientemente piccolo" affinché la soluzione ottima di $(P_{y(\alpha)})$ sia ancora $\bar{x}$ (non è detto che questo sia possibile, ma è evidente che in molti casi può accadere in particolare se X è discreto): si ha quindi

$$\varphi(y(\alpha)) = c\bar{x} + y(\alpha)(b - A\bar{x}) = c\bar{x} + (y + \alpha d)(b - A\bar{x}) = \varphi(y) + \alpha d(b - A\bar{x}) \ .$$

Siamo interessati a scegliere d in modo tale che $\varphi(y(\alpha)) < \varphi(y)$; ciò richiede necessariamente $d(b - A\bar{x}) < 0$. A parità di α, vorremmo quindi scegliere d in modo tale che il prodotto scalare sia il più possibile negativo: ovviamente, la scelta ottima (almeno come direzione, ossia ignorando la norma) è $d = -(b - A\bar{x})$. Pertanto, ci si può attendere (anche se non è garantito) che un passo opportuno in direzione dell'*antigradiente* $d = -g$ porti ad una diminuzione del valore di φ.

Questa considerazione porta immediatamente alla definizione dell'approccio iterativo

$$y^{i+1} = y^i - \alpha^i g^i \quad \text{dove} \quad g^i = b - A\bar{x}^i \quad \text{e } \bar{x}^i \text{ è una soluzione ottima di } (P_{y^i}) \ .$$

La principale difficoltà di questi approcci è la scelta opportuna del passo α^i ad ogni iterazione, ma sono note regole che garantiscono che l'algoritmo determini (al limite per $i \to \infty$) approssimazioni arbitrariamente accurate di $z(D)$. Alcune regole sono straordinariamente semplici, come $\alpha^i = 1/i$, ma sono scarsamente efficienti in pratica. Una regola interessante è disponibile nel caso in cui $z(D)$ sia noto: in questo caso, per il *passo di Polyak*

$$\alpha^i = (\varphi(y^i) - z(D)) / \| g^i \|^2$$

è possibile dimostrare che la sequenza di iterate $\{ y^i \}$ converge (al limite per $i \to \infty$) ad una qualche soluzione ottima y^* di (D). In pratica la convergenza può essere lenta, per cui questi approcci in generale sono utilizzabili solamente quando si possa accettare un'approssimazione non particolarmente accurata del valore ottimo. Questo però può essere il caso ad esempio nelle applicazioni del Capitolo 7; poichè una soluzione di questa qualità può essere ottenuta in un numero relativamente contenuto di iterazione, e l'algoritmo sostanzialmente non ha altri costi rilevanti se non la soluzione di (P_y), questi approcci possono essere utili in pratica. Restano chiaramente molti dettagli da specificare: tipicamente il valore di $z(D)$ non è noto, e quindi deve essere approssimato con regole opportune. Queste dispense non sono la sede opportuna per una descrizione approfondita di questi e dei molti altri algoritmi esistenti per la soluzione del duale Lagrangiano; per tali argomenti si rimanda alla letteratura citata ed a corsi avanzati dell'area di Ricerca Operativa.

6.3.3 Informazione generata dal rilassamento Lagrangiano

Risolvere un duale Lagrangiano, ad esempio utilizzando l'algoritmo dei piani di taglio presentato nel paragrafo precedente, fornisce, oltre alla valutazione superiore, una quantità di informazione sul problema equivalente a quella prodotta da un rilassamento continuo (cf. §6.1.2). Infatti, al termine dell'algoritmo si dispone sia di una soluzione ottima y^* di (D) che di una soluzione ottima x^* di $(\tilde{P})$; quest'ultima è una soluzione frazionaria del tutto analoga a quella prodotta dal rilassamento continuo—le due coincidono se il rilassamento Lagrangiano ha la proprietà di integralità. In effetti, il rilassamento Lagrangiano produce un'informazione potenzialmente "più ricca" di quella fornita dal rilassamento continuo: non solo una soluzione continua x^*, ma un insieme di soluzioni $\mathcal{B} \subset X$ ed i relativi moltiplicatori convessi θ_x che producono x^*. Forniremo adesso alcune brevi considerazioni su come sia possibile sfruttare questa informazione per la soluzione di un problema di *OC*.

6.3.3.1 Uso dell'informazione primale

La soluzione continua x^* di $(\tilde{P})$ può chiaramente essere usata all'interno di tecniche di arrotondamento, quali quelle viste nel §6.1.2.1, esattamente allo stesso modo in cui viene utilizzata la soluzione ottima di un rilassamento continuo (lo stesso dicasi per le regole di separazione negli algoritmi enumerativi, che saranno discusse nel prossimo capitolo). Poichè, nel caso in cui il rilassamento Lagrangiano non abbia la proprietà di integralità, $(\tilde{P})$ è un rilassamento "più accurato" di quello continuo, ci si può aspettare che le soluzioni euristiche costruite a partire da x^* siano, in generale, di qualità almeno comparabile a quelle costruite a partire dalla soluzione ottima del rilassamento continuo.

In più, il processo di soluzione del duale Lagrangiano può essere sfruttato per produrre soluzioni ammissibili di (P). Si parla in tal caso di *euristiche Lagrangiane*. In molti casi si utilizza la soluzione $\bar{x}$ del rilassamento Lagrangiano all'iterazione corrente, intera ma che viola i vincoli $Ax \leq b$, e la si rende ammissibile mediante una procedura euristica, ad esempio di tipo greedy. Alla soluzione ammissibile così ottenuta possono poi essere applicate euristiche di raffinamento, tipicamente di ricerca locale, per migliorarne la qualità. Il processo iterativo per la soluzione di (D) funge quindi da "multistart" per normali euristiche, che in più possono utilizzare i costi Lagrangiani—che contengono informazione relativa ai vincoli rilassati—per guidare la costruzione della soluzione ammissibile. Le euristiche possono essere invocate ad ogni iterazione, oppure ad intervalli prestabiliti, oppure solamente quando viene prodotta una soluzione $\bar{x}$ con determinate caratteristiche. Le euristiche possono essere invocate in modo uniforme durante il processo iterativo oppure essere invocate più spesso in determinate fasi, tipicamente verso il termine dell'esecuzione dell'algoritmo, in quanto i moltiplicatori Lagrangiani y sono di "migliore qualità". Al limite è possibile invocare le euristiche solamente in corrispondenza del vettore di moltiplicatori Lagrangiani ottimo, per quanto ciò non garantisca di determinare la migliore tra le soluzioni ottenibili. Le soluzioni ammissibili prodotte dall'euristica Lagrangiana possono poi essere usate, ad esempio, come popolazione di partenza per algoritmi di tipo genetico.

Esempio 6.13. Euristiche Lagrangiane per (TSP)
Per il (TSP) abbiamo a disposizione rilassamenti combinatori basati su (MST) (ad esempio l'1-albero di costo minimo, cf. §6.6) ed euristiche che costruiscono soluzioni ammissibili a partire da un albero di copertura (ad esempio "twice around MST" e l'euristica di Christfides, cf. §5.1.2.3). È quindi facile trasformare tali euristiche in euristiche Lagrangiane semplicemente applicandole all'albero di copertura prodotto dal rilassamento Lagrangiano corrispondente a quello combinatorio. Il ciclo Hamiltoniano così ottenuto può poi essere utilizzato come punto di partenza per un'euristica di ricerca locale come quelle discusse nel capitolo precedente.

Esempio 6.14. Euristiche Lagrangiane per (CMST)
Per costruire un rilassamento Lagrangiano di (CMST) possiamo operare come nel §6.2 e rilassare i vincoli $f_{ij} \leq Qx_{ij}$ della formulazione ivi proposta in modo tale da ottenere un rilassamento Lagrangiano che si decompone in due sottoproblemi indipendenti. Alternativamente, possiamo considerare una diversa formulazione in cui i vincoli per cui in ogni vertice può entrare al più un arco sono sostituiti dai più forti vincoli di connessione (1.12) che definiscono il problema (MST) (sulle variabili y_{ij}). In questo modo, il rilassamento Lagrangiano dei vincoli di conservazione di flusso nella formulazione diviene un problema (MST) in cui i costi delle variabili x vengono "proiettati" sulle variabili y.

Esercizio 6.21. Si discuta come sia possibile risolvere questo rilassamento Lagrangiano utilizzando una sola computazione di un albero di copertura di costo minimo.

Il rilassamento Lagrangiano potrebbe, specialmente in corrispondenza a "buoni" moltiplicatori Lagrangiani, produrre alberi ammissibili rispetto al vincolo di peso sui sottoalberi della radice, e quindi fornire a costo nullo anche soluzioni ammissibili per il problema. Qualora ciò non accadesse si potrebbero modificare gli algoritmi di ricerca locale visti nel capitolo precedente per cercare di rendere ammissibile l'albero prodotto, ad esempio attraverso mosse di "Cut & Paste" o scambio di sottoalberi (cf. §5.2.1.4). Una volta prodotta una soluzione ammissibile sarebbe poi naturale applicarvi le normali procedure di ricerca locale per cercare di migliorare la funzione obiettivo.

Esercizio 6.22. Si discuta come modificare le euristiche di ricerca locale per il problema (CMST) in modo tale da produrre una soluzione ammissibile a partire da un albero di copertura che viola il vincolo di peso sui sottoalberi della radice.

Nel contesto delle euristiche Lagrangiane può risultare utile non solo la soluzione $\bar{x}$, ma l'intero insieme delle soluzioni $\mathcal{B}$ generato nel corso della soluzione del duale Lagrangiano, ed i moltiplicatori θ_x ad esse associati. In particolare, è interessante rilevare che i moltiplicatori θ_x hanno la forma di una distribuzione di probabilità sugli elementi di $\mathcal{B}$. Ciò può suggerire approcci in cui le soluzioni $x \in \mathcal{B}$ vengono combinate per costruire una soluzione ammissibile sfruttando queste indicazioni. Nel caso poi in cui X sia decomponibile e venga utilizzato un algoritmo di tipo disaggregato (cf. §6.3.2), si hanno "probabilità" diverse per componenti x^h diverse provenienti dalla stessa soluzione del rilassamento Lagrangiano, e quindi si ottiene naturalmente un effetto di "ibridazione" in cui la soluzione complessiva viene costruita sfruttando componenti provenienti da soluzioni diverse del rilassamento Lagrangiano. Nel caso di (UMMCF), ad esempio, ad ogni passo ciascun insieme $\mathcal{B}^h$ contiene cammini da o_h a d_h, e i

moltiplicatori θ possono essere (arbitrariamente) interpretati come "probabilità che il cammino faccia parte di una soluzione ottima di (P)".

Esercizio 6.23. Si propongano euristiche Lagrangiane per i problemi di *OC* presentati in questo e nei precedenti capitoli.

6.3.3.2 Uso dell'informazione duale

La soluzione ottima y^* di (D) fornisce un'informazione sui vincoli rilassati $Ax \leq b$ del tutto analoga a quella fornita dalla soluzione duale ottima del rilassamento continuo—le due coincidono se il rilassamento Lagrangiano ha la proprietà di integralità. Vedremo nel prossimo paragrafo un possibile uso di tale informazione. Poichè, nel caso in cui il rilassamento Lagrangiano non abbia la proprietà di integralità, $(\tilde{P})$ è un rilassamento "più accurato" di quello continuo, ci si può aspettare che l'informazione sull'"importanza" dei vincoli contenuta in y^* sia, in generale, di qualità almeno comparabile a quella contenuta nella soluzione duale ottima del rilassamento continuo.

Per quanto riguarda i costi ridotti, se il rilassamento Lagrangiano (P_y) è un problema di *PL*—o un suo caso particolare, come ad esempio un problema di flusso su grafo o di cammino minimo—allora i costi ridotti delle variabili in (P_{y^*}) possono essere usati, ad esempio, per il fissaggio basato sui costi ridotti (cf. §6.1.2.2), esattamente allo stesso modo in cui vengono usati i costi ridotti del rilassamento continuo. Ad esempio, se $c_i^* < 0$ è il costo ridotto di una variabile binaria x_i cha ha valore 0 nella soluzione ottima di (P_{y^*}), e si ha $z(D) + c_i^* < \underline{z}$, allora x_i ha sicuramente valore pari a 0 in qualsiasi soluzione ottima di (P), ed analogamente per il caso in cui $x_i^* = 1$ e $c_i^* > 0$.

Esercizio 6.24. Si dimostri l'affermazione precedente.

In effetti, nel caso del rilassamento Lagrangiano queste relazioni possono essere verificate non solamente al termine dell'algoritmo, ossia quando si conosce y^*, ma ogniqualvolta si risolva un rilassamento Lagrangiano $(P_{\bar{y}})$ in corrispondenza ad un qualsiasi vettore $\bar{y}$ di moltiplicatori Lagrangiani.

6.4 Rilassamento surrogato

Un ulteriore modo per utilizzare la struttura presente in problemi con la forma (6.1) è quello di effettuare un *rilassamento surrogato* di (P) rispetto agli m vincoli complicanti $Ax \leq b$: fissato un vettore $y \in \mathbb{R}^m_+$ di *moltiplicatori surrogati*, questo è il problema

$$(RS_y) \qquad \max\{\, cx \,:\, (yA)x \leq (yb) \;\;,\;\; Ex \leq d \;\;,\;\; x \in \mathbb{Z}^n \,\} \;\;.$$

In questo caso, però, non è detto che il problema sia "facile": gli m vincoli complicanti $Ax \leq b$ sono stati rimpiazzati dal singolo *vincolo surrogato* $(yA)x \leq (yb)$, che potrebbe però a sua volta essere "complicante". In effetti, il rilassamento surrogato viene utilizzato molto meno frequentemente di quello Lagrangiano proprio perchè, come abbiamo spesso accennato nei paragrafi e capitoli precedenti, la semplice aggiunta di un singolo vincolo lineare spesso trasforma un problema "facile" in uno "difficile": si veda ad esempio il caso di (CSP). Comunque, poichè (RS_y) ha "meno vincoli" di (P), si può sperare che sia in qualche senso più facile da risolvere. Si pensi ad esempio al caso di un problema di Programmazione 0/1 generico (i vincoli $Ex \leq d$ sono semplicemente $x \in [\,0\,,\,1\,]^n$): in questo caso (RS_y) è un problema dello zaino. Come vedremo in seguito, il problema dello zaino è in qualche modo "più facile" di un generico problema di Programmazione 0/1. Assumiamo dunque di avere a disposizione un algoritmo "ragionevolmente efficiente" per risolvere (RS_y).

È immediato verificare che, comunque scelto y, (RS_y) è un rilassamento di (P), ossia risulta $z(RS_y) \geq z(P)$. Infatti, le due funzioni obiettivo coincidono e la regione ammissibile di (RS_y) contiene quella di (P): data una qualsiasi soluzione $\bar{x}$ ammissibile per (P), ossia tale che $A\bar{x} \leq b$, si ha chiaramente $(yA)\bar{x} \leq yb$ per ogni $y \geq 0$. Si noti che il viceversa non è vero, ossia possono esistere soluzioni che rispettano il vincolo surrogato ma non il sistema di vincoli originario $Ax \leq b$.

Esercizio 6.25. Fornire un esempio che dimostri l'affermazione precedente.

L'interesse del rilassamento surrogato risiede nel fatto che, in generale, fornisce valutazioni superiori non peggiori di quelle prodotte dal rilassamento Lagrangiano con lo stesso vettore di moltiplicatori y.

Teorema 6.3. $z(RS_y) \leq z(P_y) \quad \forall\, y \geq 0.$

Dimostrazione Basta notare che (P_y) può essere visto come un rilassamento Lagrangiano di (RS_y) rispetto all'unico vincolo "complicante" $(yA)x \leq yb$, con moltiplicatore Lagrangiano pari ad 1. ◇

Di conseguenza, il *duale surrogato* di (P) rispetto ai vincoli $Ax \leq b$

$$(DS) \qquad \min\{\, z(RS_y) \,:\, y \geq 0 \,\}$$

fornisce una valutazione superiore non peggiore (non maggiore) di quella fornita dal duale Lagrangiano (D). Purtroppo, anche qualora si disponga di un algoritmo efficiente per risolvere il (RS_y), risolvere (DS) è molto più "difficile" di risolvere il duale Lagrangiano. Infatti, mentre la funzione Lagrangiana φ è convessa, la *funzione surrogata* $\phi(\, y\,) = z(RS_y)$ non lo è; addirittura è una funzione non continua. In effetti la funzione ha proprietà interessanti dal punto di vista dell'ottimizzazione, in quanto è *quasi convessa* e pertanto i minimi locali sono anche minimi globali. Ciò nonostante gli algoritmi utilizzabili per risolvere il duale Lagrangiano non si adattano facilmente a quello surrogato: in particolare, la funzione è tipicamente costante a tratti, e l'informazione di tipo "subgradiente" non aiuta ad orientare la ricerca per il minimo locale. Ciò può essere visto ripercorrendo in questo caso le argomentazioni viste nel §6.3.2: per $y(\,\alpha\,) = y + \alpha d$ ed un passo $\alpha > 0$ "sufficientemente piccolo" affinché la soluzione ottima di (RS_y) sia la stessa $\bar{x}$ di $(RP_{y(\,\alpha\,)})$, si ha $\phi(\, y(\,\alpha\,)\,) = \phi(\, y\,)$ (a differenza del caso Lagrangiano, y non influenza il costo delle soluzioni). Se questo accade per tutte le d, il che è possibile, la funzione è costante in un intorno di y e gli algoritmi non hanno indicazioni su quali nuove soluzioni esplorare.

Per ovviare a questo inconveniente si può ricorrere ad uno "stratagemma" interessante che sfrutta la similitudine "sintattica" tra il rilassamento Lagrangiano e quello surrogato: si risolve il duale Lagrangiano (D), e si usano i moltiplicatori Lagrangiani ottimi y^* come moltiplicatori surrogati, risolvendo un singolo rilassamento surrogato. Questo fornisce una valutazione superiore non peggiore di $z(P_{y^*}) = z(D)$, e quindi può consentire di migliorare la valutazione fornita dal duale Lagrangiano; naturalmente non si ha nessuna garanzia che y^* sia una soluzione ottima del duale surrogato. Alternativamente si possono utilizzare come moltiplicatori surrogati le variabili duali ottime dei vincoli $Ax \leq b$ nel rilassamento continuo di (P), il che può consentire di ottenere una valutazione superiore migliore di quella fornita dal solo rilassamento continuo.

Esercizio 6.26. Si discuta sotto quali condizioni la procedura appena accennata ottiene sicuramente una valutazione superiore non peggiore di quella determinata dal duale Lagrangiano.

Riferimenti Bibliografici

T. Berthold, A. Lodi, D. Salvagnin "Ten years of feasibility pump, and counting" *EURO Journal on Computational Optimization* volume 7, pages 1—14, 2019

C. Lemaréchal "Lagrangian Relaxation" in *Computational Combinatorial Optimization*, M. Jünger and D. Naddef eds., Springer-Verlag, 2001

V. Varzirani "Approximation Algorithms" Springer-Verlag, 2001

L. Wolsey "Integer Programming" Wiley, 1998

Capitolo 7

Algoritmi enumerativi

Come abbiamo visto nei capitoli precedenti, gli algoritmi efficienti sono spesso non efficaci: né gli algoritmi greedy né quelli basati sulla ricerca locale sono in grado, in molte situazioni, di garantire l'ottimalità della soluzione trovata. Nel caso in cui sia importante determinare una soluzione ottima del problema è necessario quindi ricorrere ad algoritmi diversi. Esistono più approcci per la determinazione della soluzione esatta di problemi di *OC* $\mathcal{NP}$-ardui, ma tra questi quelli di *enumerazione implicita* sono certamente i più diffusi. Questi algoritmi esplorano in modo sistematico lo spazio delle soluzioni alla ricerca di una soluzione ottima. Le valutazioni inferiori (euristiche) e superiori (rilassamenti) del valore ottimo della funzione obiettivo discussi nei capitoli precedenti, insieme ad opportune regole di dominanza, vengono sfruttate per ottenere informazioni sul problema che permettano di escludere dalla ricerca aree dello spazio delle soluzioni in cui dimostrabilmente non si trovi una soluzione ottima; queste aree si dicono quindi *visitate implicitamente* dall'algoritmo. Le modalità della ricerca fanno sì che, al termine, si abbia la garanzia dell'ottimalità della soluzione determinata. Se opportunamente implementati, utilizzando euristiche e rilassamenti efficaci ed efficienti e molti altri importanti dettagli discussi nel seguito (regole di visita, separazione e dominanza, pre-trattamento, tecniche poliedrali ...), gli algoritmi di enumerazione implicita riescono spesso a risolvere in tempi accettabili istanze di dimensioni rilevanti di problemi $\mathcal{NP}$-ardui. In particolare sono a tutt'oggi disponibili strumenti software—anche open-source—in grado spesso di risolvere all'ottimo, o comunque con elevata accuratezza, in tempi ragionevolmente brevi problemi di *PLI* di dimensioni rilevanti (migliaia o decine di migliaia di variabili, a volte anche dimensioni maggiori) per le istanze che derivano da molti contesti applicativi reali. Comunque, anche usando le migliori tecnologie disponibili, non si può mai escludere l'eventualità di dover esaminare una frazione consistente dello spazio delle soluzioni, per cui questi algoritmi hanno in generale una complessità esponenziale. Nonostante il buon successo dei solutori di *PLI general-purpose*, non esistono—e non possono esistere, a meno che $\mathcal{P} = \mathcal{NP}$—implementazioni generiche in grado di risolvere istanze di dimensione arbitraria di qualsiasi problema di *OC*. Quando le tecniche disponibili non si rivelino sufficienti a risolvere le istanze di interesse per l'applicazione nei tempi richiesti con le risorse di calcolo disponibili, diviene necessario l'intervento di esperti in grado di comprendere il funzionamento di questi algoritmi ad un elevato livello di dettaglio, per intervenire su di essi sfruttando al meglio tutte le informazioni disponibili sullo specifico problema da risolvere. Questo si sostanzia spesso nell'interazione con i solutori di *PLI general-purpose*, che proprio per questo motivo dispongono di meccanismi (*callback*) che consentono all'utente di intervenire in alcune parti specifiche del processo risolutivo (euristiche, separazione di disuguaglianze valide, branching, ...) utilizzando logiche specializzate per il problema di *OC* di interesse; ciò può consentire miglioramenti molto sostanziali nell'efficacia ed efficienza dell'algoritmo. Più raro, ma comunque utile in diverse applicazioni, è il caso in cui si realizzi ex-novo un approccio di enumerazione implicita per uno specifico problema in *OC*. In entrambi questi casi è indispensabile una conoscenza dei meccanismi fondamentali di funzionamento di questa classe di algoritmi, che sono descritti in questo capitolo.

7.1 Algoritmi di enumerazione implicita

Descriveremo adesso le idee base degli algoritmi di enumerazione implicita, o *Branch and Bound* (B&B), fornendo uno schema molto generale di algoritmo e discutendone le principali proprietà. Ci soffermeremo poi su alcuni importanti aspetti dell'implementazione di algoritmi di questo tipo, discutendo alcuni esempi.

Gli algoritmi di enumerazione implicita possono essere visti come un caso particolare del ben noto schema algoritmico "divide et impera", che affronta la soluzione di un problema mediante i seguenti passi:

- suddividere il problema in un certo numero di sottoproblemi "più piccoli";
- risolvere separatamente i singoli sottoproblemi, tipicamente applicando ricorsivamente lo stesso procedimento finchè la soluzione non può essere ottenuta mediante un qualche procedimento alternativo (caso base);
- combinare le soluzioni dei singoli sottoproblemi per ottenere una soluzione del problema originale.

Consideriamo un generico problema di OC

$$(P) \qquad \max\{\, c(\,x\,) \,:\, x \in X \,\} \ ,$$

per il quale abbiamo a disposizione due procedimenti (arbitrariamente complessi) che forniscano, rispettivamente, una valutazione superiore $\bar{z}(P)$ ed una inferiore $\underline{z}(P)$ di $z(P)$. Dalla discussione del §4.3 ricordiamo che "risolvere" (P) significa certificare $\underline{z}(P) = \bar{z}(P)$. Se, come spesso accade, alla valutazione inferiore è anche associata una soluzione $\bar{x} \in X$ (tale che $c(\,\bar{x}\,) = \underline{z}(P)$), allora si è anche ottenuta una soluzione dimostrabilmente ottima. In generale, come già osservato, il rilassamento e l'euristica possono non essere sufficientemente efficaci per garantire la risoluzione del problema. Si deve pertanto applicare il meccanismo del "divide et impera", utilizzando la seguente osservazione:

Lemma 7.1. Sia $X_1, X_2, \ldots, X_k$ una suddivisione di X ($X_i \subset X$, $i = 1, \ldots, k$, $\bigcup_{i=1}^{k} X_i = X$) e sia

$$(P_i) \quad \max\{\, c(\,x\,) \,:\, x \in X_i \,\} \ ; \qquad \text{allora}$$
$$\max\{\, \underline{z}(P_i) \,:\, i = 1, \ldots, k \,\} \le z(P) \le \max\{\, \bar{z}(P_i) \,:\, i = 1, \ldots, k \,\} \ .$$

Esercizio 7.1. Dimostrare il lemma precedente, assumendo per semplicità che X sia un insieme finito. Si discuta poi quali ipotesi aggiuntive sono necessarie qualora X abbia cardinalità infinita, enumerabile e non enumerabile.

In altre parole, suddividendo un problema in un certo numero di sottoproblemi, l'unione delle cui regioni ammissibili sia la regione ammissibile del problema originario, ed ottenendo una valutazione superiore ed inferiore del valore ottimo di ciascuno dei essi individualmente, si ottengono (mediante una semplice operazione di massimo) una valutazione superiore ed inferiore del valore ottimo del problema originario. Ciò mostra come implementare l'operazione di ricombinazione dei risultati dei sottoproblemi nello schema del "divide et impera". Si noti che, delle due operazioni di massimo, una è "favorevole" mentre l'altra è "sfavorevole". In particolare, per restringere il gap tra la valutazione inferiore e quella superiore è necessario far crescere la prima e decrescere la seconda. L'operazione di massimo sulle valutazioni inferiori è quindi "favorevole" in quanto aiuta a far crescere la valutazione inferiore; ciò corrisponde al fatto che la migliore delle soluzioni trovate dall'euristica per i singoli sottoproblemi è una soluzione ammissibile per il problema originario. Viceversa, l'operazione di massimo sulle valutazioni superiori è "sfavorevole": per poter dimostrare che $z(P) \le \bar{z}$ occorre dimostrare che $z(P_i) \le \bar{z}$ per *ogni* $i = 1, \ldots, k$. In altri termini, per migliorare la valutazione superiore disponibile occorre che le valutazioni superiori corrispondenti a *tutti* i sottoproblemi siano migliori di essa.

Occorre però notare che ciascun sottoproblema è definito su un insieme ammissibile "più piccolo" di quello di (P), e quindi ci si può aspettare che sia "più facile". In effetti, se l'insieme ammissibile del sottoproblema è "abbastanza piccolo" il problema diviene banale: ad esempio, per $X_i = \{\, \bar{x} \,\}$ si ha ovviamente $\underline{z}(P_i) = c(\,\bar{x}\,) = \bar{z}(P_i)$, mentre per $X_i = \emptyset$ si ha $\underline{z}(P_i) = \bar{z}(P_i) = +\infty$. In altre parole, esiste una suddivisione "sufficientemente fine" di X in un opportuno numero (esponenziale) di sottoinsiemi

tale che i corrispondenti sottoproblemi siano sicuramente risolubili; è sufficiente che i sottoproblemi abbiano al più una soluzione. Questi sono quindi sicuramente possibili "casi base" del procedimento "divide et impera" per un problema di *OC*.

Questo tipo di considerazioni può essere facilmente esteso a problemi con parziale struttura combinatoria, quali ad esempio i problemi di Programmazione Lineare Mista: esiste un numero finito (per quanto esponenziale) di possibili sottoproblemi, corrispondenti ai possibili valori del vettore delle variabili intere, ciascuno dei quali è un problema di *PL* e quindi "facilmente" risolubile. In realtà questo è vero, strettamente parlando, solamente se i vincoli del problema definiscono un insieme compatto rispettivamente alle variabili intere, come ad esempio $\{ 0 , 1 \}^n$; si può però dimostrare che se i coefficienti della matrice dei vincoli e del vettore dei lati destri sono razionali questo può essere assunto senza perdita di generalità.

Conviene a questo punto introdurre il modo più utilizzato (per quanto non l'unico possibile) per implementare l'*operazione di separazione*, ossia la suddivisione di X in $X_1, X_2, \ldots, X_k$: infatti, ciò mostra un diverso modo di descrivere questi algoritmi. Per semplificare l'esposizione supponiamo inizialmente che sia $X \subseteq \{ 0 , 1 \}^n$, ossia che le soluzioni del problema possano essere descritte attraverso n decisioni binarie, ciascuna rappresentata da una variabile x_i, $i = 1, \ldots, n$: un modo possibile per suddividere X è quello di *prendere decisioni* su alcune delle variabili. Ad esempio, fissato un qualsiasi indice i, possiamo partizionare X come $X_0 \cup X_1$, dove

$$X_0 = \{ x \in X \, : \, x_i = 0 \} \quad \text{e} \quad X_1 = \{ x \in X \, : \, x_i = 1 \} \ .$$

Esistono quindi molti modi diversi di partizionare lo stesso insieme X, a seconda ad esempio della scelta dell'indice i (della decisione da prendere). Inoltre, gli insiemi così ottenuti possono a loro volta essere partizionati seguendo lo stesso schema; ad esempio, fissato un altro indice $j \neq i$, possiamo partizionare X_0 e X_1 come

$$\begin{aligned} X_0 &= (X_{00} = \{ x \in X_0 \, : \, x_j = 0 \}) \cup (X_{01} = \{ x \in X_0 \, : \, x_j = 1 \}) \\ X_1 &= (X_{10} = \{ x \in X_1 \, : \, x_j = 0 \}) \cup (X_{11} = \{ x \in X_1 \, : \, x_j = 1 \}) \end{aligned} \ .$$

In questo modo si possono ottenere partizioni di X di qualsiasi dimensione; infatti, per ottenere un sottoinsieme che contenga (al più) una sola soluzione è sufficiente ripetere il procedimento n volte, ossia prendere decisioni su tutte ed n le variabili. In effetti, è possibile rappresentare l'insieme ammissibile X attraverso un *albero delle decisioni*, che associa a ciascuna soluzione ammissibile una sequenza di decisioni che la generi.

Esempio 7.1. Un semplice albero delle decisioni

Si consideri il caso in cui $X = ([x_1 , x_2 , x_3] \in \{ 0 , 1 \}^3)$: un albero delle decisioni per X è mostrato in Figura 7.1. Ciascuna foglia dell'albero corrisponde ad un elemento di X; equivalentemente, ciascun cammino dalla radice ad una foglia rappresenta la sequenza di decisioni che genera quell'elemento. Ciascun nodo interno rappresenta un sottoinsieme di X, ed il cammino dalla radice a quel nodo rappresenta la sequenza di decisioni che caratterizzano tutti gli elementi di quel sottoinsieme. La radice dell'albero corrisponde all'intero insieme X, ossia alla sequenza vuota di decisioni.

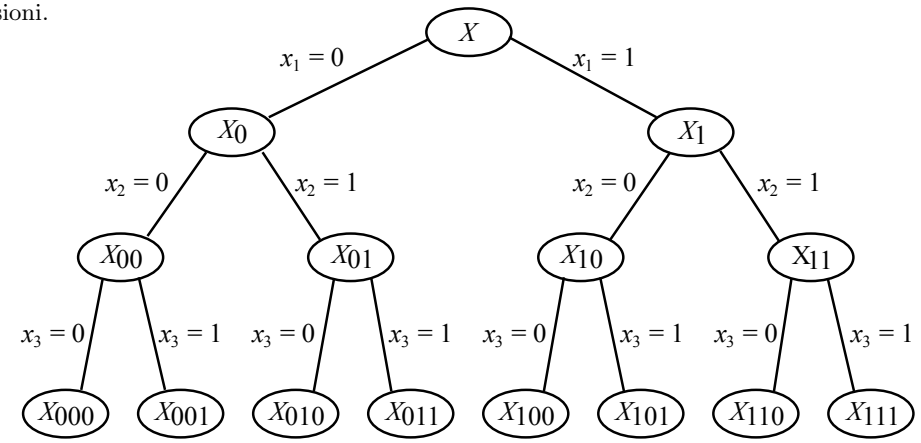


Figura 7.1: Un albero delle decisioni per $\{ 0 , 1 \}^3$

Si noti che l'albero delle decisioni corrispondente ad uno stesso insieme X non è unico, non foss'altro che per l'ordine in cui vengono prese le decisioni (l'ordinamento delle variabili). In generale, per problemi di OC con particolari strutture può essere utile costruire l'albero delle decisioni in modo tale da "rispettare" la struttura del problema.

Esempio 7.2. Un albero delle decisioni per (TSP)
Si consideri ad esempio l'insieme X di tutti i cicli Hamiltoniani di un dato grafo $G = (V, E)$: poichè X può essere identificato con un opportuno sottoinsieme di $\{0, 1\}^{|E|}$, è possibile costruire un albero delle decisioni per X esattamente come illustrato in precedenza: si considerano i lati (le variabili) secondo un qualsiasi ordinamento prefissato e, ad ogni livello dell'albero delle decisioni, si decide se un determinato lato appartiene oppure no al ciclo. Un diverso albero delle decisioni per X può però essere ottenuto nel modo seguente: selezionato in G un nodo arbitrario (ad esempio il nodo 1), si suddivide X in tanti sottoinsiemi quanti sono i lati uscenti dal nodo, ove ciascun sottoinsieme contiene tutti i cicli Hamiltoniani che contengono quel particolare lato. In altre parole, in ciascuno dei sottoinsiemi si è presa la decisione che il lato corrispondente deve appartenere al ciclo Hamiltoniano. Per continuare la costruzione dell'albero delle decisioni si itera il procedimento: a ciascun nodo interno X' dell'albero delle decisioni è associato un cammino semplice P di G che inizia dal nodo 1 e termina in un certo nodo $i \in N$, e $X' = \{C \in X : P \subseteq C\}$. X' avrà quindi tanti figli quanti sono i lati $\{i, j\} \in E$ tali che j non appartiene a P, ossia uno per ciascun arco che può essere aggiunto a P ottenendo ancora un cammino semplice. Il figlio di X' corrispondente al lato $\{i, j\}$ contiene tutti i cicli Hamiltoniani che contengono $P \cup \{\{i, j\}\}$; si è cioè presa l'ulteriore decisione che anche $\{i, j\}$ deve appartenere al ciclo. A ciascuna foglia dell'albero delle decisioni corrisponde quindi un cammino semplice P di G che non può essere ulteriormente esteso; se P è Hamiltoniano ed esiste il lato tra il suo nodo terminale ed 1, allora al nodo è univocamente associato un ciclo Hamiltoniano di G. L'albero delle decisioni relativo al grafo in Figura 7.2(a) è mostrato in Figura 7.2(b). Per semplificare la rappresentazione, in ciascun nodo dell'albero delle decisioni è riportato il nodo terminale i del corrispondente cammino semplice P in G (nella radice è riportato il nodo iniziale, ovvero 1, di ogni cammino).

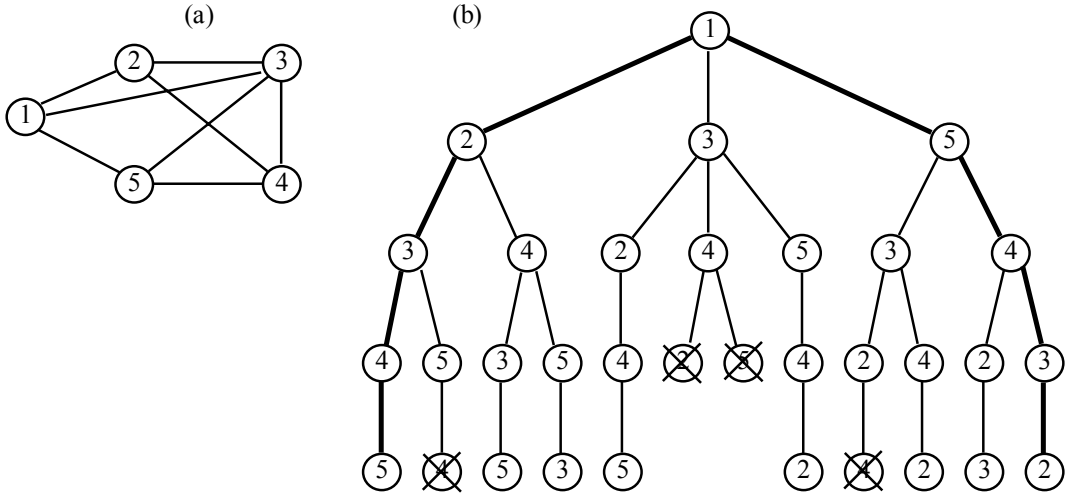


Figura 7.2: Un albero delle decisioni per (TSP)

Questo albero delle decisioni ha alcune caratteristiche rilevanti che è opportuno sottolineare, e che lo differenziano da quello in Figura 7.1:

- il numero di nodi per ogni figlio può essere diverso da due e non costante;
- alcuni cammini nell'albero delle decisioni terminano in nodi (evidenziati in figura con una croce) che corrispondono a sottoinsiemi di X vuoti, ossia a sequenze di decisioni che non generano nessun ciclo Hamiltoniano;
- vengono prese solamente decisioni di un certo tipo, ossia si decide quali lati appartengono al ciclo ma non si decide quali lati *non* appartengono al ciclo, ovvero si fissano variabili a 1 ma non si fissano mai variabili a 0: le decisioni rispetto i lati che non appartengono al ciclo sono prese *implicitamente* (per ogni nodo $h \in N$ interno al cammino P associato ad un nodo dell'albero delle decisioni sono già decisi i due lati incidenti che faranno parte del ciclo, quindi tutti gli altri lati incidenti in h sono di fatto esclusi dal ciclo);
- i sottoinsiemi X'_i, figli di un certo sottoinsieme X' corrispondente ad un nodo nell'albero delle decisioni, non necessariamente formano una partizione di X', ossia può risultare $X'_i \cap X'_j \neq \emptyset$; ad esempio, i due cammini nell'albero delle decisioni evidenziati nella figura corrispondono ovviamente allo stesso ciclo Hamiltoniano di G.

L'albero delle decisioni è uno strumento in grado di generare in modo sistematico tutte le soluzioni ammissibili di X; un algoritmo che intenda esplorare in modo esaustivo X può quindi procedere

visitando un qualsiasi albero delle decisioni di X. Ovviamente, nel caso di un problema di *OC* l'albero delle decisioni avrà dimensione esponenziale, e quindi una sua visita completa è in generale troppo costosa. L'uso di valutazioni superiori ed inferiori può però consentire di evitare di visitare effettivamente alcune zone dell'albero delle decisioni, rendendo l'approccio potenzialmente efficiente in pratica per istanze di dimensioni compatibili con applicazioni reali. È però vero che l'algoritmo ha potenzialmente complessità esponenziale, e che—a differenza di altri casi visti in precedenza, come gli algoritmi del simplesso o alcune varianti di SPT—l'esplosione combinatoria del numero di nodi dell'albero delle decisioni si osserva frequentemente in pratica. Per questo lo schema abbastanza generale di algoritmo di enumerazione implicita (B&B) che descriveremo è posto direttamente in forma di algoritmo ε-*approssimato*, ossia in cui si è soddisfatti quando (il valore ottimo del)la soluzione ottenuta ha un gap dimostrabilmente inferiore ad un parametro ε fornito in input. La scelta di tale parametro può essere cruciale per la possibilità di veder terminare l'algoritmo in tempi ragionevoli, in quanto il tempo di esecuzione può crescere in modo drammatico quando ε diminuisce. Valori tipici del gap *relativo* che vengono considerati per le applicazioni sono `1e-4` (corrispondente a soluzioni che hanno un errore massimo dello 0.01%) o `1e-6` per casi in cui si richiedono soluzioni con alta accuratezza, ma non è infrequente accettare gap di `1e-3` o anche superiori.

L'algoritmo costruisce e visita un albero delle decisioni del problema: la radice dell'albero rappresenta il problema originale (P) (il suo intero insieme ammissibile X), mentre il generico nodo dell'albero rappresenta un sottoproblema

$$(P') \quad \max\{\, c(\,x\,) \,:\, x \in X' \,\} \ ,$$

con $X' \subseteq X$. Un elemento che rende non del tutto generale lo schema è il fatto che la funzione obiettivo viene considerata costane. Ciò è ragionevole per i problemi che possono essere formulati come *PLI*, in quanto in generale non ci sono ragioni per modificare una funzione lineare. La situazione può essere diversa per problemi di *Programmazione NonLineare* (*PNL*) in cui anche la funzione obiettivo può essere "complessa"; esistono quindi schemi algoritmici che la approssimano iterativamente in modi diversi man mano che l'approccio procede. Si noti, però, che qualsiasi problema nella forma generale (P) può essere riformulato come un problema con funzione obiettivo lineare attraverso la semplice introduzione di una variabile ausiliaria

$$(P) \quad \max\{\, v \,:\, v \leq c(\,x\,) \ , \ \ x \in X \,\}$$

"nascondendo" la "complessità" della funzione obiettivo nell'insieme ammissibile X, il che rende la nostra descrizione dell'approccio corretta per il livello di dettaglio appropriata per queste dispense.

La relazione di discendenza nell'albero corrisponde all'applicazione ricorsiva del procedimento "divide et impera": i sottoproblemi rappresentati dai figli di un nodo hanno come regioni ammissibili quelle ottenute dalla suddivisione della regione ammissibile del problema rappresentato dal nodo. Le valutazioni superiori ed inferiori vengono utilizzate per evitare di visitare interi sottoalberi dell'albero delle decisioni. Nel caso più semplice, in corrispondenza ad un nodo (P') per cui si abbia $\underline{z}(P') = \bar{z}(P')$ (o, più in generale, $\bar{z}(P') \leq \underline{z}(P') + \varepsilon$), ossia il sottoproblema venga risolto (con errore massimo ε) dalla combinazione del rilassamento e dell'euristica disponibili, la visita del sottoalbero di radice (P') viene evitata, in quanto si conoscono una valutazione superiore ed inferiore "sufficientemente accurate" di $z(P')$. In questo caso si dice che il sottoalbero viene *visitato implicitamente*, o *potato*. Uno schema generale di algoritmo B&B è rappresentato nel seguente pseudo-codice.

```
procedure z = B&B( P , ε )    {
  Q = { (P) }; z = −∞;
  do { (P′) = Next( Q ); Q = Q \ { (P′) };
       ( z̄ , z ) = rilassamento( P′ );
       if( z > z ) then z = z;
       if( z̄ ≤ z + ε ) then continue;
       z = euristica( P′ );
       if( z > z ) then z = z;
       if( z̄ ≤ z + ε ) then continue;
       Q = Q ∪ branch( P′ );
       } while( Q ≠ ∅ );
  }
```

Procedura 7.1: Algoritmo *B&B*

L'algoritmo prende in input, oltre al(la descrizione di un'istanza del) problema da risolvere, la tolleranza (assoluta) $\varepsilon \geq 0$ richiesta per la soluzione, e ritorna una valutazione inferiore z sul valore ottimo di (P) con un gap non superiore a ε, ossia per la quale è garantito che $z \leq z(P) \leq z + \varepsilon$. L'algoritmo visita un sottoinsieme dell'albero delle decisioni; poiché l'albero ha una dimensione in principio esponenziale, la parte dell'albero visitata viene costruita dinamicamente nel corso della visita. L'insieme Q contiene i *nodi attivi*, ossia i nodi che sono stati generati ma non ancora esplorati (esaminati), e viene inizializzato con la radice (P) dell'albero. L'algoritmo mantiene (ed infine ritorna) la miglior valutazione inferiore $z \leq z(P)$ determinata fino all'iterazione corrente. z viene inizializzato a $-\infty$, ed assume valore finito appena viene generata la prima soluzione ammissibile; se al termine dell'algoritmo si ha $z = -\infty$ allora $X = \emptyset$; si noti che ovviamente questo rispetta $z \leq z(P) \leq z + \varepsilon$, in quanto allora $z(P) = -\infty$. Normalmente a z è associata la miglior soluzione $x \in X$ determinata fino a quel momento (detta *incumbent*), tale che $z = c(\,x\,)$; al termine x viene restituita come soluzione ε-ottima del problema. Nello pseudocodice questo aspetto viene trascurato e si assume di essere interessati solamente al calcolo del *valore ottimo* del problema (il che in un certo senso può essere assunto senza perdita di generalità, si veda il §4.3).

Nella generica iterazione dell'algoritmo viene estratto un nodo (P') da Q; la regola di selezione del nodo in Q (le funzione $Next()$) determina la *strategia di visita* dell'albero delle decisioni. Viene quindi risolto un *rilassamento*

$$(\bar{P}') \quad \max\{\, \bar{c}(\,x\,) \,:\, x \in \bar{X}' \,\}$$

di (P'), in cui si ha cioé $\bar{X}' \supseteq X'$ e $\bar{c}(\,x\,) \geq c(\,x\,)$ per ogni $x \in X'$; la soluzione ottima x' di $(\bar{P}')$ produce una valutazione superiore $\bar{c}(\,x'\,) = \bar{z} \geq z(P')$. Tale valutazione viene utilizzata per l'operazione critica (detta *bounding*) dell'algoritmo: se si ha $\bar{z} \leq z + \varepsilon$, allora nessuna soluzione in X' ha un valore che supera quello dell'incumbent di più di ε, e si può evitare di esplorare ulteriormente la parte dello spazio delle soluzioni rappresentata da X'. In questo caso si dice che il nodo (P') è stato *potato* ("pruned") *dalla valutazione superiore*; infatti, il nodo viene scartato e si passa immediatamente a visitare un altro dei nodi attivi (se ve ne sono).

Un caso particolare in cui ciò sicuramente accade è quando $\bar{X}' = \emptyset$, e quindi si pone, per definizione $\bar{z} = -\infty$; in questo caso si dice che il nodo è stato *potato per inammissibilità*. Come abbiamo già notato, è necessario essere in grado di *risolvere all'ottimo* $(\bar{P}')$, in quanto una soluzione euristica non garantisce di determinare una corretta valutazione superiore di $z(P')$; ciò implica che si deve, in particolare, essere in grado di determinare se l'insieme ammissibile $\bar{X}'$ di (P') è oppure no vuoto (questo può non essere un compito banale, si pensi al caso della PL e, a maggior ragione, a molti problemi $\mathcal{NP}$-ardui). Se $\bar{X}' = \emptyset$ allora anche $X' = \emptyset$ (in quanto $X' \subseteq \bar{X}'$); è quindi inutile esplorare ulteriormente questa parte dello spazio delle soluzioni, in quanto non ne contiene alcuna. Si noti che, in generale, non vale l'inclusione opposta: può cioè capitare che $X' = \emptyset$ ma $\bar{X}' \neq \emptyset$. In questo caso è in effetti ancora "inutile" proseguire l'esplorazione di quel sottoalbero, ma ciò può risultare comunque necessario perché il rilassamento non è in grado di *dimostrare* che $X' = \emptyset$.

Come evidenziato nello pseudo-codice, il rilassamento può anche produrre una valutazione *inferiore*; questo in particolare capita se la sua soluzione ottima è ammissibile ($x' \in X'$), e si può quindi porre

$\underline{z} = c(x')$. Se inoltre i valori delle due funzioni obiettivo coincidono, ossia $\bar{c}(x') = c(x')$, allora x' è in particolare ottima per (P'): in questo caso si dice che il nodo viene *potato per ottimalità*, in quanto sarebbe inutile proseguire ulteriormente nell'esplorazione di quel sottoalbero, avendo già dimostrabilmente ottenuto la migliore delle soluzioni in X'. Ciò è assicurato dal fatto che, come risulta naturale, qualora $z < \underline{z}$ il valore dell'incumbent viene aggiornato per tener conto della nuova e migliore valutazione inferiore ottenuta. Ovviamente il nodo viene in generale potato per *ε-ottimalità*. Se il rilassamento non determina alcuna soluzione ammissibile, basta porre $\underline{z} = -\infty$.

Tutto ciò aiuta a capire come il rilassamento (P') sia una componente fondamentale dell'algoritmo. Infatti esso:

- determina una valutazione superiore che può permettere di evitare di esplorare parti dell'albero delle decisioni;
- può determinare se parti dell'albero delle decisioni non contengono nessuna soluzione;
- può determinare soluzioni ammissibili, e quindi anche le valutazioni inferiori che sono parimenti necessarie per "potare" i nodi.

In principio si potrebbe implementare l'algoritmo *B&B* anche qualora non si disponga di un rilassamento: basta porre $\bar{z} = +\infty$ a meno che $X' = \{\bar{x}\}$, nel qual caso si pone invece $\bar{z} = c(\bar{x})$. Naturalmente, in questo caso tipicamente si esplora tutto l'albero delle decisioni, e quindi l'algoritmo non è efficiente. Ciò in effetti può essere in parte evitato attraverso l'uso di opportune *regole di dominanza*; per semplicità rimanderemo però la discussione di questo concetto ad un paragrafo successivo.

È bene rimarcare che i nodi "potati" sono quelli in cui la visita si interrompe, ossia le foglie del sottoalbero che viene effettivamente generato (e visitato); non necessariamente sono, e sperabilmente *non* sono, foglie dell'intero albero delle decisioni. Eliminare un nodo permette di non visitare (esplicitamente) tutto il corrispondente sottoalbero dell'albero delle decisioni (per questo si parla di "potatura"); è quindi di fondamentale l'importanza utilizzare rilassamenti (ed euristiche) efficaci, che permettano di potare nodi quanto più precocemente possibile nella visita, evitando così di generare ampie porzioni dell'albero delle decisioni.

Il fatto che nodo non venga potato, ossia che $\bar{z} > z + \varepsilon$, può dipendere da due diversi fattori (oltreché, ovviamente, dalla scelta di ε): o la valutazione superiore non è sufficientemente piccola, oppure la valutazione inferiore non è sufficientemente grande. In generale non è dato di sapere quale delle due cause sia quella prevalente, ma chiaramente migliorare la qualità dell'*incumbent* non può che essere benefico. Per questo, prima di rinunciare a potare e passare alla generazione di discendenti del nodo può essere appropriato applicare un'euristica a (P') per determinare una nuova valutazione inferiore $\underline{z} \leq z(P')$. Se la valutazione inferiore è migliore del (valore dell')incumbent z questo viene aggiornato, fornendo una nuova possibilità di potare il nodo, anche a parità di valutazione superiore.

Come abbiamo già osservato per il rilassamento, in linea di principio è possibile implementare l'algoritmo *B&B* anche qualora non si disponga di un'euristica: basta porre $\underline{z} = -\infty$ a meno che $\bar{X} = \{\bar{x}\}$, nel qual caso si pone invece $\underline{z} = c(\bar{x})$. A differenza del caso del rilassamento, però, questo tipo di scelta non è necessariamente disastrosa in pratica; in certi casi è persino la scelta più efficiente. Ciò dipende dal fatto che l'algoritmo ha altri due modi per produrre valutazioni inferiori:

1. ripetendo un numero sufficiente di volte l'operazione di suddivisione si generano insieme X' "piccoli", come ad esempio $X' = \{\bar{x}\}$, nei quali è "facile" determinare soluzioni ammissibili (in pratica, come vedremo, l'operazione spesso coincide col fissare valori per la variabili del problema: una volta fissate tutte le variabili si è determinata una soluzione, ammissibile o meno);
2. come abbiamo visto, il rilassamento può produrre autonomamente valutazioni inferiori (soluzioni ammissibili), ed in effetti questo tipicamente accade, almeno "ad una certa profondità nell'albero delle decisioni" (al limite nelle foglie, come nel caso precedente).

Se nonostante tutto questo il nodo non viene potato, ossia $\bar{z} > \underline{z} + \varepsilon$, allora (P') non è ancora stato risolto (con precisione sufficiente): il rilassamento e l'euristica non sono "abbastanza efficaci" per (P').

Si applica quindi il procedimento "divide et impera": si suddivide la regione ammissibile X' in un certo numero finito di regioni ammissibili più piccole $X'_1, \ldots X'_k$ (normalmente prendendo decisioni sul problema) e si aggiungono i corrispondenti sottoproblemi a Q. Ciò viene detto *branching* in quanto si aggiungono nuovi nodi ed archi al sottoalbero effettivamente generato dell'albero delle decisioni. A questo punto l'iterazione termina e si passa ad esaminare un altro dei nodi attivi. Si noti che, nello schema generale, non è detto che i sottoproblemi corrispondenti ad $X'_1, \ldots X'_k$ siano visitati immediatamente, né che siano visitati in sequenza; ciò dipende dalla strategia di visita dell'albero, che sarà discussa nel seguito.

Sotto alcune ipotesi, è facile dimostrare che l'algoritmo *B&B* determina $z(P)$ in un tempo finito.

Esercizio 7.2. Si dimostri l'affermazione precedente, discutendo quali siano le ipotesi necessarie.

Se l'euristica e/o il rilassamento producono soluzioni ammissibili, oltre alla valutazione inferiore, allora l'algoritmo produce anche una soluzione ε-ottima del problema. In effetti non è difficile modificare lo schema in modo da ottenere possibilmente più soluzioni ε-ottime alternative, qualora ne esistano.

Esercizio 7.3. Si discuta come modificare l'algoritmo *B&B* perchè possa produrre più di una soluzione ε-ottima di (P). Si discuta inoltre se sia possibile modificarlo in modo tale che produca *tutte* le soluzioni ε-ottime del problema.

Per dimostrare che l'algoritmo produce una valutazione inferiore di $z(P)$ affetta da errore assoluto minore od uguale a ε, ossia $z \geq z(P) - \varepsilon$, mostreremo che l'algoritmo in effetti produce una valutazione *superiore* arbitrariamente accurata di $z(P)$. Ciò si basa sul seguente risultato:

Lemma 7.2. Al termine di una qualsiasi iterazione dell'algoritmo *B&B*, sia Q' l'insieme dei predecessori (nell'albero delle decisioni) dei nodi in Q e Q'' l'insieme dei nodi "potati": allora

$$z(P) \leq \max\{\, \bar{z}(P') \,:\, P' \in Q' \cup Q'' \,\} \ .$$

Dimostrazione È facile verificare che

$$\bigcup_{P' \in Q} X' \ \cup \bigcup_{P' \in Q''} X' = X \ ,$$

ossia che l'unione delle regioni ammissibili dei problemi corrispondenti a tutti i nodi potati e dei problemi corrispondenti a tutti i nodi attivi è equivalente all'insieme ammissibile originario X. Infatti, questo è vero alla prima iterazione (in cui $Q = \{\,(P)\,\}$ e $Q'' = \emptyset$) e resta vero ogniqualvolta un nodo viene potato (il nodo viene rimosso da Q ma viene posto in Q'') oppure viene applicata l'operazione di separazione (il nodo viene rimosso da Q ma i suoi figli sono inseriti in Q', e per definizione l'unione delle regioni ammissibili dei figli contiene la regione ammissibile del padre). Ovviamente, questo è a maggior ragione vero sostituendo Q con Q' (alcuni nodi potati potrebbero essere figli di nodi in Q'). Il risultato segue quindi dal Lemma 7.1. ◇

Corollario 7.1. Al termine di una qualsiasi iterazione dell'algoritmo *B&B* si ha

$$z \leq z(P) \leq \bar{z} = \max\{\, z + \varepsilon \,,\, \max\{\, \bar{z}(P') \,:\, P' \in Q' \,\}\,\} \ ;$$

quindi, quando l'algoritmo termina ($Q = \emptyset \Rightarrow Q' = \emptyset$), allora z è ε-ottima.

Dimostrazione Sia (P') un nodo potato: sicuramente vale $\bar{z}(P') \leq z + \varepsilon$, indipendentemente dal fatto che (P') sia stato potato per inammissibilità ($\bar{z} = -\infty$) o dalla valutazione superiore (il che comprende anche il caso cui sia stato nodo potato per ottimalità). Infatti la relazione era vera al momento in cui (P') è stato potato, e z è non decrescente nel corso dell'algoritmo. Il risultato segue quindi dal Lemma 7.2. ◇

Si noti che la valutazione superiore $\bar{z}$ indicata dal Corollario 7.1 può essere effettivamente calcolata ad ogni iterazione dell'algoritmo, in quanto tutti i valori di $\bar{z}(P')$ richiesti sono stati effettivamente calcolati. Di consequenza, l'algoritmo *B&B* produce ad ogni iterazione una valutazione superiore ed una inferiore di $z(P)$, e termina quando le due coincidono a meno di ε. In effetti, l'algoritmo potrebbe (e dovrebbe) restituire anche $\bar{z}$ insieme a z. Ciò è particolarmente utile in quanto in pratica non è possibile sapere se l'algoritmo terminerà effettivamente in tempi sufficientemente rapidi: pertanto è tipico imporre anche un *time limit*, per cui l'algoritmo viene interrotto una volta che è stato speso tutto il tempo che l'utente è disposto ad aspettare. Se ciò accade non si ha che $\bar{z} \leq z + \varepsilon$, ossia non si è

ottenuta una soluzione dimostrabilmente ε-ottima, ma si dispone comunque della stima $\bar{z}-z \geq z(P)-z$ del gap che si ha accettando (la soluzione corrispondente a) z come "ottima". Si noti $\bar{z}-z>\varepsilon$ non implica necessariamente $z(P)-z>\varepsilon$ (mentre ovviamente l'altra implicazione è vera), ossia, è possibile che l'algoritmo abbia determinato una soluzione "accurata", ma che la valutazione superiore globale data dalla combinazione di rilassamento e branching non sia stata in grado di dimostrarlo. Ciò in effetti accade frequentemente in molte applicazioni, in quanto determinare "buone" soluzioni ammissibili è più facile che dimostrarne l'ottimalità (approssimata). Un caso particolare in cui questo accade è se l'algoritmo viene interrotto quando $z=-\infty$: non è stata determinata nessuna soluzione ammissibile, il che potrebbe essere perchè in effetti $X=\emptyset$, ma l'algoritmo non è stato in grado di dimostrare che questo sia vero.

7.2 Implementare un algoritmo enumerativo

Lo schema generale del paragrafo precedente include chiaramente algoritmi di efficienza pratica molto diversa. Infatti, come abbiamo già notato, lo schema include anche algoritmi completamente enumerativi, che esaminano cioè tutte le soluzioni ammissibili del problema. Si noti comunque che, per istanze di piccole dimensioni, questi algoritmi possono anche risultare efficienti, in quanto l'esplorazione dell'albero delle decisioni, senza lo sforzo ulteriore richiesto dalla computazione del rilassamento e dell'euristica ad ogni nodo, può essere organizzata in modo da generare e valutare ogni soluzione ammissibile a basso costo. L'efficienza pratica degli algoritmi di enumerazione implicita dipende da come sono implementati alcune componenti fondamentali dell'algoritmo, tra cui le principali sono:

- il *rilassamento* e l'*euristica*;
- la *strategia di visita* dell'albero delle decisioni;
- le eventuali *regole di dominanza* utilizzate;
- la *regola di branching* utilizzata;
- le eventuali operazioni di *pretrattamento* (*preprocessing*) dei dati utilizzate.

Nel seguito discuteremo ciascuna di queste componenti individualmente.

7.2.1 Rilassamento ed euristica

Le euristiche ed i rilassamenti sono già stati ampiamente discussi nei capitoli precedenti; in questo contesto ci limiteremo pertanto ad alcune considerazioni generali.

Come già rimarcato, il rilassamento e l'euristica devono essere quanto più possibile efficaci ed efficienti; ma, come discusso in precedenza, purtroppo tipicamente un aumento dell'efficacia va a discapito di una diminuzione dell'efficienza e viceversa. È quindi necessario operare un attento bilanciamento tra i due aspetti, sia per l'euristica che per il rilassamento. In molti casi pratici risulta essere di fondamentale importanza l'efficacia del rilassamento. Gli algoritmi di enumerazione implicita più efficienti sono spesso quelli che, grazie ad un rilassamento molto efficace anche se computazionalmente costoso, riescono a mantenere molto basso il numero di nodi dell'albero delle decisioni effettivamente visitati; spesso questi algoritmi fanno ampio uso di *tecniche poliedrali*, che saranno discusse nel paragrafo 7.4, per ottenere valutazioni superiori molto accurate. Una regola pratica, valida in molti casi, indica che si hanno discrete probabilità di ottenere un algoritmo ragionevolmente efficiente qualora il *gap relativo al nodo radice*, ossia $(\bar{z}(P)-\underline{z}(P))/\underline{z}(P)$, sia al massimo dell'$1-2\%$; usualmente la valutazione superiore è il punto critico che permette, o non permette, di ottenere gap sufficientemente bassi.

Naturalmente ciò deve essere considerato solamente come indicativo. In particolare, le considerazioni precedenti sono adeguate per il caso di problemi in cui la difficoltà consiste non tanto nel trovare una soluzione ottima, quanto nel dimostrarne l'ottimalita. Non tutti i problemi di OC ricadono in questa classe. Ad esempio, per i problemi decisionali (che possono essere considerati problemi di ottimizzazione in cui tutte le soluzioni ammissibili hanno lo stesso valore della funzione obiettivo) il problema è evidentemente quello di determinare una soluzione ammissibile, oppure dimostrare che non ne esistono; per questi problemi, il ruolo del rilassamento può solamente essere quello di determinare

precocemente che $\bar{X}' = \emptyset \rightarrow X' = \emptyset$ per evitare l'esplorazione di alcune aree dello spazio delle soluzioni. In effetti, si può ribaltare la regola precedentemente enunciata per dire che i problemi che si prestano ad essere risolti efficientemente da algoritmi di enumerazione implicita sono quelli in cui la difficoltà consiste fondamentalmente nel dimostrare l'ottimalita della soluzione. Per i problemi con "molti" vincoli e quindi "poche" soluzioni ammissibili, in cui la difficoltà è determinare una soluzione, possono infatti risultare più efficienti tecniche risolutive alternative (per quanto anch'esse utilizzino visite di strutture simili all'albero delledecisioni) basate su tecniche di inferenza logica, che rientrano sotto il nome generico di *programmazione logica con vincoli*. Per una descrizione delle idee base di queste tecniche, e circa la possibilità di utilizzarle in alternativa o in congiunzione con le tecniche di *programmazione matematica* qui descritte, si rimanda alla letteratura citata.

Terminiamo questa breve discussione con alcune considerazioni molto generali ed intuitive. Poiché euristica e rilassamento devono essere eseguite in principio ad ogni nodo dell'albero delle decisioni, è chiaramente fondamentale, insieme all'efficacia, la loro efficienza. È anche vero che le implementazioni reali permettono di bilanciare tra i due aspetti anche attraverso semplici strategie, quali quella di non eseguire l'euristica ad ogni nodo, ma solo ad alcuni, ad esempio ad intervalli regolari (tipo, ogni 10 o 100 nodi) oppure quando si verifichino particolari condizioni. Inoltre, come abbiamo visto possono esistere euristiche con un rapporto costo/prestazioni molto diverso, ossia "costose ma molto efficaci" o "economiche ma meno efficaci". Una strategia tipica è quindi quella di eseguire le prime poche volte, ad esempio solamente al nodo radice, e considerare solamente le seconde per l'esecuzione in tutti (o una frazione significativa de)i nodi. Ciò però richiede scelte potenzialmente arbitrarie, governate da *parametri algoritmici* (quali euristiche eseguire al nodo radice ed agli altri nodi, la frequenza con cui queste sono eseguite, ...) per i quali non esistono in generale criteri di scelta efficaci basati su argomentazioni teoriche. Questo aspetto in effetti non si limita alle scelte relative al rilassamento ed all'euristica, ma è generale per tutti gli aspetti discussi in questo paragrafo. Il punto fondamentale è che gli algoritmi B&B sono complessi, e la loro efficacia ed efficienza dipende in ultima analisi anche da scelte non ovvie relative all'implementazione di ciascuno degli elementi algoritmici, per i quali sono disponibili tipicamente più varianti, ed al bilanciamento tra lo sforzo computazionale speso in ciascuno dei passi. Ciò porta al fatto che le implementazioni reali di questi approcci hanno anche molte decine di parametri algoritmici, la scelta dei quali richiede estese *campagne di test*, ossia la sperimentazione su grandi gruppi di istanze e la compilazione di regole euristiche per la scelta dei parametri algoritmici in funzione delle caratteristiche dell'istanza da risolvere. Questa fase di *parameter tuning* è tipicamente costosa, sia in termini di tempo di calcolo che di sforzo degli esperti per decodificarne i risultati e compilare regole sperimentalmente efficienti. Poiché si tratta di approcci fondamentalmente basati su dati sperimentali, e non si considerazioni teoriche, possono essere utili tecniche *data driven*, quali inferenza statistica, tecniche di apprendimento automatico (*machine learning*) e di intelligenza artificiale. Ovviamente questi aspetti sono molto al di là del livello di queste dispense, e per essi si rimanda alla letteratura citata o a corsi più avanzati dell'area di Ricerca Operativa.

Si può comunque affermare che l'importanza di sviluppare tecniche risolutive sempre più efficienti per problemi "facili" risieda in buona parte nell'impatto che ciò ha sull'efficienza di tecniche enumerative che le utilizzano. In particolare, risulta spesso importante scegliere un rilassamento ed un'euristica che "collaborino", ossia tale che il primo sia in grado di sfruttare parte del lavoro svolto dalla seconda, o viceversa. Esempi di euristiche e rilassamenti di questo tipo sono:

- rilassamenti ed euristiche che sfruttano la risoluzione di uno stesso problema, si pensi ad esempio al rilassamento basato su (MS1-T) ed all'euristica "twice around MST" per il problema del commesso viaggiatore, oppure al rilassamento continuo ed all'euristica CUD per il problema dello zaino;
- come estensione del caso precedente, rilassamenti Lagrangiani e corrispondenti euristiche Lagrangiane;
- il rilassamento continuo (o Lagrangiano) e le tecniche di arrotondamento.

Infine, è utile osservare come risulti fondamentale, ai fini dell'efficienza complessiva dell'approccio, non tanto l'efficienza della soluzione di un singolo rilassamento/euristica, quanto il tempo complessivo utilizzato per risolvere tutti quelli richiesti dalla visita dello spazio delle soluzioni. L'osservazione fondamentale è che durante la visita vengono risolti una serie di problemi di ottimizzazione simili, ossia che differiscono per "pochi" dati: in questi casi, la conoscenza di una soluzione ottima per uno di questi problemi può costituire un valido aiuto nel determinare la soluzione ottima degli altri. Si consideri ad esempio un algoritmo *B&B* in cui ad ogni nodo dell'albero delle decisioni venga risolto un problema di *PL*, ad esempio il rilassamento continuo di una formulazione *PLI* del problema, e la regola di separazione sia implementata semplicemente fissando una variabile (questo esempio sarà discusso più in dettaglio nel seguito). In questo caso si possono applicare le *tecniche di riottimizzazione* per la *PL* discusse nel §3.3.4. Ciò tipicamente permette di rendere estremamente più efficiente l'approccio complessivo, in quanto il tempo necessario per risolvere i problemi di *PL* nei nodi interni dell'albero della visita è una frazione di quello necessario per risolvere il rilassamento continuo iniziale al nodo radice (questo può avere un impatto anche sulla scelta delle strategie di visita dell'albero delle decisioni, come discusso nel seguito). Può quindi accadere che algoritmi per la *PL* che siano efficienti nel risolvere "da zero" i problemi, ma non molto efficienti nel riottimizzare a seguito di piccoli cambiamenti nei dati del problema, risultino complessivamente meno efficaci, quando utilizzati all'interno di un approccio di enumerazione implicita, di algoritmi magari meno efficienti "da zero", ma molto efficienti in riottimizzazione; questo è ad esempio il caso dei *metodi del punto interno*, un'alternativa agli algoritmi del simplesso discussi in queste dispense. Ciò vale ovviamente anche per altri problemi di ottimizzazione. Possiamo concludere questo paragrafo ribadendo che, per l'efficienza complessiva di un approccio di enumerazione implicita, è necessario scegliere accuratamente rilassamento ed euristica, tenendo conto non solo dell'efficienza ed efficacia di ciascuno dei due separatamente, ma anche delle interazioni tra i due. Occorre inoltre selezionare accuratamente, tra gli algoritmi risolutivi disponibili, quelli più adatti all'uso nel contesto di un algoritmo enumerativo.

7.2.2 La strategia di visita

La strategia di visita dell'albero di enumerazione è fondamentalmente dettata dalla strategia di selezione del prossimo nodo da visitare dall'insieme Q (funzione $Next()$). Si distingue usualmente tra *visite topologiche* e *visite basate sull'informazione*.

Le visite topologiche scelgono il prossimo nodo da visitare unicamente sulla base della struttura topologica dell'albero delle decisioni; ciò corrisponde a strategie di selezione del nodo che dipendono unicamente dalla sua posizione in Q. Le due strategie di visita topologica più note sono quella a ventaglio, o *breadth-first*, corrispondente ad implementare Q come una fila (queue), e quella a scandaglio, o *depth-first*, corrispondente ad implementare Q come una pila (stack). Di queste, nel contesto degli algoritmi di enumerazione implicita può essere particolarmente utile la strategia depth-first nel caso in cui non si disponga di un'euristica (efficace). Poichè per potare i nodi attraverso la valutazione superiore è necessario disporre di una valutazione inferiore (tranne nel caso in cui il rilassamento sia vuoto), l'uso di una strategia depth-first può essere indicato in quanto porta la visita velocemente verso le foglie dell'albero delle decisioni, e quindi può consentire di generare velocemente soluzioni ammissibili. Le visite topologiche offrono alcuni vantaggi dal punto di vista dell'implementazione. In primo luogo sono semplici da realizzare ed il costo di gestione di Q è basso. In secondo luogo, la strategia depth-first si presta ad essere implementata in modo tale da mantenere molto basso il numero di nodi attivi nell'albero delle decisioni. Infatti, è possibile, durante l'operazione di separazione, evitare di generare tutti i figli per inserirli in Q; si può invece iniziare l'esame del figlio appena generato, rimandando la generazione dei suoi ulteriori fratelli (se ne ha) al momento in cui sia terminata la visita del sottoalbero corrispondente. In questo modo, ad ogni istante saranno attivi solo un numero di nodi non superiore all'altezza dell'albero (sono attivi anche i nodi parzialmente visitati, ossia per cui la visita non ha ancora terminato di esaminare tutti i figli). Poichè può essere necessario, nelle implementazioni, memorizzare molta informazione in corrispondenza di ogni nodo attivo dell'albero delle decisioni, ciò può consentire di risparmiare molta memoria, specialmente nel caso in cui i nodi dell'albero delle decisioni abbiano molti figli. Inoltre, in questo modo i problemi esaminati in sequenza

mentre si scende in profondità nell'albero delle decisioni—ma non necessariamente quando si risale, ossia si effettua un *backtrack*—sono molto simili tra loro, il che può essere utile qualora si utlizzino tecniche di riottimizzazione (si veda il paragrafo precedente) nel rilassamento e/o nell'euristica. Infine, è più facile mantenere l'informazione relativa ad ogni singolo sottoproblema in "forma differenziale", ossia memorizzare, per ogni nodo dell'albero delle decisioni, solamente le differenze tra il problema corrispondente al nodo ed il problema corrispondente al padre: ciò può consentire di risparmiare ulteriormente memoria. Si noti che se si utilizza una visita depth-first può essere cruciale selezionare oculatamente l'ordine con cui vengono esaminati i figli di ogni nodo: infatti, generare un figlio prima di un altro significa esplorare tutto il sottoalbero corrispondente al primo prima del sottoalbero corrispondente al secondo, per cui è necessario cercare di indirizzare la ricerca prima verso i figli "più promettenti".

Le regole di visita basate sull'informazione utilizzano informazione sui nodi in Q per definire il prossimo nodo da estrarre; in altri termini, corrispondono ad implementare Q come una coda di priorità. La strategia con informazione più usata è la *best-first*, in cui ad ogni nodo viene associato il valore della valutazione superiore prodotta dal corrispondente rilassamento e viene selezionato il nodo con valore maggiore, che corrisponde al sottoalbero "più promettente", ossia nel quale si dovrebbero avere maggiori probabilità di incontrare una soluzione ottima. Si noti che ciò richiede una modifica dell'algoritmo 7.1, in quanto il rilassamento deve essere risolto immediatamente dopo la generazione di un nodo e prima del suo inserimento in Q; naturalmente, il nodo non viene inserito se risulta potato dalla valutazione superiore, ma il controllo deve comunque essere ripetuto quando il nodo viene estratto da Q, in quanto la valutazione inferiore potrebbe essere cambiata nel frattempo. La strategia best-first è molto usata in quanto solitamente riesce ad indirizzare la visita verso le zone "più promettenti" dell'albero delle decisioni, meglio di quanto non accada utilizzando strategie topologiche. Inoltre, il Lemma 7.2 mostra che il nodo attivo avente il maggior valore della valutazione superiore associata è quello che determina la valutazione superiore corrente per l'intero algoritmo; visitare per primo quel nodo corrisponde dunque a cercare di far diminuire la valutazione superiore, il che può rendere più efficiente l'approccio. Per contro, la strategia best-first è più complessa da implementare, e tipicamente accade che vengano esaminati in sequenza nodi "lontani" nell'albero delle decisioni, che corrispondono a problemi "molto diversi" tra loro; questo può rendere meno efficaci, o più onerose da implementare, le tecniche di riottimizzazione. Pertanto nelle implementazioni efficienti si usano anche tecniche miste. Ad esempio, è possibile effettuare una visita tendenzialmente depth-first dell'albero delle decisioni, ma controllando il valore del gap relativo $(\bar{z}(P') - \underline{z}(P')) / \underline{z}(P')$ dei nodi generati: se il valore diminuisce "abbastanza" discendendo nell'albero delle decisioni (ad esempio, almeno dello 0,1% per ogni decisione presa) si continua con la visita depth-first, altrimenti si usa una qualche strategia con informazione per selezionare un nuovo nodo "promettente" da cui continuare la ricerca.

7.2.3 Regole di dominanza

Un modo ulteriore per "potare" nodi dell'albero delle decisioni, rispetto a quelli indicati nello schema generale, consiste nell'individuare ed applicare regole di dominanza. Si consideri un nodo attivo (P') tale che esista un diverso nodo (P''), attivo o già esaminato, per il quale si possa affermare con sicurezza che $z(P') \leq z(P'')$: in questo caso si dice che *(P'') domina (P')*. È chiaro che non è necessario che l'algoritmo esplori il nodo (P') purchè venga esplorato (P''); (P') può quindi essere *potato per dominanza*. Si noti che la potatura basata sulla valutazione superiore è un caso particolare della potatura basata sulla dominanza: in tale caso, infatti, nella visita è stato generato un nodo (P'') tale che $z \leq z(P'')$, e quindi la relazione $\bar{z}(P') \leq z$ implica che (P') sia dominato da (P''). Il concetto si estende in modo ovvio alla *ε-dominanza*, ossia $z(P') \leq z(P'') + \varepsilon$, che garantisce che esaminare (P') non potrebbe comunque migliorare il valore di z di una quantità superiore a ε una volta esaminato (P'').

Una relazione tra (P') e (P'') che permetta di affermare che (P'') domina (P') viene detta *regola di dominanza*. Per l'osservazione precedente, la normale "potatura" dei nodi basata sul valore della loro valutazione superiore è quindi una particolare regola di dominanza. Esistono diversi modi possibili per costruire regole di dominanza tra sottoproblemi di uno stesso problema di *OC*. Alcune regole di

dominanza verranno illustrate nel contesto degli algoritmi di programmazione dinamica, discussi nel §7.5. Pertanto, per il momento ci limitiamo a notare come il principale problema dell'applicazione delle regole di dominanza all'interno di un algoritmo di enumerazione implicita sia la necessità di verificare la regola tra un nodo (P'), ad esempio quello appena estratto da Q, e tutto l'insieme dei nodi attivi e dei nodi già visitati: ciò può risultare molto oneroso. Infatti, le regole di dominanza si sfruttano di solito quando sia possibile organizzare la visita dell'albero delle decisioni in modo tale che il loro controllo sia effettuabile in modo efficiente, come accade, appunto, nel caso degli algoritmi discussi nel §7.5.

7.2.4 Regole di branching

Per uno stesso problema di *OC* si possono usare diverse regole di branching. La selezione della regola di branching può avere un forte impatto sull'efficienza di un algoritmo di enumerazione implicita, pertanto la scelta delle regola deve essere compiuta con oculatezza. In generale, affinchè l'algoritmo termini occorre solamente che la regola sia *completa*, ossia tale che $X' = X'_1 \cup \ldots \cup X'_k$, e che ciascuno dei sottoinsiemi X'_i sia un sottoinsieme proprio di X'. In generale, però, le regole di branching possono avere proprietà che le rendono particolarmente attraenti, quali:

1. partizionare lo spazio delle soluzioni, ossia garantire che $X'_i \cap X'_j = \emptyset$ comunque scelti $i \neq j$;
2. equisuddividere X', ossia garantire che la cardinalità di tutti i sottoinsiemi X'_i sia (approssimativamente) uguale;
3. garantire che la soluzione ottima del rilassamento utilizzato per X' *non* sia ammissibile per i rilassamenti utilizzati per ciascun X'_i, in modo tale da rendere possibile una decrescita stretta della valutazione superiore;
4. essere "compatibile" con il rilassamento e l'euristica utilizzata, ossia non distruggere la struttura che li rende efficienti;
5. generare "pochi" figli per ogni nodo, ossia tipicamente un numero costante che non dipende dalla dimensione del problema.

Discutiamo adesso brevemente le proprietà enunciate.

La proprietà 4., ossi la compatibilità della regola di branching con il rilassamento e l'euristica è fondamentale: prendere decisioni su un problema significa modificarlo, e, come abbiamo visto, modificare un problema di *OC* "facile" può renderlo "difficile". Illustriamo questa problematica con un esempio. Si consideri il problema del cammino minimo vincolato introdotto nel §6.4. Un ovvio rilassamento per questo problema è il rilassamento per eliminazione di vincoli, o quello Lagrangiano, che produce un problema di cammino minimo. Si consideri quindi la regola di branching "standard" che prende decisioni su una singola variabile del problema, ossia che seleziona un arco (i, j) e definisce X'_0 come l'insieme di tutti i cammini in X' che non contengono (i, j) e X'_1 come l'insieme di tutti i cammini in X' che lo contengono. Il rilassamento da risolvere ad un generico nodo dell'albero delle decisioni richiede di determinare un cammino di costo minimo che non contiene un certo insieme di archi, e che contiene un altro insieme di archi. Mentre la prima richiesta non ha alcun impatto sulla struttura del problema—è sufficiente eliminare tali archi dal grafo sul quale si calcola il cammino minimo—la seconda richiesta ne altera profondamente la difficoltà: determinare un cammino di costo minimo tra tutti quelli che contengono un insieme prefissato di archi è un problema $\mathcal{NP}$-arduo. Il problema può essere risolto con una complessità polinomiale nella dimensione del grafo, ma esponenziale nella cardinalità dell'insieme di archi fissati.

Esercizio 7.4. Si dimostri l'affermazione precedente.

Di conseguenza, il costo di risolvere il rilassamento può aumentare in modo esponenziale con la profondità dei nodi dell'albero, il che è particolarmente problematico perché anche il numero di nodi aumenta in tale modo. Quindi, questa regola di branching non può essere utilizzata in combinazione con questo rilassamento. Vedremo infatti nel §7.3.3 una regola di branching alternativa costruita ad-hoc per questo problema e questo particolare rilassamento del problems.

Fortunatamente, l'esempio precedentemente illustrato è l'eccezione piuttosto che la regola: in moltissimi casi è possibile adattare facilmente euristiche e rilassamenti affinchè lavorino su problemi in cui sono state prese decisioni. Ad esempio, per il problema dello zaino è immediato adattare il rilassamento e le euristiche viste al caso in cui alcune variabili sono fissate a 0 (basta eliminare i corrispondenti oggetti dalla lista) o a 1 (basta eliminare i corrispondenti oggetti dalla lista diminuendo opportunamente la capacità dello zaino). Ciò corrisponde al fatto che i problemi di OC molto spesso hanno la proprietà di *auto-riducibilità* (cf. §4.3), per cui il problema (P) in cui alcune variabili sono fissate può essere riformulato come un'istanza diversa (più piccola) dello stesso problema.

Esercizio 7.5. Si discuta se e come sia possibile adattare le euristiche ed i rilassamenti presentati per (CMST) affinché possano essere utilizzati per lavorare su un'istanza del problema sulla quale sono state prese decisioni, ossia alcuni lati appartengono/non appartengono all'albero ottimo (suggerimento: occorre "aggregare" in modo appropriato vertici del grafo).

Esercizio 7.6. Si discuta come adattare il rilassamento (MS1-T) per il problema (TSP) al caso in cui si sia deciso che alcuni lati fanno parte del ciclo Hamiltoniano ed altri non ne fanno parte (suggerimento: è sufficiente eliminare dal grafo i secondi ed "aggregare" i vertivi corrispondenti ai primi, avendo però cura di operare ricorsivamente e rispettando la struttura dei cammini creati dai lati già fissati).

Per quanto riguarda 1., una regola di branching che partizioni lo spazio delle soluzioni garantisce che esista un solo cammino nell'albero delle decisioni per ogni soluzione. Quando ciò non accade, come nell'esempio in Figura 7.2, la stessa soluzione può essere visitata più volte, il che chiaramente corrisponde ad uno spreco di risorse. Ciò nonostante, in certi casi regole di questo tipo vengono comunque utilizzate perchè hanno altre proprietà che le rendono complessivamente preferibili.

Per quanto riguarda 2., una regola di branching che equisuddivida gli insiemi garantisce che il corrispondente albero delle decisioni sia "bilanciato", ossia che la lunghezza di tutti i cammini dalla radice alle foglie sia approssimativamente uguale, e quindi "relativamente poco profondo". Ad esempio, l'albero delle decisioni per un problema con n variabili binarie costruito come in Figura 7.1 ha altezza n, anche se ha 2^n foglie. Si consideri, come esempio estremo di una regola che costruisce alberi "sbilanciati", quella che costantemente selezioni un elemento $\bar{x} \in X'$ e costruisca due figli di X', uno corrispondente a $X'_1 = \{ \bar{x} \}$ e l'altro corrispondente a $X'_2 = X' \setminus X'_1$: chiaramente il corrispondente albero delle decisioni ha una profondità pari alla cardinalità di X, ossia tipicamente esponenziale nella dimensione del problema. Equisuddividere fa in modo che, in linea di massima, tutti i problemi allo stesso livello dell'albero delle decisioni siano "ugualmente difficili" e "significativamente più facili" di quelli del livello precedente. Si pensi ad esempio alla regola di branching "sbilanciata" appena discussa: uno dei due figli corrisponde ad un problema estremamente facile, l'altro ad un problema che ha all'incirca le stesse soluzioni ammissibili di quello originario, e quindi è in generale altrettanto difficile; o anche di più, perché in generale non è ovvio risolvere un problema di ottimizzazione nel quale siano eliminati alcuni elementi specifici dalla regione ammissibile. Infatti, una tale regola sarebbe oltretutto difficilmente compatibile con un qualsiasi rilassamento ed euristica.

Un esempio più realistico di regola "non bilanciata" è il seguente: si consideri un problema di Programmazione 0/1 che contenga un vincolo "tipo semiassegnamento" (normalmente chiamato *SOS*, da Special-Ordered Set)

$$\sum_{i \in S} x_i = 1 \ .$$

Chiaramente, in questo caso è possibile utilizzare la regola di branching "standard" che seleziona una delle variabili e la fissa a 0 oppure ad 1; tale regola risulta però sbilanciata. Infatti, per via del vincolo di semiassegnamento fissare ad 1 una variabile x_i corrisponde di fatto a fissare a 0 tutte le variabili x_j per $j \in S$ (si veda il §7.2.5); per contro, fissare x_i ha 0 non ha, in generale, un analogo effetto. Di conseguenza, X'_0 contiene tutte le soluzioni di X' in cui una variabile è fissata a 0, mentre X'_1 contiene tutte le soluzioni di X' in cui $|\, S \,| - 1$ variabili sono fissate a 0. X'_0 contiene quindi probabilmente più soluzioni di X'_1. Una strategia "bilanciata" per lo stesso problema potrebbe essere implementata come segue: si partiziona S in due sottoinsiemi S_1 ed S_2, di cardinalità approssimativamente uguale, e si costruiscono i due insiemi X'_1 ed X'_2 in cui sono fissate a 0 rispettivamente tutte le variabili di S_1 ed S_2.

Questa regola—o sue generalizzazioni per vincoli che definiscono altri tipi di Special Ordered Set—si dimostra normalmente migliore della strategia "standard". Si noti che questa strategia partiziona lo spazio delle soluzioni: qualsiasi soluzione ammissibile del problema ha $x_i = 1$ per un qualche indice $i \in S$, quindi la soluzione appartiene esattamente ad uno tra X'_1 ed X'_2. Ciò non sarebbe vero nel caso di vincoli di tipo $\sum_{i \in S} x_i \leq 1$.

L'equisuddivisione dei sottoproblemi è particolarmente importante qualora si usi una strategia depth-first, che visita un sottoalbero finchè non ne ha completato l'esplorazione. Si pensi, come caso estremo, alla strategia "totalmente sbilanciata" in cui al nodo radice viene selezionata in X'_1 l'unica soluzione ottima del problema, ma l'algoritmo inizia a visitare X'_2.

Per quanto riguarda 3., garantire che la soluzione ottima del rilassamento utilizzato per X' *non* sia ammissibile per i rilassamenti utilizzati per ciascun X'_i è necessario se si vuole avere una speranza che l'operazione di branching produca una decrescita stretta della valutazione superiore associata al nodo, e quindi possibilmente della valutazione superiore complessiva, nel caso—tipico—in cui la funzione obiettivo non cambi. Sia infatti $\bar{x} \in \bar{X}'$ la soluzione ottima del rilassamento al nodo X', e supponiamo per semplcità che tutti i rilassamenti usino la funzione obiettivo $c()$ del problema originale (P): se $\bar{x} \in \bar{X}'_i$ per qualche i, allora ovviamente $\bar{x}$ è una soluzione ottima del rilassamento al nodo $\bar{X}'_i$.

Esercizio 7.7. Si dimostri l'affermazione precedente.

Di conseguenza, la valutazione superiore $\bar{z}(P') = c(\bar{x})$ di $z(P')$, ottenuta risolvendo il rilassamento al nodo X', non viene migliorata dal branching: infatti si ha $\bar{z}(P'_i) = c(\bar{x})$ per almeno un i, e quindi $\max_i \bar{z}(P'_i) \geq c(\bar{x})$. Per questo è importante che la regola di branching renda inammissibile la soluzione ottima del rilassamento. In effetti, questo è esattamente il modo in cui normalmente si costruisce una regola di branching: si esamina la soluzione ottima del rilassamento e si prendono decisioni sulle variabili in modo da renderla inammissibile. Ciò sarà illustrato più in dettaglio nel seguito.

Infine, una regola di branching che generi "pochi" figli (proprietà 5.) evita alcuni problemi. Innanzitutto, una regola che generi, ad esempio, solo due figli per ogni nodo è più semplice da implementare. Viceversa, una regola che genera "molti" figli per ogni nodo può essere complessa da implementare, e può risultare costosa sia in termini di memoria che di tempo. Infatti, se ogni nodo genera "molti" figli l'insieme Q cresce rapidamente; quindi cresce la quantità di memoria necessaria per memorizzare le descrizioni dei nodi attivi, e le operazioni su Q possono diventare costose. Inoltre, generare molti figli ha intrinsecamente un costo che può essere rilevante; come esempio estremo si pensi ad una regola che genera, per ogni nodo, un numero di figli esponenziale nella dimensione del problema. Questo è in un certo senso il requisito "duale" rispetto a quello dell'equisuddivisione degli insiemi (proprietà 2.): mentre quello garantisce che l'albero non sia troppo profondo, ossia che esista un cammino "ragionevolmente corto" dalla radice a ciascuna soluzione, il criterio relativo ai pochi figli garantisce che l'albero non sia troppo poco profondo, ossia che lo sforzo necessario per esaminare tutti i figli di un dato nodo sia limitato.

7.2.5 Preprocessing

Le operazioni di "pretrattamento" (preprocessing) del problema sono tutte quelle manipolazioni sui dati del problema che consentono di ottenere rapidamente informazione utile a rendere più efficiente il processo risolutivo. Spesso le operazioni di preprocessing sfruttano proprietà apparentemente banali ed ovvie del problema; alcuni semplici esempi sono:

- nel problema dello zaino, qualsiasi oggetto i con $a_i > b$ non può far parte della soluzione ottima, ossia si può porre $x_i = 0$;
- nel problema dello zaino, per qualsiasi oggetto i con $c_i \leq 0$ ed $a_i \geq 0$ si può porre $x_i = 0$, e, viceversa, per qualsiasi oggetto i con $c_i \geq 0$ ed $a_i \leq 0$ si può porre $x_i = 1$;
- nel problema del (CMST), nessun lato $\{ i , j \}$ tale che $c_{ij} \geq \max\{ c_{ri} , c_{rj} \}$ (dove r è la radice) può far parte dell'albero ottimo;

- nel problema del (CMST), se un nodo i ha peso $b_i = Q$ (o, più in generale, $b_i + \max_{j \neq i}\{ b_j \} > Q$) il lato $\{ r , i \}$ deve necessariamente far parte dell'albero ottimo.

Può apparire sorprendente che sia necessario verificare questo tipo di condizioni; in prima approssimazione sembrerebbe ragionevole che, ad esempio, non venissero forniti in input ad un problema dello zaino oggetti che da soli non entrano nello zaino. Ciò però non sempre accade, per almeno due motivi:

1. l'istanza del problema può essere stata generata a partire da dati complessi con una procedura complessa, e può essere di dimensioni tali da non rendere possibile una sua verifica puntuale da parte di un esperto;
2. l'istanza del problema può essere stata generata a partirecda un'istanza diversa, tipicamente prendendo alcune decisioni o rilassando alcuni vincoli.

La seconda osservazione è particolarmente importante nel contesto degli algoritmi enumerativi, specialmente nei nodi diversi dalla radice. Infatti, le procedure di preprocessing applicate al problema (P') corrispondente al nodo dell'albero delle decisioni attualmente visitato possono essere viste come un modo per *sfruttare implicazioni logiche delle decisioni prese* (nel cammino dell'albero delle decisioni che raggiunge il nodo) per *estrarre informazione utile su* (P'). Ad esempio, nel caso dello zaino, denotando con S l'insieme degli indici delle variabili fissate ad 1 in (P'), si ha che qualsiasi oggetto con $a_i > b - \sum_{j \in S} a_j$ non può più essere inserito nello zaino, anche se $a_i \leq b$ e quindi l'oggetto è, in linea di principio, un input "ragionevole" per l'istanza: è solamente quando vengono prese alcune decisioni che si può "dedurre" che tale oggetto non è più utilizzabile. Analogamente, nel caso del (CMST) fissare alcuni lati come facenti parte dell'albero ottimo richiede di aggregare i sottoalberi così formati trasformandoli in nodi (si veda l'Esercizio 7.5): il peso di tali nodi (dato dalla somma dei pesi dei nodi che compongono il sottoalbero) può quindi diventare abbastanza grande da permettere di applicare le operazioni di preprocessing precedentemente indicate.

Esercizio 7.8. Per tutti i problemi di *OC* incontrati finora si discutano opportune operazioni di preprocessing.

Le operazioni di preprocessing sono dunque uno dei modi possibili per estrarre efficientemente informazione sul problema, in particolare sui sottoproblemi corrispondenti ai nodi durante la visita dell'albero delle decisioni. In effetti questa può essere una delle operazioni che contribuiscono in modo decisivo all'efficienza degli approcci enumerativi. Questo aspetto è sfruttato in modo estensivo in particolare dalle tecniche di *programmazione logica con vincoli* alle quali si è già brevemente accennato, alternative (ma in realtà equivalenti e complementari) a quelle di *OC* trattate in queste dispense. In tali approcci, il preprocessing (nei nodi interni dell'albero delle decisioni) viene normalmente indicato come *propagazione*: l'idea è che ogni volta che si prende una decisione questa si "propaghi logicamente" ad altre decisioni, il che permette di prenderne più di una (e potenzialmente anche molte) ad ogni operazione di branching. Questo è particolarmente naturale da sfruttare nella programmazione logica con vincoli, in quanto il problema è dato in partenza come una formula del calcolo logico e le operazioni di inferenza logica sono una parte fondamentale del motore algoritmico. In effetti per tali approcci vengono definiti vincoli logici complessi, utili per le applicazioni, per i quali le operazioni di propagazione possono essere svolte in modo particolarmente efficiente; un esempio classico è quello di `alldifferent`, che corrisponde alla richiesta che ciascun elemento i di un certo insieme I venga assegnato ad un diverso elemento j di un insieme J, ossia sostanzialmente un vincolo di assegnamento. Non possiamo entrare nei dettagli di tali metodi in queste dispense, ma riportiamo ancora una volta il fatto che essi possano essere particolarmente efficienti per problemi di *OC* che hanno "poche" soluzioni ammissibili, quindi "difficili da trovare". In questo contesto, prendere decisioni può facilmente portare a sottoinsiemi X_i vuoti: il preprocessing (propagazione) può permettere di stabilire questo fatto in modo efficiente, potando rapidamente i nodi corrispondenti e rendendo quindi più veloce l'approccio.

In un certo senso, anche la computazione della valutazione superiore o il fissaggio per costi ridotti (si veda il §6.1.2.2) sono operazioni di preprocessing, in quanto consentono di ottenere informazione sul sottoproblema (sicuramente non contiene una soluzione ottima, sicuramente alcune variabili devono

essere fissate in certi modi). Essi sono però fondamentalmente basati sul concetto di rilassamento, che inizialmente non veniva utilizzato in modo estensivo negli approcci di programmazione logica con vincoli. Ciò ha contribuito al fatto che il B&B, basato sui rilassamenti, sia generalmente considerato preferibile per quei problemi di *OC* con "tante soluzioni", in cui la difficoltà è principalmente di determinare la soluzione ottima, ed ancor più di *dimostrarne l'ottimalità*, come opposti ai problemi con "poche soluzioni" in cui invece la programmazione logica con vincoli eccelle. Nei fatti, però, i due approcci sono largamente equivalenti, e le idee che hanno avuto successo in uno di essi possono essere (e sono state) adottate dall'altro. Da una parte, quindi, i moderni approcci di programmazione logica con vincoli fanno largo uso di rilassamenti; dall'altra l'importanza delle tecniche di preprocessing è ormai chiaramente riconosciuta anche nelle implementazioni degli approcci B&B.

Infatti, uno degli ambiti in cui le operazioni di preprocessing sono state maggiormente sviluppate è quello dei solutori "general purpose" per la *PLI*. Normalmente questi (complessi) codici risolutivi usano solutori generali ed efficienti per la *PL* per risolvere il rilassamento continuo del modello, e saranno discussi più in dettaglio nel §7.3.1. Poichè non hanno nessuna informazione sulla struttura combinatoria del problema da risolvere, questi algoritmi si concentrano sui vincoli della rappresentazione *PLI* e tentano di ricavare informazione da questi. Si consideri ad esempio un singolo vincolo lineare $\sum_{i=1}^{n} a_i x_i \leq b$ per cui tutte le variabili x_i siano binarie (ossia un *vincolo di zaino*). Chiamiamo $P = \{\, i \,:\, a_i > 0 \,\}$ e $N = \{\, i \,:\, a_i < 0 \,\}$, e definiamo $\underline{b} = \sum_{i \in N} a_i$. Per qualsiasi $j \in P$, se si ha $a_j + \underline{b} > b$ allora sicuramente deve essere $x_j = 0$. Questa è un'ovvia generalizzazione delle operazioni di preprocessing viste nel caso del problema dello zaino; come discusso in precedenza, ed illustrato nel §7.3.1, questo tipo di relazione può non essere vera per il vincolo nella formulazione iniziale del problema, ma diventarlo dopo che sono state prese alcune decisioni. In generale, si può ottenere da queste tecniche informazione sul problema in forma diversa, e più generale, del fissare variabili. Con la notazione precedente, ad esempio, si ha:

- se $a_i + a_j + \underline{b} > b$, allora il vincolo $x_i + x_j \leq 1$ è soddisfatto da tutte le soluzioni ammissibili del problema;
- se $a_i + (\underline{b} - a_j) > b$ per $j \in N$, allora il vincolo $x_i \leq x_j$ (corrispondente all'implicazione logica "$x_i = 1 \Longrightarrow x_j = 1$") è soddisfatto da tutte le soluzioni ammissibili del problema.

Queste relazioni permettono di derivare vincoli lineari che sono "logicamente implicati" dai vincoli originali e dall'integralità delle variabili. Questi vincoli, se aggiunti alla formulazione, possono migliorare la qualità della valutazione superiore ottenuta dal rilassamento continuo.

Esempio 7.3. Preprocessing per la *PLI*
Si consideri ad esempio il problema

$$\max\{\, 2x_1 + x_2 \,:\, 27x_1 + 14x_2 \leq 40 \;\;,\;\; x_i \in \{\, 0\,,\, 1 \,\} \quad i = 1, 2 \,\} \;\;.$$

In questo caso $P = \{\, 1\,,\, 2 \,\}$, $N = \emptyset$, $\underline{b} = 0$; applicando la prima relazione si ha che il vincolo $x_1 + x_2 \leq 1$ deve essere soddisfatto da tutte le soluzioni ammissibili del problema. Aggiungere il vincolo alla formulazione non cambia quindi l'insieme delle soluzioni ammissibili, ma cambia il rilassamento continuo. Infatti, è facile verificare che la soluzione ottima del rilassamento continuo senza il vincolo è $[\, 40/27\,,\, 0 \,]$, che corrisponde ad una valutazione superiore sull'ottimo del problema di $80/27$; invece aggiungendo il vincolo la soluzione ottima del rilassamento continuo diviene $[\, 1\,,\, 0 \,]$, che è intera e quindi ottima per il problema originario.

Esercizio 7.9. Si verifichino le affermazioni precedenti disegnando la regione ammissibile del rilassamento continuo del problema nell'esempio con e senza il vincolo $x_1 + x_2 \leq 1$.

Esercizio 7.10. Si estendano le relazioni precedenti al caso di insiemi di variabili di cardinalità maggiore di due, discutendo come siano legate all'approccio visto nell'Esempio 4.7.

Queste operazioni di preprocessing sono quindi un caso particolare delle *tecniche poliedrali* discusse al §7.4, in quanto, come l'esempio precedente mostra chiaramente, cercano di ottenere una più accurata descrizione dell'inviluppo convesso dell'insieme delle soluzioni intere del problema. Per ulteriori dettagli sulle tecniche di preprocessing sviluppate la *PLI* si rimanda alla letteratura citata.

7.2.6 Parallelismo nel B&B

Per terminare questa sezione notiamo che gli algoritmi di enumerazione implicita si prestano bene ad essere implementati su architetture parallele; a parte l'uso di algoritmi paralleli per risolvere euristica e rilassamento, che dipende dal particolare problema affrontato, è chiaramente sempre possibile effettuare in parallelo la visita dell'albero delle decisioni, valutando più nodi attivi contemporaneamente, uno su ciascun processore. In questo caso Q è una struttura dati distribuita, e le operazioni di aggiornamento della valutazione inferiore z richiedono comunicazione tra i vari processori. La quantità di dati da comunicare è però limitata e non ci sono sostanziali necessità di sincronizzazione tra i vari processi, che possono procedere in modo sostanzialmente asincrono. In principio, quindi, sviluppare versioni parallele del B&B non è eccessivamente complesso, ed infatti sono disponibili diversi framework software generali che implementano tali concetti.

In effetti, in linea di principio il B&B è uno degli algoritmi che possono essere considerati *paralleli in modo imbarazzante* (embarrassingly parallel), ossia che possono facilmente fare uso di un gran numero di processori diversi. Per vederlo si consideri per semplicità un problema di OC con n variabili binarie per il quale l'albero delle decisioni sia del tipo "semplice" mostrato nell'Esempio 7.1. Si supponga che l'implementazione sia tale per cui quando un processore, assegnato ad un nodo dell'albero delle decisioni, effettua l'operazione di branching esso passi immediatamente ad esaminare uno dei due figli, ma possa far partire un nuovo processo, in esecuzione su un diverso processore, dedicato ad esaminare l'altro. Ovviamente il numero di processori attivi aumenterebbe in principio in modo esponenziale, ma se fossero disponibili abbastanza processori il problema potrebbe essere risolto in tempo polinomiale (se tale fosse il costo di processare un singolo nodo, ma questo è sempre possibile scegliendo rilassamenti ed euristiche "abbastanza semplici" come discusso in precedenza). In generale, quindi, un algoritmo B&B può essere in grado di far uso di un numero arbitrariamente grande di processori in parallelo (per n sufficientemente grande).

Ovviamente, nessuna architettura parallela che sia attualmente possibile immaginare (neanche, per quel che si capisce, utilizzando computer quantistici) potrebbe permettere la disponibilità di un numero di processori tale da rendere fattibile un tale processo risolutivo per n anche solo pari a un centinaio: poiché $2^{10} \approx 1000$, $2^{100} \approx 1000^{10} = 10^{30}$, un numero astronomico. L'osservazione rende però l'idea del fatto che sia concettualmente semplice utilizzare attivamente un numero arbitrario di processori in parallelo per eseguire un approccio B&B.

È bene però osservare che "concettualmente semplice" non è la stessa cosa di "facile in pratica". Implementare un B&B in parallelo ha un certo numero di aspetti assolutamente non banali, e non solo dal punto di vista del fatto che la realizzazione di algoritmi con un elevato grado di parallelismo è comunque un compito complesso. Anche se non è ovviamente possibile entrare in dettaglio in queste dispense, riportiamo uno degli aspetti che è più semplice descrivere: il processo precedentemente accennato, in cui il numero di processori aumenta esponenzialmente con l'altezza dell'albero delle decisioni, ha comunque una fase iniziale di "ramp up" in cui sono attivi solo un piccolo numero di processori. In particolare, in assenza di implementazioni parallele del rilassamento e delle euristiche (che non sono ovvie), al nodo radice è attivo un solo processore. Questo è particolarmente problematico in quanto, per molti motivi (alcuni dei quali sono stati descritti) le implementazioni efficienti spendono nel nodo radice una quantità di tempo molto superiore—anche di ordini di grandezza—rispetto a quello che utilizzano nei nodi intermedi. Ciò significa che in pratica un algoritmo di B&B parallelo possa soffrire di una fase iniziale, anche di lunga durata, in cui non è possibile (o comunque facile) fare uso efficiente di un gran numero di processori.

In effetti, l'esperienza mostra che l'uso di tecniche di calcolo parallelo per velocizzare gli algoritmi B&B può certamente avere un impatto, in particolare nel presente momento in cui l'aumento della capacità di calcolo si sostanzia principalmente nell'aumento del numero di "cores" disponibili nei processori (anziché, come è accaduto a lungo in passato, nell'aumento esponenziale della potenza di ciascuno di essi). Le implementazioni attuali possono fare un uso relativamente efficiente di un numero limitato di "cores", come 16 o 32, ma difficilmente riescono a farlo per numeri molto superiori. In altri termini, non si può fare affidamento sulla "forza bruta" dell'aumento della potenza di calcolo disponibile per

rendere efficiente un algoritmo B&B che non sia stato adeguatamente progettato, mentre è possibile rendere più veloce (intorno ad un ordine di grandezza) un algoritmo che sia già efficiente.

Questo è un trend che appare essere vero in generale: alcuni studi hanno mostrato che mentre l'efficienza dei computer è aumentata di un fattore 20 in 20 anni, nello stesso periodo l'efficienza dei solutori per problemi di *OC* è aumentata qualcosa come un fattore 1000. Quest'ultimo dato è in effetti sottostimato, nel senso che molti problemi che sarebbero stati totalmente irresolubili 20 anni fa, qualsiasi fosse la potenza di calcolo utilizzata, sono oggi risolvibili con facilità. In questo senso si può affermare che la speranza di rendere risolubili problemi di *OC* che attualmente non lo sono risiede fondamentalmente nello sviluppo di algoritmi più sofisticati ed efficienti piuttosto che nel puro aumento della potenza di calcolo disponibile. Per ulteriori approfondimenti su questi affascinanti temi si rimanda alla letteratura citata ed a corsi più avanzati nell'area di Ricerca Operativa.

7.3 Esempi di algoritmi enumerativi

In questa sezione discuteremo alcuni esempi di algoritmi di enumerazione implicita applicati a specifici problemi di *OC*. Per ogni problema discuteremo varie possibilità per implementare i vari aspetti cruciali dell'algoritmo (rilassamento ed euristica, regola di branching, strategia di visita, ...) esaminando i prò ed i contro di alcune alternative diverse. Mostreremo poi il funzionamento dell'algoritmo su un esempio.

7.3.1 Il problema dello zaino

Per il problema dello zaino possiamo implementare un algoritmo di enumerazione implicita caratterizzato dai seguenti componenti:

- rilassamento continuo;
- euristica greedy CUD;
- branching "standard" sull'unica variabile frazionaria;
- visita depth-first;
- preprocessing corrispondente nel fissare $x_i = 0$ se $a_i > \bar{b}$, dove $\bar{b}$ è la capacità residua dello zaino che tiene in conto delle variabili fissate a 1, e fissaggio basato sui costi ridotti.

La scelta del rilassamento continuo e dell'euristica CUD è dettata dal fatto che sono entrambe efficienti e "collaborano"; infatti, una volta ordinati gli oggetti dello zaino per costo unitario non crescente all'inizio dell'algoritmo, entrambe possono essere risolti in $O(n)$, e riutilizzando per l'euristica parte del lavoro svolto nel rilassamento. Il rilassamento continuo permette anche il fissaggio basato sui costi ridotti. Poichè la soluzione del rilassamento è intera tranne per la variabile "critica" h (la prima variabile a non entrare completamente nello zaino) al più, ed ottima se x_h è intera, l'ovvia regola di branching è quella di fissare x_h rispettivamente a 0 ed 1 nei figli del nodo. Questa strategia partiziona lo spazio delle soluzioni, costruisce esattamente due figli per nodo e rende inammissibile la soluzione ottima del rilassamento. La strategia è anche "teoricamente" bilanciata, ma in pratica ciò può non risultare del tutto vero. Infatti, in molte istanze del problema dello zaino la soluzione ottima contiene "pochi" oggetti, per cui fissare un oggetto come facente parte della soluzione può avere un effetto maggiore che escluderlo dalla soluzione (può esservi un numero minore di soluzioni ammissibili nel primo caso che nel secondo). La visita depth-first permette di implementare l'algoritmo in modo molto efficiente. Ogni volta che si scende (o si sale) di un livello nell'albero delle decisioni la capacità residua $\bar{b}$ può diminuire (aumentare), oppure rimanere costante. La soluzione ottima del rilassamento può essere determinata molto efficientemente partendo dalla soluzione ottima del rilassamento nel nodo padre. Per via della potenziale asimmetria tra il fissare una variabile a 0 e ad 1, può essere consigliabile generare (e quindi visitare) per primo il nodo in cui la variabile viene fissata ad 1.

Esercizio 7.11. Si discuta nei dettagli come implementare questo algoritmo di enumerazione implicita in modo che risulti il più efficiente possibile, o meglio ancora lo si implementi e si verifichi nella pratica l'impatto delle diverse scelte implementative.

Esempio 7.4. B&B per lo zaino

Si consideri la seguente istanza del problema dello zaino:

$$\begin{array}{rrcrcrcrcl} \max & 11x_1 & + & 8x_2 & + & 7x_3 & + & 6x_4 & & \\ & 5x_1 & + & 4x_2 & + & 4x_3 & + & 4x_4 & \leq & 12 \\ & x_1 & , & x_2 & , & x_3 & , & x_4 & \in & \{0,1\} \end{array} .$$

Ad ogni nodo dell'albero delle decisioni denotiamo con $\bar{x}$ la soluzione ottima del rilassamento continuo (si noti che gli oggetti sono già ordinati per costo unitario non crescente) e $\bar{z}$ la corrispondente valutazione superiore, con $\underline{x}$ la soluzione determinata dall'euristica CUD e $\underline{z}$ la corrispondente valutazione inferiore, e con c^* il vettore dei costi ridotti; si noti che

$$c_i^* = c_i - y^* a_i = c_i - \frac{c_h}{a_h} a_i = \left(\frac{c_i}{a_i} - \frac{c_h}{a_h} \right) a_i$$

(si vedano i §5.1.2.1 e 6.1.2.2). Al nodo radice dell'albero delle decisioni abbiamo quindi

$$\bar{x} = [\,1\,,\,1\,,\,0.75\,,\,0\,] \quad , \quad \bar{z} = 24.25 \quad , \quad \underline{x} = [\,1\,,\,1\,,\,0\,,\,0\,] \quad , \quad \underline{z} = z = 19 \ .$$

Il gap assoluto al nodo radice è quindi di 5.25. Essendo $c^* = [\,2.25\,,\,1\,,\,0\,,\,-1\,]$, nessun costo ridotto è in valore assoluto maggiore del gap è non è quindi possibile fissare variabili. Occorre quindi effettuare l'operazione di branching sulla variabile frazionaria x_3.

Il successivo nodo estratto da Q corrisponde quindi al problema in cui $x_3 = 1$, ossia

$$\begin{array}{rrcrcrcrcl} 7 + \max & 11x_1 & + & 8x_2 & + & 6x_4 & & & & \\ & 5x_1 & + & 4x_2 & + & 4x_4 & \leq & 8 & \\ & x_1 & , & x_2 & , & x_4 & \in & \{0,1\} & \end{array} .$$

Il rilassamento continuo e l'euristica forniscono rispettivamente

$$\bar{x} = [\,1\,,\,0.75\,,\,1\,,\,0\,] \quad , \quad \bar{z} = 24 \quad , \quad \underline{x} = [\,1\,,\,0\,,\,1\,,\,0\,] \quad , \quad \underline{z} = 18 < z = 19$$

per cui il gap al nodo scende a 5 (la soluzione candidata non viene modificata); dato che $c^* = [\,1\,,\,0\,,\,*\,,\,-2\,]$ (la variabile x_3 è fissata e quindi non compare nel rilassamento continuo, pertanto non ha costo ridotto), nessuna variabile viene fissata ed è necessario effettuare l'operazione di branching sulla variabile frazionaria x_2.

Il successivo nodo estratto da Q corrisponde quindi al problema in cui $x_2 = x_3 = 1$, $\bar{b} = 4$. Poichè $a_1 = 5 > \bar{b}$ viene quindi fissata anche $x_1 = 0$. Il rilassamento continuo e l'euristica forniscono quindi

$$\bar{x} = \underline{x} = [\,0\,,\,1\,,\,1\,,\,1\,] \quad , \quad \bar{z} = \underline{z} = z = 21$$

ed il nodo viene potato per ottimalità, mentre viene aggiornata la soluzione candidata. Si noti che non effettuare il preprocessing, ossia non fissare $x_1 = 0$, avrebbe comportato ottenere $\bar{x} = [\,0.80\,,\,1\,,\,1\,,\,0\,]$ e $\bar{z} = 23.8$ e quindi non poter potare il nodo.

Si esegue quindi un "backtrack" e si passa ad esaminare il problema in cui $x_2 = 0$, $x_3 = 1$, per cui il rilassamento continuo e l'euristica forniscono

$$\bar{x} = [\,1\,,\,0\,,\,1\,,\,0.75\,] \quad , \quad \bar{z} = 22.5 \quad , \quad \underline{x} = [\,1\,,\,0\,,\,1\,,\,0\,] \quad , \quad \underline{z} = 18 < z = 21$$

corrispondenti ad un gap di 1.5. Si noti che in questo momento si ha una valutazione superiore sull'ottimo del problema in cui $x_3 = 1$ pari a $\max\{\,21\,,\,22.5\,\} = 22.5$, ossia migliore della valutazione superiore di 24 ottenuta dal rilassamento continuo. Poichè si ha $c^* = [\,2.75\,,\,*\,,\,*\,,\,0\,]$, è possibile fissare ad 1 la variabile x_1: infatti, $22.5 - 2.75 = 19.75 < 21$. Si noti che 19.75 è una valutazione superiore del valore del rilassamento continuo in cui $x_1 = 0$; tale valutazione è molto approssimata (ma corretta) in quanto la soluzione ottima di tale rilassamento è $[\,0\,,\,0\,,\,1\,,\,1\,]$ che ha costo 13. Fissare $x_1 = 1$ lascia $\bar{b} = 3$, e quindi porta a fissare anche $x_4 = 0$. Siamo così giunti ad una foglia dell'albero delle decisioni, ossia a dimostrare che nessuna soluzione nel sottoalbero corrispondente a $x_2 = 0$, $x_3 = 1$ ha costo migliore di 18. È così terminata l'esplorazione dell'intero sottoalbero corrispondente ad $x_3 = 1$, per il quale si ha adesso $\bar{z} = \underline{z} = 21$.

Si esegue quindi un "backtrack" e si passa ad esaminare il problema in cui $x_3 = 0$, per cui il rilassamento continuo e l'euristica forniscono

$$\bar{x} = [\,1\,,\,1\,,\,0\,,\,0.75\,] \quad , \quad \bar{z} = 23.5 \quad , \quad \underline{x} = [\,1\,,\,1\,,\,0\,,\,0\,] \quad , \quad \underline{z} = 19 < z = 21 \ ;$$

abbiamo quindi $z(P) \leq \max\{\,21\,,\,23.5\,\} = 23.5$, e quindi possiamo affermare che la soluzione candidata ha un gap massimo di 2.5 rispetto alla soluzione ottima. Poichè si ha $c^* = [\,2.75\,,\,2\,,\,*\,,\,0\,]$, è possibile fissare $x_1 = 1$: infatti, $23.5 - 2.75 = 20.75 < 21$. Ancora una volta si noti che 20.75 è una valutazione superiore (approssimata) del valore del rilassamento continuo in cui $x_1 = 0$, che ha soluzione ottima $[\,0\,,\,1\,,\,0\,,\,1\,]$ con costo 14. Comunque questo non permette di terminare l'esplorazione, per cui si deve eseguire il branching sulla variabile frazionaria x_4.

Il nodo successivo corrisponde al problema in cui $x_1 = x_4 = 1$, $x_3 = 0$. Poichè $\bar{b} = 3 > a_2 = 4$ si può fissare $x_2 = 0$ e si è raggiunta la foglia dell'albero delle decisioni corrispondente alla soluzione $[\,1\,,\,0\,,\,0\,,\,1\,]$ di costo $17 < z = 21$. Poiché non ci sono più decisioni da prendere si esegue un "backtrack", ossia si passa ad esaminare il nodo successivo.

Il nodo corrisponde al problema in cui $x_1 = 1$, $x_3 = x_4 = 0$. In questo caso si ha

$$\bar{x} = \underline{x} = [\,1\,,\,1\,,\,0\,,\,0\,] \quad , \quad \bar{z} = \underline{z} = z = 19$$

ed il nodo viene potato per ottimalità. A questo punto abbiamo $Q = \emptyset$, e $z(P) \leq \max\{\,21\,,\,19\,\} = 21$; infatti, la soluzione candidata $x = [\,0\,,\,1\,,\,1\,,\,1\,]$ è ottima.

Questo algoritmo di enumerazione implicita è un caso particolare del *Branch and Bound basato sulla PL*, che è uno tra gli algoritmi più generali per risolvere problemi di *OC*. L'algoritmo risolve in tempo finito qualsiasi problema di Programmazione Lineare Intera (o mista)

$$(P) \quad \max\{\,cx \,:\, Ax \leq b\,,\; x \in \mathbb{Z}^n\,\}$$

utilizzando un metodo di enumerazione implicita che usa il rilassamento continuo per produrre valutazioni superiori. Il branching viene effettuato sulle variabili frazionarie: scelta una variabile x_i frazionaria nella soluzione ottima x^* del rilassamento continuo, si costruiscono i due figli del nodo nell'albero delle decisioni semplicemente aggiungendo alla formulazione del problema i vincoli

$$x_i \leq \lfloor\, x_i^* \,\rfloor \quad \text{oppure} \quad x_i \geq \lceil\, x_i^* \,\rceil\,. \tag{7.1}$$

Questi vincoli equipartizionano (in teoria) l'insieme ammissibile e sono soddisfatti da tutte le soluzioni intere ma non da x^*. Si noti che questa regola di branching generalizza il fissare a 0 oppure 1 una variabile binaria che abbia valore frazionario nel rilassamento continuo. In presenza di strutture specifiche nel modello si possono comunque utilizzare anche regole di branching diverse, come per il caso degli Special Ordered Sets descritti nel §7.2.4. Sfruttando le opportunamente le capacità di riottimizzazione degli algoritmi del simplesso (si veda il §7.2.1) e strategie di visita opportune si può implementare questo algoritmo in modo che sia in grado di valutare in modo piuttosto efficiente ciascun nodo nell'albero delle decisioni.

I problemi principali di questo algoritmo sono l'efficacia e l'efficienza di euristiche e rilassamenti. Per le prime, le tecniche di arrotondamento semplici spesso hanno scarsa efficacia; le euristiche più sofisticate (si veda il §6.1.2.1) hanno dimostrato buone prestazioni in molti casi, contribuendo a migliorare in modo significativo le prestazioni dei solutori "general-purpose" basati su queste tecniche, ma possono essere relativamente costose. Per il secondo problema sono molto utili le tecniche poliedrali discusse al §7.4. Poiché sia euristica che rilassamento richiedono la soluzione di problemi di *PL* (anche molti in corrispondenza di un singolo nodo), un punto critico di questo approccio riguarda il fatto che gli algoritmi per la *PL* sono comunque relativamente costosi, sia in tempo che in memoria, quando la dimensione del modello cresce, ad esempio rispetto ad altri tipi di problema come quelli di flusso su rete.

Esistono implementazioni estremamente sofisticate ed efficienti del B&B basato sulla *PL* che, sfruttando lo stato dell'arte nel campo della *PL* e delle tecniche poliedrali generali, riescono a risolvere problemi di *PLI* di dimensioni rilevanti senza necessità di interventi da parte dell'utente. Resta però vero che in molti casi i problemi di *OC* derivanti da modelli realistici di problemi decisionali concreti non vengono risolti con sufficiente efficienza anche dai migliori di questi approcci, almeno non senza un rilevante intervento da parte di esperti in grado di sfruttare al meglio le proprietà del problema da risolvere.

7.3.2 Il problema del commesso viaggiatore

Discutiamo adesso un algoritmo di enumerazione implicita per il problema del commesso viaggiatore che utilizzi (MS1-T) come rilassamento e "twice around MST" come euristica. Questa versione base potrebbe poi essere modificata utilizzando

- tecniche di ricerca locale per migliorare la soluzione determinata dall'euristica;
- il rilassamento Lagrangiano avente (MS1-T) come sottoproblema, possibilmente trasformando quindi l'euristica in un'euristica Lagrangiana.

Per la regola di branching supponiamo quindi di avere a disposizione un 1-albero ottimo T^*: dobbiamo prendere decisioni sul problema in modo da rendere T^* non ammissibile. Se ogni nodo in T^* ha esattamente due lati incidenti allora T^* è un ciclo Hamiltoniano, e quindi è la soluzione ottima del problema; possiamo quindi supporre che esista un nodo i in cui incidano più di due lati di T^*. Sia $L = \{ l_1 , \ldots , l_k \}$ $(k > 2)$ l'insieme di questi lati. Un semplice modo per effettuare il branching corrisponde a selezionare un qualsiasi $l_i \in L$ ed a costruire due figli, uno nel quale si è deciso che l_i faccia parte del ciclo Hamiltoniano e l'altro in cui si è deciso che non ne faccia parte. Questa regola è semplice da implementare, potenzialmente equipartiziona l'insieme (ma si noti che ogni nodo ha solo due lati incidenti nel ciclo e molti lati incidenti che non appartengono al ciclo, quindi in uno dei due figli viene presa una decisione "più forte") e crea esattamente due figli per nodo. Per contro, nel figlio in cui si decide che il lato fa parte del ciclo, la soluzione ottima del rilassamento non cambia.

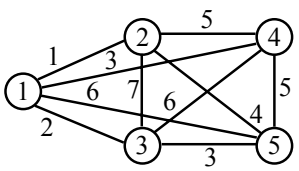
Una regola che assicura l'inammissibilità di T^* in tutti i figli è la seguente. Vengono costruiti $k(k-1)\,/\,2+k+1$ figli del nodo. Nei primi $k(k-1)\,/\,2$ vengono fissati come appartenenti al ciclo Hamiltoniano tutte le possibili coppie di lati nell'insieme L; di conseguenza, una volta fissata una qualsiasi coppia tutti gli altri lati incidenti in i vengono fissati come non appartenenti al ciclo. Nei successivi k figli si fissa uno dei lati $l_i \in L$ come appartenente al ciclo e tutti gli altri come non appartenenti. Nell'ultimo figlio si fissano tutti i lati in L come non appartenenti al ciclo. Questa regola partiziona l'insieme, ma ha diversi svantaggi: è piuttosto complessa da implementare, fissa lati come appartenenti al ciclo e quindi costringe ed operazioni complesse per adattare euristica e rilassamento (si veda l'Esempio 7.6) e può produrre molti figli.

Una regola di branching ancora diversa prevede di costruire k figli del nodo: in ciascuno dei figli si fissa uno dei lati $l_i \in L$ come *non* appartenente al ciclo, e non si prende alcuna altra decisione. Questa regola assicura l'inammissibilità di T^*, è relativamente semplice da implementare, fissa solamente lati come non appartenenti al ciclo (il che semplifica molto l'adattamento di rilassamento ed euristica) e produce un numero minore di figli rispetto alla precedente. Per contro non partiziona l'insieme: qualsiasi ciclo che non contenga nessuno degli archi in L appartiene a tutti i figli. Comunque quest'ultima regola di branching è probabilmente la più ragionevole di quelle proposte; si noti che, per limitare ulteriormente il numero di figli da generare, è possibile scegliere, tra tutti i nodi in cui incidano più di due lati di T^*, quello in cui ne incide il minor numero. Dato che comunque il numero di figli può essere alto, può essere ragionevole utilizzare una strategia di visita di tipo depth-first. Per questo algoritmo non proponiamo nessuna specifica tecnica di preprocessing.

Esercizio 7.12. Si propongano e discutano altre regole di branching e tecniche di preprocessing per questo algoritmo. Si implementi l'algoritmo e si verifichi nella pratica l'impatto delle diverse scelte implementative.

Esempio 7.5. B&B per il (TSP)

Si consideri l'istanza del problema (TSP) illustrata in figura qui accanto. Al nodo radice si ottiene l'1-albero mostrato in Figura 7.3(a) (l'arco tratteggiato è quello aggiunto al (MST)), di costo $\underline{z} = 13$, e quindi l'euristica ottiene il ciclo mostrato a fianco, di costo $\bar{z} = 21$. Si ha quindi un gap di 8, e bisogna effettuare l'operazione di branching. Per questo si seleziona il nodo 1, che ha tre archi incidenti nell'1-albero, e si costruiscono tre figli, nei quali sono escluso dal ciclo rispettivamente il lato $\{ 1 , 2 \}$, il lato $\{ 1 , 4 \}$ ed il lato $\{ 1 , 3 \}$.



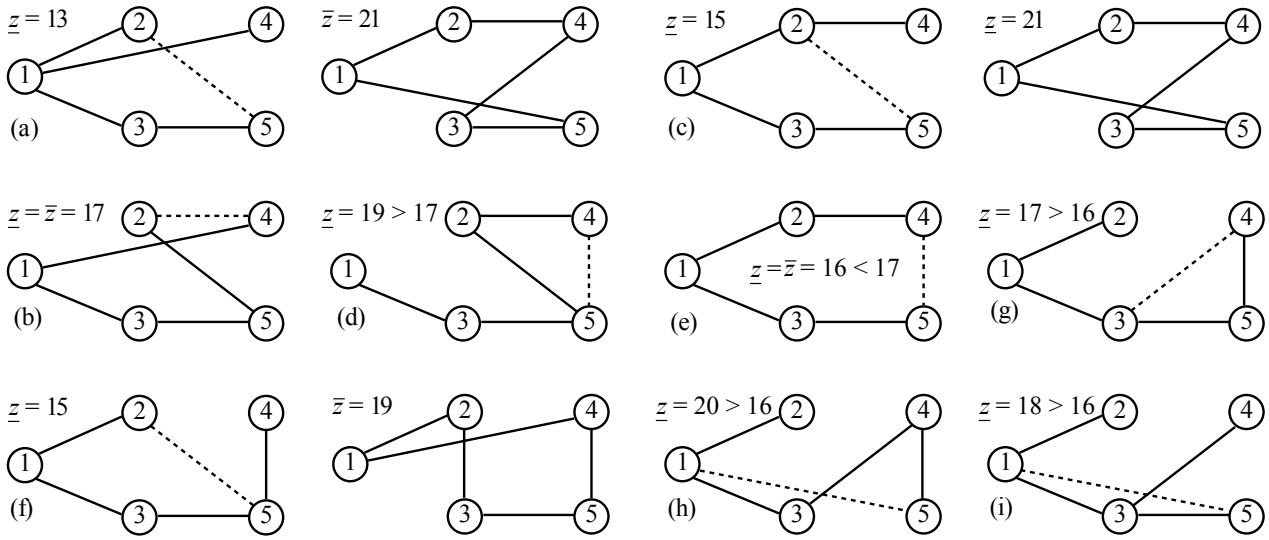


Figura 7.3: Algoritmo *B&B* per il (TSP)

$\boxed{x_{12} = 0}$: l'1-albero corrispondente, mostrato in Figura 7.3(b), è un ciclo Hamiltoniano di costo 17. Viene quindi aggiornata la soluzione candidata, ed il nodo è potato per ottimalità.

$\boxed{x_{14} = 0}$: si ottengono l'1-albero ed il ciclo Hamiltoniano mostrati in Figura 7.3(c); poiché il problema non è stato risolto e $\underline{z} = 15 < z = 17$, si effettua l'operazione di branching. Per questo si seleziona il nodo 2, che ha tre archi incidenti nell'1-albero, e si costruiscono i tre figli nei quali sono esclusi dal ciclo rispettivamente i lati $\{1, 2\}$, $\{2, 5\}$ e $\{2, 4\}$.

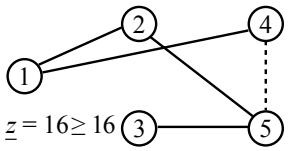
$\boxed{x_{14} = x_{12} = 0}$: si ottiene l'1-albero mostrato in Figura 7.3(d), di costo $19 > z = 17$, e quindi il nodo viene potato dalla valutazione superiore.

$\boxed{x_{14} = x_{25} = 0}$: si ottiene l'1-albero mostrato in Figura 7.3(e), che è un ciclo Hamiltoniano di costo 16; viene quindi aggiornata la soluzione candidata, ed il nodo è potato per ottimalità.

$\boxed{x_{14} = x_{24} = 0}$: si ottengono l'1-albero ed il ciclo Hamiltoniano mostrati in Figura 7.3(f); poiché il problema non è stato risolto e $\underline{z} = 15 < z = 16$, si effettua l'operazione di branching. Per questo si seleziona il nodo 5, che ha tre archi incidenti nell'1-albero, e si costruiscono i tre figli nei quali sono esclusi dal ciclo rispettivamente i lati $\{2, 5\}$, $\{3, 5\}$ e $\{4, 5\}$.

Per tutti e tre i nodi appena generati si ottengono 1-alberi, mostrati rispettivamente in Figura 7.3(g), 7.3(h) e 7.3(i), che hanno costo maggiore di $z = 16$; tutti e tre i nodi vengono quindi potati dalla valutazione superiore, e si risale nell'albero delle decisioni (si esegue un "backtrack") di due livelli.

$\boxed{x_{13} = 0}$: si ottiene l'1-albero mostrato in Figura 7.3(e); poiché il costo dell'1-albero è pari a $z = 16$, non possono esistere in questo sottoalbero dell'albero delle decisioni soluzioni di costo inferiore a 16 (si noti che in principio potrebbero essercene di costo uguale) ed il nodo viene potato dalla valutazione superiore. Siccome $Q = \emptyset$ l'algoritmo termina, dimostrando che la soluzione candidata, mostrata qui accanto, è ottima per il problema.



7.3.3 Il problema del cammino minimo vincolato

Per questo problema, introdotto nell'Esempio 6.4, analizzeremo un algoritmo enumerativo che usa come rilassamento un problema di cammino minimo, ossia il rilassamento per eliminazione del vincolo sulla lunghezza dei cammini. Analoghe considerazioni potrebbero poi essere fatte per il corrispondente rilassamento Lagrangiano. In questo caso decidiamo di non utilizzare nessuna euristica: le soluzioni ammissibili verranno fornite dal rilassamento stesso, o al limite dalla visita di foglie dell'albero delle decisioni. Ciò suggerisce l'utilizzo della regola di visita depth-first.

Dobbiamo quindi decidere un'opportuna regola di branching. Il rilassamento produce un cammino P tra l'origine e la destinazione: se il cammino rispetta il vincolo di lunghezza allora è ottimo e non occorre effettuare l'operazione di branching. Possiamo quindi supporre di avere, al momento di dover effettuare il branching, un cammino P di lunghezza maggiore della soglia L. Abbiamo già discusso come la regola di branching "standard", ossia selezionare un arco di P e fissarlo come appartenente o no al cammino ottimo, non sia applicabile, in quanto la sua applicazione ripetuta porterebbe a dover

risolvere nei nodi interni dell'albero delle decisioni problemi di cammino minimo vincolato (in cui il vincolo è di passare per un insieme di archi dato) sostanzialmente difficili come il problema che si vuole risolvere. Inoltre questa regola, al momento in cui fissa l'arco come appartenente al cammino, non rende inammissibile la soluzione ottima del rilassamento. Una regola di branching alternativa può essere ottenuta come nell'esempio precedente: si costruiscono tanti figli quanti sono gli archi di P, ed in ciascun figlio si fissa come non appartenente al cammino ottimo uno degli archi. Questa regola rende inammissibile la soluzione ottima del rilassamento, ma ha gli inconvenienti già sottolineati nel caso dell'esempio precedente: può produrre un numero di figli elevato, e soprattutto non partiziona l'insieme ammissibile. Un vantaggio di questa regola è invece che, decidendo solo per l'esclusione di archi, rende molto facile adattare il rilassamento a lavorare nei nodi interni: è sufficiente eliminare gli archi dal grafo su cui si calcola il cammino minimo.

Per questo problema possiamo però proporre una regola alternativa che produce un numero di figli inferiore ed ha il vantaggio di partizionare l'insieme delle soluzioni, anche se al costo di complicare l'implementazione del rilassamento. La regola funziona come segue: dato il cammino P tra r e t, che viola il vincolo sulla massima lunghezza, si costruisce un primo figlio in cui il primo arco del cammino è fissato come non appartenente al cammino ottimo. Nel secondo figlio si fissa invece il primo arco del cammino come appartenente al cammino ottimo, ed il secondo arco del cammino come non appartenente al cammino ottimo. Nel terzo figlio si fissano i primi due archi del cammino come appartenenti al cammino ottimo ed il terzo come non appartenente, e così via finché non si giunge al primo arco del cammino tale che il sottocammino che parte da r e contiene tutti i primi archi di P, fino a quello, viola il vincolo di lunghezza. L'ultimo figlio è quindi quello in cui l'arco "critico" è fissato come non appartenente al cammino ottimo e tutti i precedenti sono fissati come appartenenti: non è necessario generare altri figli, perchè nessun cammino ammissibile può contenere altri sottocammini di P che comprendono r. Questa regola di branching è un caso particolare di una regola che si può applicare per problemi con variabili $x_i \in \{0, 1\}$, in cui nella soluzione ottima x^* del rilassamento si abbia che l'insieme $S = \{ i : x_i^* = 1 \} = \{ i_1, i_2, \ldots, i_k \}$ contiene "troppe" variabili. La regola di branching crea un figlio con $x_{i_1} = 0$, un figlio con $x_{i_1} = 1$ e $x_{i_2} = 0$, un figlio con $x_{i_1} = x_{i_2} = 1$ e $x_{i_3} = 0$ e così via. Nel caso di (CSP) non è necessario generare tutti i figli in quanto alcuni di essi sono certamente non ammissibili. Si noti che questa regola non è bilanciata.

Questa regola di branching fissa archi come appartenenti al cammino, e quindi rischierebbe di incorrere negli stessi problemi di incompatibilità col rilassamento della regola standard. Ma in questo caso non sono fissati archi qualsiasi: vengono sempre fissati interi sottocammini che partono da r. È quindi possibile adattare il rilassamento semplicemente rimuovendo tutti i nodi del cammino fissato dal grafo, tranne il nodo terminale, e rendendo il nodo terminale del cammino la radice del cammino minimo. Contemporaneamente occorre aggiornare la soglia di massima lunghezza L sottraendo ad essa la lunghezza del sottocammino fissato: a questo punto si può anche applicare un'operazione di preprocessing eliminando dal grafo tutti gli archi che abbiano $l_{ij} > L'$, dove L' è la soglia aggiornata.

Esempio 7.6. B&B per (CSP)

Applichiamo l'algoritmo *B&B* all'istanza dell'Esempio 6.7. Al nodo radice si ha la situazione rappresentata in Figura 7.4(a): il cammino P (archi evidenziati) ha costo 3 ma lunghezza $3 > L = 2$. Poiché non si è determinata nessuna soluzione ammissibile si ha $z = +\infty$, e quindi un gap infinito: si applica quindi l'operazione di branching.

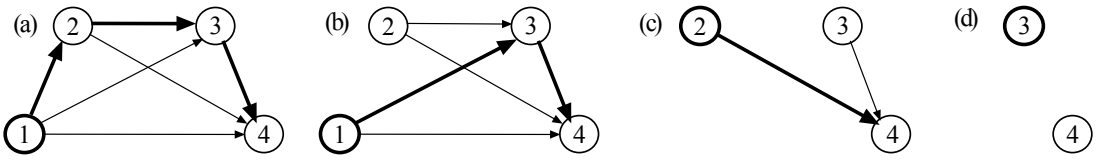


Figura 7.4: Algoritmo *B&B* per il (CSP)

$\boxed{x_{12} = 0}$: il corrispondente cammino minimo è mostrato in Figura 7.4(b); il cammino ha costo 4 ed è ammissibile, per cui si pone $z = 4$ ed il nodo viene potato per ottimalità.

$\boxed{x_{12} = 1,\ x_{23} = 0}$: si sposta la radice r al nodo 2 ponendo $L = 1$, ottenendo il cammino ottimo illustrato in Figura 7.4(c) che ha costo $5 > z = 4$: il nodo viene quindi potato dalla valutazione inferiore.

$\boxed{x_{12} = x_{23} = 1,\ x_{34} = 0}$: si sposta la radice r al nodo 3 e si pone $L = 0$, come illustrato in Figura 7.4(d); sul grafo non esistono cammini tra 3 e 4, quindi il nodo viene potato per inammissibilità. In effetti, poiché il cammino fissato è lungo esattamente L ed il suo nodo terminale non è la destinazione in questo caso si potrebbe anche evitare di creare il nodo. Essendo Q vuoto l'algoritmo termina riportando come soluzione ottima il cammino di costo 4 mostrato in Figura 7.4(b).

Esercizio 7.13. Si discuta se e come sia possibile adattare gli algoritmi per il problema dei cammini minimi studiati nel §2.2 affinché sfruttino la conoscenza del cammino e delle etichette ottime corrispondenti al padre per risolvere in modo più efficiente i rilassamenti nei nodi figli.

Esercizio 7.14. Si propongano algoritmi di enumerazione implicita per tutti i problemi di *OC* visti in queste dispense, proponendo per ciascuno uno o più rilassamenti, euristiche, regole di branching, operazioni di preprocessing e così via, e discutendo i pró ed i contro delle diverse alternative.

7.4 Tecniche poliedrali

Nei solutori *general-purpose* per la *PLI* basati sulla *PL* (si veda, ad esempio, la discussione in fondo al §7.3.1) uno dei principali limiti all'efficacia è costituito dalla "debolezza" delle valutazioni superiori (nel caso di un problema di massimo) ottenute dal rilassamento continuo (cf. §4.2.1), come illustrato nel §6.1.1. Ciò dipende dalla differenza tra il poliedro descritto dalla formulazione *PLI* utilizzata e l'inviluppo convesso delle soluzioni intere, come discusso nel §4.2.2. Visto che, in generale, l'espressione dell'inviluppo convesso non è disponibile, nemmeno sotto forma di un separatore per i vincoli corrispondenti (che, del resto, sappiamo avere la stessa complessità del problema di partenza) sono state sviluppate diverse classi di *disuguaglianze valide*, o *tagli*, che cercano di approssimare quanto meglio possibile l'inviluppo convesso entro i limiti di tempo richiesti dall'algoritmo di enumerazione implicita. Infatti questo deve far girare (in linea di principio) i separatori ad ogni nodo dell'albero delle decisioni, e potenzialmente (molto) più di una volta per ogni nodo, in quanto se il separatore determina dei tagli violati dalla soluzione x^* del rilassamento continuo esso deve essere risolto nuovamente. Per questo è comunemente utilizzata la strategia di non fermare il processo di separazione non appena si individui il primo taglio violato, ma di inserirne più di uno (tipicamente qualche centinaio) insieme, in modo da diminuire sia il numero di problemi di *PL* risolti che il numero di volte che vengono invocati i separatori.

L'aspetto critico di questo processo è chiaramente la scelta dei separatori da utilizzare. In generale si possono individuare tre diverse tipologie di tagli, in ordine decrescente di specificità per il problema da risolvere:

- *Tagli specifici per il problema da risolvere*, come quelli dell'Esempio 4.5 per n-co-cubo o quelli dell'Esempio 4.6 per (MST). Si noti che in entrambi questi casi i tagli sono in effetti una parte integrante della formulazione, nel senso che essi devono essere tutti rispettati affinché la soluzione intera sia ammissibile. In questo caso si parla di *lazy constraints* (vincoli "pigri"), per distinguerli dagli *user cuts* che invece servono unicamente a rendere inammissibili soluzioni frazionarie, come quelli dell'Esempio 4.7 per (KP). La differenza tra le due classi è che i separatori per i lazy constraints vanno invocati (anche) sulle soluzioni intere, mentre quelli per gli user cuts solamente su quelle frazionarie. In entrambi i casi, però, i separatori non possono essere parte integrante del solutore, in quanto i tagli sono validi solamente per quel(la) particolare (formulazione *PLI* del) problema, mentre il solutore si intende essere completamente generale. Per questo i solutori forniscono dei meccanismi, tipicamente indicati come `callback`, mediante i quali invocano in momenti prestabiliti funzioni definite dagli utenti, passando loro le informazioni necessarie (la soluzione ottima x^* del rilassamento continuo nel caso degli user cuts o una soluzione intera nel caso dei lazy constraints). La correttezza e l'efficienza dei separatori sono quindi completamente a carico dell'utente del solutore. L'alta specifictà dei tagli spesso implica che questi hanno un grande effetto sull'efficacia del rilassamento continuo, ma la loro implementazione richiede un'approfondita conoscenza della struttura matematica della formulazione *PLI* e tipicamente un non banale lavoro implementativo.

- *Tagli specifici per sotto-strutture molto comuni.* I primi due esempi fatti nel punto precedente potrebbero sembrare poco sensati, in quanto entrambi i separatori sono per problemi polinomiali. In realtà almeno quello per (MST) è di possibile utilità pratica ad esempio nella risoluzione di problemi di (TSP), dato che, come sappiamo (cf. Esempio 1.7), le possibili formulazioni di (TSP) contengono "naturalmente" i vincoli di connessione tipici di (MST) che il separatore è in grado di trattare efficientemente. Pertanto, una volta che sia disponibile un separatore per (i vincoli corrispondenti a) una certa struttura combinatoria esso è utilizzabile in tutti i problemi che contengono tale struttura. Questa idea è stata utilizzata con successo per identificare un certo numero di strutture combinatorie molto comuni che sono facilmente desumibili dalla formulazione algebrica del problema. Un caso paradigmatico è quello delle cover inequalities per lo zaino (cf. Esempio 4.7): nei problemi di Programmazione Lineare 0/1, ogni singolo vincolo può essere considerato uno zaino; in altri termini le soluzioni ammissibili sono quelle che appartengono all'intersezione di tutti gli zaini. Pertanto, ciascuna disuguaglianza valida per ciascuno degli zaini è valida per tutte le soluzioni intere del problema complessivo. Il separatore per tali tagli è molto generale e può essere implementato in modo efficiente (ad esempio, utilizzando le euristiche dei §5.1.1.1 o §5.2.1.1 o gli approcci che vedremo al §7.5) nel solutore *general-purpose*. Questo approccio è stato ripetuto con successo per un certo numero (circa una dozzina) di separatori che corrispondono a strutture algebriche molto comuni nelle formulazioni *PLI*; pur senza poterle descrivere in nessun dettaglio, alcune di queste sono note come *clique inequalities*, *disjunìctive cuts*, *flow path/cover cuts*, *mixed integer rounding cuts*, *zero-half cuts*, ed altre. Le sofisticate implementazioni di tali separatori, e del processo complessivo che li usa, disponibili nei moderni solutori general-purpose forniscono spesso il principale contributo a rendere risolubili in tempi ragionevoli molte istanze di problemi corrispondenti ad applicazioni reali.
- Il principale limite dei separatori descritti al punto precedente riguarda il fatto che essi richiedono strutture specifiche all'interno della matrice dei coefficienti del problema, ancorché possibilmente semplici: ad esempio, per le cover inequalities—almeno come descritte nell'Esempio 4.7, ma esistono generalizzazioni—è necessario che le variabili siane tutte binarie. Se la struttura non è presente, il separatore non può essere utilizzato. Esistono però classi di disuguaglianze valide del tutto generali che si applicano a qualsiasi problema di *PLI*, qualsiasi sia la sua matrice dei coefficienti. Separatori per queste classi di tagli sono quindi particolarmente adatti ad essere implementati all'interno di solutori di *PLI* general-purpose.

Nel seguito di questo paragrafo discuteremo prima una classe molto generale di disuguaglianze valide del terzo tipo, i *tagli di Chvátal*, che permettono in principio di derivare tutti i vincoli che caratterizzano l'inviluppo convesso dei punti interi. Tale classe è però definita in modo non algoritmico, ossia senza specificare come i tagli debbano essere separati. Per questo presenteremo a seguire i *tagli di Gomory*, un caso particolare dei precedenti che ha un separatore algoritmico molto ben adattato all'uso nel contesto del Branch&Bound basato sulla *PL*. Termineremo quindi con alcune considerazioni generali relative all'utilizzo di tecniche poliedrali all'interno di metodi di enumerazione implicita.

7.4.1 I tagli di Chvátal

Per il generico problema

$$(PLI) \quad \max\{\, cx \,:\, Ax \leq b \,,\; x \in \mathbb{Z}^n \,\}$$

ed il poliedro $\mathcal{P} = \{\, x \in \mathbb{R}^n \,:\, Ax \leq b \,\}$ ad esso associato, si può dimostrare un risultato abbastanza sorprendente secondo cui "tutti i tagli necessari" possono essere derivati da una semplice operazione. Per semplicità si assume che tutte le variabili abbiano vincoli di integralità—ma il risultato può essere generalizzato a problemi misti-interi—e che sia A che b abbiano componenti intere ($A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}^m$); poiché nella stragrande maggioranza dei casi i numeri reali sui computer sono approssimati mediante numeri in virgola mobile (`float`, `double`, ...) che sono di fatto numeri razionali, questa assunzione può essere fatta senza perdita di generalità.

Si consideri un qualsiasi vettore $y \in \mathbb{R}^m_+$: è immediato verificare che la singola disuguaglianza lineare

$$a_y x \leq b_y \qquad \text{con} \qquad a_y = yA \text{ e } b_y = yb \tag{7.2}$$

è valida per tutti i punti di $\mathcal{P}$: infatti,

$$Ax \leq b \text{ e } y \geq 0 \quad \Longrightarrow \quad (yA)x \leq (yb) \ .$$

Questo in effetti può essere considerato il "principio base" della teoria della dualità: se $a_y = yA = c$ allora $cx \leq b_y = yb$, ossia qualsiasi soluzione duale ammissibile fornisce una valutazione superiore sul valore di qualsiasi soluzione primale ammissibile (cf. §3.2.2).

In questo caso utilizziamo però il principio in modo diverso: selezionando y in modo tale che $a_y \in \mathbb{Z}^n$ si ottiene che

$$a_y x \leq \lfloor b_y \rfloor \tag{7.3}$$

è una disuguaglianza valida per tutti i punti di $conv(\,\mathcal{P} \cap \mathbb{Z}^n\,)$, dove $\lfloor \cdot \rfloor$ indica la parte intera inferiore. Infatti, poiché $x \in \mathbb{Z}^n$ e $a_y \in \mathbb{Z}^n$ allora $a_y x \in \mathbb{Z}$, e quindi se è inferiore a b_y allora è inferiore anche alla sua parte intera inferiore. Si noti che il *taglio di Chvátal* (7.3) è "più forte" della disuguaglianza originaria (7.2) se $b_y = yb \notin \mathbb{Z}$: siccome il lato destro è più piccolo ci sono punti che soddisfano la seconda ma non la prima. Definiamo ora l'insieme $\mathcal{P}^{(1)} \subseteq \mathcal{P}$ dei punti che rispettano *tutti* i tagli di Chvátal, detto *chiusura di Chvátal di rango 1*: è facile vedere che $\mathcal{P}^{(1)} \subseteq \mathcal{P}$, infatti qualsiasi disuguaglianza originaria $A_i x \leq b_i$ è un taglio di Chvátal (basta prendere $y = u_i$, con u_i l'i-esimo vettore della base canonica di $\mathbb{R}^m$). Potrebbe apparire che la differenza tra (7.2) e (7.3) sia trascurabile, e che quindi $\mathcal{P}^{(1)}$ sia non troppo diverso da $\mathcal{P}$, ma questo in effetti non è il caso. È infatti possibile dimostrare che:

- non è necessario considerare tutti i vettori $y \geq 0$ ma solamente quelli $0 \leq y \leq u$ (u essendo il vettore con tutte componenti unitarie);
- $\mathcal{P}^{(1)}$ è un poliedro, ossia esiste un numero finito di tagli di Chvátal che lo caratterizzano completamente;
- definendo ricorsivamente la *chiusura di Chvátal di rango* $k > 1$, $\mathcal{P}^{(k)}$, come il poliedro ottenuto ripetendo l'operazione a partire dalla *chiusura di Chvátal di rango* $k-1$, è possibile dimostrare che esiste un numero finito t per cui $\mathcal{P}^{(t)} = conv(\,\mathcal{P} \cap \mathbb{Z}^n\,)$.

In altri termini, qualsiasi disuguaglianza necessaria a rappresentare l'inviluppo convesso delle soluzioni intere del problema può essere ottenuta ripetendo un numero finito di volte l'operazione (7.3). Si noti però che non è sufficiente utilizzare le diseguaglianze originarie di $\mathcal{P}$, ossia $\mathcal{P}^{(1)}$ può contenere strettamente l'inviluppo convesso. Le disuguaglianze di $\mathcal{P}^{(1)}$ sono dette tagli di Chvátal di rango 1, quelle di $\mathcal{P}^{(2)}$ che non sono già presenti in $\mathcal{P}^{(1)}$ tagli di Chvátal di rango 2, e così via. Intuitivamente, più alto è il rango di una disuguaglianza più essa è "difficile da ottenere". Infatti, il numero t tale per cui $\mathcal{P}^{(t)} = conv(\,\mathcal{P} \cap \mathbb{Z}^n\,)$, detto il *rango di Chvátal* del problema *PLI*, può essere considerato una sorte di misura di "quanto sia difficile" rappresentare l'inviluppo convesso, ossia risolvere il problema. Si può dimostrare che per problemi di Programmazione 0/1 il rango di Chvátal non è superiore a $O(\,n^2 \log n\,)$, ma esistono problemi per i quali è stato dimostrato un rango di almeno $O(\,n^2\,)$; pertanto, non ci si può aspettare di poter ripetere solamente "poche" volte l'operazione (7.3).

Per quanto interessanti, i risultati appena descritti sono principalmente teorici: non indicano cioé come effettivamente scegliere y per generare i tagli. Per questo discuteremo un caso particolare dei tagli di Chvátal—che, in effetti, è stato ideato precedentemente ad essi—per i quali invece è disponibile una chiara procedura di separazione che si integra perfettamente con gli algoritmi di B&B basati sulla *PL*.

7.4.2 I tagli di Gomory

La separazione dei *tagli di Gomory* avviene una volta determinata la soluzione ottima $\bar{x}$ del rilassamento continuo ed aver verificato che essa non è intera, ossia che per un qualche fissato $i \in \{\,1\,,\,\ldots\,,\,n\,\}$ si ha $\bar{x}_i \notin \mathbb{Z}$. Per separare i tagli di Gomory si usa non solamente la soluzione ottima $\bar{x}$, ma anche (e soprattutto) la *base ottima* B che la definisce, ossia tale per cui $\bar{x} = A_B^{-1} b_B$. In particolare si useranno (il che, intuitivamente, ha senso) solamente i vincoli appartenenti a B per costruire il taglio di Chvátal: in altri termini, $y = [\,y_B\,,\,y_N\,]$ con $y_N = 0$.

Per costruire y_B si seleziona $h = u_i A_B^{-1}$, ossia la *riga* dell'inversa della matrice corrispondente alla variabile frazionaria $\bar{x}_i$: infatti, $hb_B = u_i A_B^{-1} b_B = u_i \bar{x} = \bar{x}_i$. Definiamo quindi

$$y_B = h - \lfloor h \rfloor \ . \tag{7.4}$$

Poiché $\lfloor h \rfloor \leq h$ per definizione, $y_B \geq 0$. Pertanto, ponendo $r = \lfloor h \rfloor \in \mathbb{Z}^n$ si ha che

$$\begin{aligned} a_y &= y_B A_B = (h - r)A_B = u_i A_B^{-1} A_B - rA_B = u_i - rA_B \\ b_y &= y_B b_B = (h - r)b_B = u_i A_B^{-1} b_B - rb_B = \bar{x}_i - rb_B \end{aligned} \tag{7.5}$$

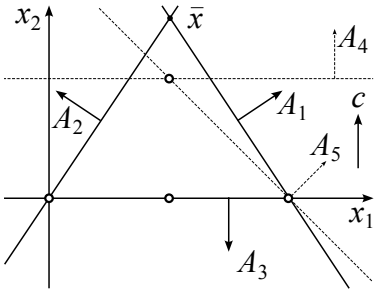
definiscono una disuguaglianza valida. Si noti innanzi tutto che $a_y \in \mathbb{Z}^n$ ($u_i \in \mathbb{Z}^n$, $r \in \mathbb{Z}^n$, $A_B \in \mathbb{Z}^{n \times n}$), ossia y_B definisce un taglio di Chvátal. Affinché il taglio sia "utile" occorre che $b_y \notin \mathbb{Z}$, di modo tale che (7.3) produca un taglio "più forte" di (7.5), che è logicamente implicato dai vincoli in B. È inoltre particolarmente utile che il taglio renda inammissibile $\bar{x}$ in modo da avere la possibilità di far diminuire strettamente la valutazione superiore, analogamente a quanto discusso nel punto 3. del §7.2.4. È facile vedere che entrambe le cose sono vere. Infatti, poiché per definizione $A_B \bar{x} = b_B$, ovviamente si ha che $a_y \bar{x} = b_y$, ossia il taglio è attivo in $\bar{x}$. Ma è anche ovvio che $b_y \notin \mathbb{Z}$: infatti $r \in \mathbb{Z}^n$, $b_B \in \mathbb{Z}^n$ e pertanto $rb_B \in \mathbb{Z}$, ma $\bar{x}_i \notin \mathbb{Z}$. Di conseguenza, il *taglio di Gomory* (un particolare taglio di Chvátal)

$$g = a_y = u_i - rA_B \ \ , \ \ \delta = \lfloor b_y \rfloor = \lfloor \bar{x}_i - rb_B \rfloor \tag{7.6}$$

è violato dalla soluzione ottima $\bar{x}$ del rilassamento continuo, ossia $g\bar{x} > \delta$. Poiché $g \in \mathbb{Z}^n$ e $\delta \in \mathbb{Z}$, aggiungendo il nuovo vincolo alla formulazione del problema si mantiene la proprietà che la matrice dei coefficienti ed il vettore dei lati destri sono interi, il che permette di ripetere il procedimento un numero arbitrario di volte, fintanto che $\bar{x}$ ha componenti frazionarie. In effetti è possibile dimostrare che, analogamente a quanto visto per i tagli di Chvátal—ma stavolta in modo completamente algoritmico—esiste un massimo numero finito t di iterazioni che *l'algoritmo dei tagli di Gomory* può compiere, il che significa che in un numero finito di iterazioni si ottiene una soluzione $\bar{x}$ intera, che quindi risolve il problema *PLI* originario. Si noti che il taglio può essere costruito in modo completamente automatico una volta conosciuta la base B e l'indice della variabile frazionaria i utilizzando informazione (l'inversa della matrice di base o una sua fattorizzazione) che è già calcolata dall'algoritmo del simplesso, e quindi la sua separazione ha un costo trascurabile. Inoltre, è ovviamente possibile in principio costruire più di un taglio di Gomory ad ogni iterazione, uno per ciascuna delle componenti frazionarie di $\bar{x}$.

Esempio 7.7. Applicazione dei tagli di Gomory
Si consideri il semplice problema di *PLI*

$$\begin{array}{lrcrcl} \max & & & x_2 & & \\ & 3x_1 & + & 2x_2 & \leq & 6 \\ & -3x_1 & + & 2x_2 & \leq & 0 \\ & & - & x_2 & \leq & 0 \\ & x_1 & , & x_2 & \in & \mathbb{Z} \end{array}$$



rappresentato geometricamente qui accanto. I punti evidenziati sono le soluzioni intere ammissibili, ed è evidente che la soluzione ottima è $x^* = [1, 1]$, con $cx^* = 1$. $B = \{1, 2\}$ è la base ottima per il rilassamento continuo, infatti

$$A_B = \begin{bmatrix} 3 & 2 \\ -3 & 2 \end{bmatrix} \ , \ A_B^{-1} = \begin{bmatrix} 1/6 & -1/6 \\ 1/4 & 1/4 \end{bmatrix} \ , \ \bar{x} = A_B^{-1} b_B = \begin{bmatrix} 1 \\ 3/2 \end{bmatrix} \ , \ \bar{y}_B = cA_B^{-1} = [1/4, 1/4]$$

e quindi B è sia primale che duale ammissibile. Poiché $c\bar{x} = 3/2$ si ha un gap relativo $(c\bar{x} - cx^*)/cx^* = 0.5$ (il 50%) tra la valutazione superiore del rilassamento continuo ed il valore ottimo del problema.

Calcoliamo adesso un taglio di Gomory. L'unica componente frazionaria di $\bar{x}$ è la seconda, quindi $i = 2$. Si deve considerare quindi la seconda riga di A_B^{-1}, $h = [1/4, 1/4]$, e $r = \lfloor h \rfloor = [0, 0]$. Pertanto

$$g = u_i - rA_B = [0, 1] \ , \ \delta = \lfloor \bar{x}_i - rb_B \rfloor = \lfloor 3/2 - 0 \rfloor = 1 \ .$$

In altri termini, la disuguaglianza $x_2 \leq 1$ (linea orizzontale tratteggiata in figura) è valida per tutte le soluzioni intere, e quindi per il loro inviluppo convesso: come la teoria prevede, $\bar{x}$ viola il taglio di Gomory. Aggiungendolo alla formulazione (come quarto vincolo) e risolvendo di nuovo il rilassamento continuo si ottiene una delle due possibili basi ottime $B = \{1, 4\}$ o $B = \{2, 4\}$, come è ovvio geometricamente ed è facile verificare algebricamente: per la

prima, infatti,

$$A_B = \begin{bmatrix} 3 & 2 \\ 0 & 1 \end{bmatrix} \ , \ A_B^{-1} = \begin{bmatrix} 1/3 & -2/3 \\ 0 & 1 \end{bmatrix} \ , \ \bar{x} = A_B^{-1} b_B = \begin{bmatrix} 4/3 \\ 1 \end{bmatrix} \ , \ \bar{y}_B = cA_B^{-1} = [\,0\,,\,1\,]$$

ossia B è ancora primale e duale ammissibile con $c\bar{x} = 1 = cx^*$, e quindi si è completamente chiuso il gap tra il rilassamento continuo e la soluzione intera. Non si è però ancora ottenuta una "buona descrizione" dell'inviluppo convesso, infatti $\bar{x}$ non è intera e quindi il problema non è ancora stato completamente risolto (si ha una valutazione superiore esatta ma non ancora una valutazione inferiore). È quindi possibile continuare l'algoritmo dei tagli di Gomory: in questo caso la componente frazionaria della soluzione è la prima, e quindi

$$h = u_i A_B^{-1} = [\,1/3\,,\,-2/3\,] \ , \ r = \lfloor h \rfloor = [\,0\,,\,-1\,] \ , \ g = u_i - rA_B = [\,1\,,\,1\,] \ , \ \delta = \lfloor \bar{x}_i - rb_B \rfloor = \lfloor 4/3 - (-1) \rfloor = 2 \ .$$

Ancora una volta, come la teoria predice, il taglio di Gomory $x_1 + x_2 \leq 2$ è violato da $\bar{x} = [\,4/3\,,\,1\,]$. Aggiungendo anche esso alla formulazione come quinto vincolo (linea diagonale tratteggiata in figura) e risolvendo di nuovo il rilassamento continuo è possibile ottenere la base ottima $B = \{\,4\,,\,5\,\}$ a cui corrisponde $\bar{x} = [\,1\,,\,1\,] = c^*$, come è facile verificare algebricamente e geometricamente. Se invece il simplesso determinasse l'altra possibile base ottima, $B = \{\,2\,,\,4\,\}$, alla quale corrisponde una soluzione frazionaria, sarebbe possibile derivare un nuovo taglio di Gomory; è facile verificare che in questo modo si terminerebbe di descrivere completamente l'inviluppo convesso delle soluzioni intere, determinando necessariamente alla successiva soluzione del rilassamento continuo una soluzione intera e quindi ottima per la *PLI*.

Esercizio 7.15. Si determini il taglio di Gomory generato dalla base $B = \{\,2\,,\,4\,\}$.

Il fatto che in teoria l'algoritmo dei tagli di Gomory termini in un numero finito di iterazioni, come l'esempio mostra, non significa che—nella sua versione "pura" qui descritta—sia computazionalmente efficiente: in pratica durante le prime iterazioni la valutazione superiore generata dal rilassamento continuo diminuisce in effetti in modo significativo, ma man mano che l'algoritmo progredisce la diminuzione è sempre meno marcata. Sostanzialmente i tagli divengono "sempre meno profondi", eliminando sezioni del rilassamento continuo che non appartengono all'inviluppo convesso di dimensione man mano inferiore: si dice che l'algoritmo si "attenua progressivamente" (tail off), e per ottenere che la soluzione ottima divenga effettivamente intera sarebbe necessario attendere un numero enorme di iterazioni (questo poi potrebbe non accadere davvero in pratica per via degli errori numerici inerenti nella computazione con numeri in virgola mobile). Nonostante questo, i tagli di Gomory si dimostrano spesso utili nella pratica all'interno di algoritmi B&B basati sulla *PL* in quanto le prime "poche" iterazioni dell'algoritmo riescono comunque a migliorare rapidamente in modo sostanziale la valutazione superiore, permettendo di diminuire il numero di nodi dell'albero delle decisioni che vengono esplorati.

7.4.3 Il "Branch & Cut"

Nella pratica, i tagli di Gomory vengono quindi utilizzati fondamentalmente come uno dei diversi possibili separatori, insieme a quelli già discussi per strutture specifiche. La combinazione della generazione di disuguaglianze valide e del B&B viene definito *Branch & Cut* (B&C), ed è attualmente il metodo generale per la *PLI* di gran lunga più efficiente in pratica. Ciascuno dei due approcci sarebbe in teoria capace di risolvere il problema quando usato da solo, ma in pratica soffre di limiti significativi:

- il B&B basato sulla *PL*, con le formulazioni "naturali" dei problemi, spesso determina valutazioni superiori poco accurate che portano a non riuscire a potare abbastanza nodi da impedire l'esplosione combinatoria dell'albero delle decisioni;
- il problema del "tailing off" sopra descritto per i tagli di Gomory è comune a molti separatori; in altri casi, inoltre, può i separatori possono non essere in grado di trovare tagli violati per la soluzione corrente $\bar{x}$ del rilassamento continuo, anche per via del fatto che vengono utilizzati algoritmi euristici per risolvere il problema di separazione.

Quando usate insieme, però, le due componenti—generazione dei piani di taglio e branching—funzionano in modo sinergico. L'operazione di branching fa sostanzialmente "ripartire" i separatori: accade spesso che dopo il branching in ciascuno dei figli i primi tagli generati possono nuovamente essere efficienti. In effetti si ritiene che nel B&C il ruolo del branching sia fondamentalmente quello di permettere la ripartenza della generazione efficiente dei piani di taglio, ancor più e prima di ottenere un miglioramento delle valutazioni superiori per via della restrizione della regione ammissibile dei sottoproblemi.

A loro volta i tagli permettono di migliorare sensibilmente le valutazioni superiori e quindi ridurre (a volte drammaticamente) il numero di operazioni di branching necessarie a risolvere il problema; inoltre spesso permettono di ottenere soluzioni continue "meno frazionarie" (quando non direttamente intere, risolvendo così i sottoproblemi), e quindi rendono più efficaci le euristiche basate sul rilassamento continuo.

Per cercare di illustrare alcuni degli elementi di complessità degli approcci B&C riportiamo adesso un esempio della soluzione di un problema di *PLI* mediante la combinazione di enumerazione implicita e separazione di disuguaglianze valide.

Esempio 7.8. Esecuzione del Branch & Cut

Si vuole risolvere il problema di *PLI* riportato qui accanto mediante un approccio Branch & Cut. In particolare, il rilassamento è quello continuo, risolto attraverso l'algoritmo del simplesso primale per la prima soluzione al nodo radice, mentre per sfruttare opportunamente le capacità di riottimizzazione degli algoritmi del simplesso (cf. §7.2.1) si utilizza il simplesso duale a partire dalla base ottima precedente per tutte le altre risoluzioni (si noti che per i nodi figli si parte dall'ultima base ottima determinata al nodo padre). Al termine della soluzione del primo rilassamento continuo ad un nodo, se necessario si separa un singolo taglio di Gomory sulla variabile frazionaria di indice minimo e si ripete la soluzione del rilassamento. Per ogni soluzione continua ottima determinata si prova a generare euristicamente una soluzione intera ammissibile attraverso l'arrotondamento all'intero più vicino della componenti frazionarie; se la soluzione così generata non è ammissibile la si scarta. Il branching viene effettuato con la regola di branching standard (7.1) per il B&B basato sulla *PL* sulla variabile frazionaria di indice minimo. Infine si visita in ampiezza l'albero delle decisioni, visitando per primo il nodo destro, ossia quello con aggiunto il vincolo $x_i \geq \lceil x_i^* \rceil$.

$$\begin{array}{lrcrcr} \max & x_1 & & & \leq & -7 \\ & -8x_1 & & & \leq & 0 \\ & x_1 & - & 4x_2 & \leq & 4 \\ & x_1 & + & 4x_2 & \leq & 4 \\ & x_1 & , & x_2 & \in & \mathbb{Z} \end{array}$$

Si inizializza l'algoritmo B&C come quello B&B: si pone in Q il problema originario, corrispondente a non aver preso alcuna decisione, e si pone $z = -\infty$ il valore dell'incumbent, corrispondente a non aver ancora trovato alcuna soluzione ammissibile. Si inizia quindi il "main loop" dell'algoritmo.

Nodo radice Occorre risolvere il rilassamento continuo del problema. Poiché si vuole utilizzare il simplesso primale occorre per prima cosa individuare una base primale ammissibile. Seguendo la procedura del §3.3.1.1 si inizia determinando una qualsiasi base, ossia partendo da $B = \emptyset$ ed inserendo i vincoli in ordine arbitrario (ad esempio, quello naturale), scartando quelli linearmente dipendenti da quelli già inseriti finché non se ne hanno $n = 2$ linearmente indipendenti. In questo caso si ottiene immediatamente $B = \{ 1 , 2 \}$, per la quale

$$A_B = \begin{bmatrix} -8 & 0 \\ 1 & -4 \end{bmatrix} \ , \ A_B^{-1} = \begin{bmatrix} -1/8 & 0 \\ -1/32 & -1/4 \end{bmatrix} \ , \ \bar{x} = A_B^{-1} b_B = \begin{bmatrix} -1/8 & 0 \\ -1/32 & -1/4 \end{bmatrix} \begin{bmatrix} -7 \\ 0 \end{bmatrix} = \begin{bmatrix} 7/8 \\ 7/32 \end{bmatrix}$$

Poiché $\bar{x}$ è ammissibile per il vincolo fuori base non è necessario compiere altre operazioni e si può direttamente far partire l'algoritmo del simplesso primale dalla base B:

$$\bar{y}_B = c A_B^{-1} = [1 , 0] \begin{bmatrix} -1/8 & 0 \\ -1/32 & -1/4 \end{bmatrix} = [-1/8 , 0] \ , \ h = \min\{ i : \bar{y}_i < 0 \} = 1 \ , \ \xi = -A_B^{-1} u_{B(h)} = \begin{bmatrix} 1/8 \\ 1/32 \end{bmatrix}$$

$$A_3 \xi = 1/4 > 0 \ , \ \lambda_3 = (b_3 - A_3 \bar{x}) / A_3 \xi = (4 - 7/4) / (1/4) = 9 \ , \ \bar{\lambda} = \min\{ \lambda_i : i \in N , \ A_i \xi > 0 \} = \lambda_3 = 9$$

$$k = \min\{ i \in N : \lambda_i = \bar{\lambda} \} = \min\{ 3 \} = 3 \ , \ B = B \setminus \{ h \} \cup \{ k \} = \{ 1 , 2 \} \setminus \{ 1 \} \cup \{ 3 \} = \{ 2 , 3 \}$$

Alla seconda iterazione si ha quindi $B = \{ 2 , 3 \}$, per la quale

$$A_B = \begin{bmatrix} 1 & -4 \\ 1 & 4 \end{bmatrix} \ , \ A_B^{-1} = \begin{bmatrix} 1/2 & 1/2 \\ -1/8 & 1/8 \end{bmatrix} \ , \ \bar{x} = A_B^{-1} b_B = \begin{bmatrix} 2 \\ 1/2 \end{bmatrix} \ , \ \bar{y}_B = c A_B^{-1} = [1/2 , 1/2] \geq 0 \ .$$

Pertanto la base è ottima per il rilassamento continuo e si è stabilita la valutazione superiore valida $\bar{z} = c\bar{x} = 2$ sul valore ottimo del problema, ma la soluzione $\bar{x}$ ottenuta non è intera. Arrotondandola all'intero più vicino si ha $x = [2 , 1] \in \mathbb{Z}^2$ che viola il terzo vincolo ($\lceil 1/2 \rfloor = 1$): non si è determinata alcuna soluzione ammissibile, e pertanto rimane $z = -\infty$.

Prima di effettuare il "branching" si tenta quindi il "cutting" generando un taglio di Gomory. L'unica variabile frazionaria è la seconda, quindi $i = 2$. Si ha pertanto

$$h = u_2 A_B^{-1} = [0 , 1] \begin{bmatrix} 1/2 & 1/2 \\ -1/8 & 1/8 \end{bmatrix} = [-1 , 1] \ , \ r = \lfloor h \rfloor = [-1/8 , 1/8]$$

$$g = u_2 - r A_B = [0 , 1] - [-1 , 0] \begin{bmatrix} 1 & -4 \\ 1 & 4 \end{bmatrix} = [1 , -3] \ , \ \delta = \lfloor \bar{x}_i - r b_B \rfloor = \left\lfloor 1/2 - [-1 , 0] \begin{bmatrix} 0 \\ 4 \end{bmatrix} \right\rfloor = 0 \ .$$

Si aggiunge quindi il quarto vincolo $x_1 - 3x_2 \leq 0$ al problema: come previsto dalla teoria, esso è violato dalla precedente soluzione ottima $\bar{x}$. Questo significa che si può iniziare il simplesso duale dalla precedente base ottima $B = \{\, 2\,,\, 3\,\}$ avendo già determinato l'indice entrante $k = 4$ (l'unico vincolo che può essere violato), ottenendo

$$\eta_B = A_k A_B^{-1} = [\, 1\,,\, -3\,] \begin{bmatrix} 1/2 & 1/2 \\ -1/8 & 1/8 \end{bmatrix} = [\, 7/8\,,\, 1/8\,]$$

$$\theta = \min\{\, \bar{y}_i \,/\, \eta_i \,:\, i \in B\,,\ \eta_i > 0\,\} = \min\{\, (1/2)\,/\,(7/8)\,,\, (1/2)\,/\,(1/8)\,\} = \min\{\, 4/7\,,\, 4\,\} = 4/7$$

$$h = \min\{\, i \in B\,;\, \bar{y}_i \,/\, \eta_i = \theta\,\} = \min\{\, 2\,\} = 2 \ ,\ \ B = B \setminus \{\, h\,\} \cup \{\, k\,\} = \{\, 2\,,\, 3\,\} \setminus \{\, 2\,\} \cup \{\, 4\,\} = \{\, 3\,,\, 4\,\} \ .$$

Alla seconda iterazione si ha quindi $B = \{\, 3\,,\, 4\,\}$, per la quale

$$A_B = \begin{bmatrix} 1 & 4 \\ 1 & -3 \end{bmatrix} \ ,\ \ A_B^{-1} = \begin{bmatrix} 3/7 & 4/7 \\ 1/7 & -1/7 \end{bmatrix} \ ,\ \ \bar{x} = A_B^{-1} b_B = \begin{bmatrix} 12/7 \\ 4/7 \end{bmatrix} \ ,\ \ \bar{y}_B = c A_B^{-1} = [\, 3/7\,,\, 4/7\,]$$

$$A_N \bar{x} = \begin{bmatrix} -8 & 0 \\ 1 & -4 \end{bmatrix} \begin{bmatrix} 12/7 \\ 4/7 \end{bmatrix} = \begin{bmatrix} -96/7 \\ -4/7 \end{bmatrix} \leq \begin{bmatrix} -7 \\ 0 \end{bmatrix} = b_N \ ,$$

ossia essendo anche primale ammissibile la base è ottima per il rilassamento continuo. Ancora una volta la soluzione $\bar{x}$ ottenuta non è intera, ed arrotondandola all'intero più vicino si ottiene di nuovo $x = [\, 2\,,\, 1\,]$, intera ma inammissibile. Come ci si aspetta la valutazione superiore è migliorata: $\bar{z} = c\bar{x} = 12/7 < 2$, ma questo non è di particolare beneficio in quanto rimane $z = -\infty$. È quindi necessario procedere col branching: poiché entrambe le variabili sono frazionarie si seleziona $i = 1$ per via della regola stabilita e si creano due figli dell'albero delle decisioni, aggiungendo i vincoli

$$x_1 \leq \lfloor 12/7 \rfloor = 1 \quad \text{oppure} \quad x_1 \geq \lceil 12/7 \rceil = 2 \ .$$

$\boxed{x_1 \geq 2}$ Il nuovo vincolo (nella forma standard $-x_1 \leq -2$) viene aggiunto alla matrice dei coefficienti come quinto vincolo. Si riparte quindi ancora col simplesso duale dalla base ottima $B = \{\, 3\,,\, 4\,\}$ (per la seconda soluzione) del nodo padre, ancora sapendo che $k = 5$: è l'unico vincolo che può essere violato, e siamo sicuri che lo sia perché la regola di branching è costruita appositamente per verificare questa condizione. Pertanto

$$\eta_B = A_k A_B^{-1} = [\, -1\,,\, 0\,] \begin{bmatrix} 3/7 & 4/7 \\ 1/7 & -1/7 \end{bmatrix} = [\, -3/7\,,\, -4/7\,] \ .$$

Poiché $\eta_B \leq 0$, il problema duale è inferiormente illimitato, il che dimostra che il primale—una volta che la sua regione ammissibile è stata ridotta dall'aggiunta del vincolo di branching—è vuoto. Pertanto il nodo viene chiuso per inammissibilità: poiché non esiste nessuna soluzione ammissibile continua, a maggior ragione non ne possono esistere intere. Formalmente si ottiene una valutazione superiore $\bar{z} = -\infty$, e quindi $\bar{z} \leq z$ (anche se $z = -\infty$) ed il nodo può essere chiuso.

$\boxed{x_1 \leq 1}$ Il nuovo vincolo viene aggiunto alla matrice dei coefficienti come quinto vincolo, avendo rimosso quello $x_1 \geq 2$ del nodo "fratello". Si ripristina quindi anche l'ultima base ottima $B = \{\, 3\,,\, 4\,\}$ del nodo padre (che in effetti non è cambiata durante la risoluzione del fratello, ma questo è un caso) e si riparte da essa col simplesso duale sapendo che $k = 5$ (per la nuova e diversa versione del vincolo 5). Pertanto

$$\eta_B = A_k A_B^{-1} = [\, 1\,,\, 0\,] \begin{bmatrix} 3/7 & 4/7 \\ 1/7 & -1/7 \end{bmatrix} = [\, 3/7\,,\, 4/7\,]$$

$$\theta = \min\{\, (3/7)\,/\,(3/7)\,,\, (4/7)\,/\,(4/7)\,\} = \min\{\, 1\,,\, 1\,\} = 1 \ ,\ \ h = \min\{\, 3\,,\, 4\,\} = 3 \ ,\ \ B = \{\, 4\,,\, 5\,\} \ .$$

Alla seconda iterazione si ha quindi

$$A_B = \begin{bmatrix} 1 & -3 \\ 1 & 0 \end{bmatrix} \ ,\ \ A_B^{-1} = \begin{bmatrix} 0 & 1 \\ -1/3 & 1/3 \end{bmatrix} \ ,\ \ \bar{x} = A_B^{-1} b_B = \begin{bmatrix} 1 \\ 1/3 \end{bmatrix} \ ,\ \ \bar{y}_B = c A_B^{-1} = [\, 0\,,\, 1\,] \geq 0$$

$$A_N \bar{x} = \begin{bmatrix} -8 & 0 \\ 1 & -4 \\ 1 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 1/3 \end{bmatrix} = \begin{bmatrix} -8 \\ -1/3 \\ 7/3 \end{bmatrix} \leq \begin{bmatrix} -7 \\ 0 \\ 4 \end{bmatrix} = b_N \ ,$$

ossia la base è ottima. Si noti che si è ottenuta un'ulteriore riduzione della valutazione superiore globale: poiché $c\bar{x} = 1$, possiamo affermare che $\bar{z} = \max\{\, 1\,,\, -\infty\,\} = 1 < 12/7$ è una valutazione superiore globalmente valida sul valore ottimo del problema (è il massimo delle valutazione superiori dei due figli della radice, che complessivamente sono garantiti contenere tutte le soluzioni ammissibili—se ve ne sono). Non si dispone però di una valutazione inferiore che permetta di chiudere il nodo, in quanto l'arrotondamento di $x = \lceil \bar{x} \rfloor = [\, 1\,,\, 0\,]$ non è ammissibile (viola il secondo vincolo). Prima di effettuare il branching, secondo la strategia stabilita proviamo il cutting: l'unica variabile frazionaria è la seconda, quindi $i = 2$ e si ha

$$h = u_2 A_B^{-1} = [\, 0\,,\, 1\,] \begin{bmatrix} 0 & 1 \\ -1/3 & 1/3 \end{bmatrix} = [\, -1/3\,,\, 1/3\,] \ ,\ \ r = \lfloor h \rfloor = [\, -1\,,\, 0\,]$$

$$g = u_2 - rA_B = [\,0\,,\,1\,] - [\,-1\,,\,0\,] \begin{bmatrix} 1 & -3 \\ 1 & 0 \end{bmatrix} = [\,1\,,\,-2\,] \;\;,\;\; \delta = \lfloor \bar{x}_i - rb_B \rfloor = \left\lfloor 1/3 - [\,-1\,,\,0\,] \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right\rfloor = 0 \;\;.$$

Come la teoria prevede, il taglio di Gomory è violato dalla soluzione del rilassamento continuo: $g\bar{x} = 1/3 > 0 = \delta$. Possiamo quindi aggiungerlo come sesto vincolo alla formulazione e ripartire col simplesso duale dalla base precedente, avendo che $k = 6$: quindi

$$\eta_B = A_k A_B^{-1} = [\,1\,,\,-2\,] \begin{bmatrix} 1 & -3 \\ 1 & 0 \end{bmatrix} = [\,-1\,,\,-3\,] \leq 0$$

il che dimostra che il nuovo problema duale è inferiormente illimitato, ossia che il primale è vuoto. Formalmente si è di nuovo determinata una valutazione superiore $-\infty = \bar{z} \leq z = -\infty$ sul valore ottimo del sottoproblema corrispondente a questo nodo, che quindi può essere potato (per inammissibilità).

Poiché $Q = \emptyset$, l'algoritmo termina: $z = -\infty$ è il valore ottimo del problema, il che implica che non esisteva nessuna soluzione ammissibile intera, anche se ne esistevano di frazionarie per il rilassamento continuo. È facile verificare che senza l'aggiunta dei tagli di Gomory il rilassamento continuo relativo al secondo figlio sarebbe stato ancora non vuoto, il che avrebbe richiesto ulteriori operazioni di branching per giungere alla terminazione dell'algoritmo.

Esercizio 7.16. Si rappresenti graficamente il problema in esame e si verifichi geometricamente l'esattezza delle relazioni ottenute algebricamente.

Esercizio 7.17. Si risolva il problema di *PLI* dato con il metodo dei tagli di Gomory "puro" (ossia senza effettuare branching) e, separatamente, con il B&B basato sulla *PL* "puro" (ossia, senza generare tagli), discutendo quale degli approcci risulti più efficiente.

Nel complesso, la combinazione della due strategie risolutive (enumerazione implicita e separazione di disuguaglianze valide) ha permesso di incrementare in modo molto significativo le prestazioni dei solutori general-purpose di *PLI*, ma è bene rimarcare il fatto che rendere efficiente ed efficace questo processo richiede molte scelte non ovvie: su quali delle (potenzialmente molte) strutture specifiche far girare i separatori, che approcci risolutivi utilizzare per risolvere i problemi di ottimizzazione (spesso $\mathcal{NP}$-ardui) corrispondenti, quanti tagli generare prima di risolvere nuovamente il rilassamento continuo, ecc.. Queste scelte vengono tipicamente prese sulla base di vasti esperimenti computazionali su grandi insiemi di istanze "benchmark", e non possono essere discusse in queste dispense; in pratica si riescono a trovare scelte che (eventualmente grazie all'uso di regole euristiche o persino di tecniche di intelligenza artificiale) consentono di migliorare notevolmente le prestazioni per la soluzione di molte istanze provenienti da applicazioni reali. Per ulteriori approfondimenti su questi aspetti si rimanda alla letteratura citata ed ai corsi avanzati dell'area di Ricerca Operativa.

7.5 Programmazione dinamica

La *Programmazione Dinamica* (PD) è uno schema risolutivo generale che si applica a problemi con una particolare struttura. Non si vuole in questa sede dare una definizione formale in quanto, come vedremo, siamo particolarmente interessati alla relazione che esiste tra la PD ed il B&B, ma in modo molto generale l'approccio si applica a problemi per cui:

- la soluzione può essere costruita in un certo numero finito di *stadi*, in un ordine dato;
- la soluzione ottima corrispondente allo stadio $i + 1$ si ottiene "facilmente" una volta nota la soluzione ottima corrispondente allo stadio i (sostanzialmente esiste una definizione ricorsiva sugli stadi delle condizioni di ottimalità).

Un esempio di questo processo—utile per la discussione che faremo—è quello del cammino minimo su grafo acicilco (cf. §2.2.6): gli stadi sono i nodi del grafo, sui quali è definito l'ordinamento corrispondente ad una buona numerazione. Al momento della visita del nodo (stadio) i-esimo si è già ottenuta la soluzione ottima per quello e tutti gli stadi precedenti, e mediante questa si è certi (attraverso il controllo delle condizioni di Bellman) di costruire la soluzione ottima per il successivo stadio $(i + 1)$-esimo.

Senza approfondire ulteriormente questo aspetto illustriamo adesso le relazioni tra PD e B&B descrivendo un approccio di PD per il problema dello zaino (KP). Questo approccio richiede che i pesi a_i degli oggetti dello zaino siano numeri interi, e di conseguenza che anche la sua capacità b lo sia

(altrimenti, come visto nella sezione precedente, basta ridefinire b come la sua parte intera inferiore). Anche in questo caso l'assunzione può in teoria essere fatta senza perdita di generalià in quanto i numeri reali vengono solitamente rappresentati sui computer in formato a virgola mobile, e quindi sono razionali (ed in ultima analisi interi). In questo caso in realtà "quanto frazionari sono i numeri a_i" ha un impatto significativo sull'algoritmo, ma questo sarà discusso in seguito; per il momento assumiamo semplicemente che $a_i \in \mathbb{N}_+$ per ogni $i \in I$, e $b \in \mathbb{N}_+$.

L'algoritmo di PD per il (KP) può essere descritto come un *problema di cammino massimo su un opportuno grafo aciclico*. Mostreremo adesso come costruire il grafo: la costruzione avverrà in un ordine compatibile con la buona numerazione, il che significa che è possibile contemporaneamente anche visitare il grafo ed aggiornare le etichette secondo l'Algoritmo 2.2.6 e quindi risolvere (KP) nel tempi necessario a costruire il grafo.

I nodi del grafo sono caratterizzati da coppie (i, t), dove $i \in I = \{1, \ldots, n\}$ è un (indice di un) oggetto dello zaino e $t \in \mathbb{N}$ è un valore di occupazione dello zaino (ossia, $0 \leq t \leq b$). A questi si aggiunge il nodo "sorgente" $s = (0, 0)$ ed il nodo "destinazione" d. Il significato del nodo (i, t) è il seguente: si sono prese decisioni su tutti gli oggetti $j = 1, 2, \ldots, i$—ossia, per ciascuno di essi si è già stabilito se inserirlo oppure no dello zaino—ed il risultato di tali decisioni è che l'occupazione dello zaino è pari a t.

La costruzione—e visita—del grafo $G = (N, A)$ procede come segue. Si inizializza una lista di nodi $Q = \{s\}$ e si pone $N = \emptyset$, $A = \{d\}$. Si esegue quindi un ciclo in cui, finché Q non è vuoto, si estrae il primo nodo (i, t) da Q e si aggiunge ad N. A questo punto:

- se $i = n$ si aggiunge ad A l'arco $((n, t), d)$ di costo 0;
- se $i < n$ si aggiunge sicuramente ad A l'arco $((i, t), (i+1, t))$, detto *arco orizzontale*, di costo 0; il nodo $(i+1, t)$ viene aggiunto a Q (se già non vi appartiene);
- se $i < n$ e $t + a_i \leq b$ si aggiunge ad A l'arco $((i, t), (i+1, t+a_i))$, detto *arco diagonale*, di costo c_i; il nodo $(i+1, t+a_i)$ viene aggiunto a Q (se già non vi appartiene).

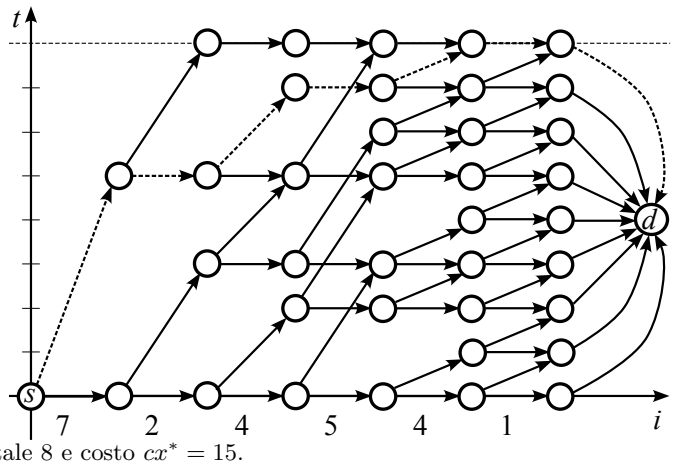
È facile vedere che:

- $G = (N, A)$ è aciclico ed il buon ordinamento è quello lessicografico su (i, t), con d come nodo finale;
- qualsiasi cammino da s a d su G contiene esattamente $n + 2$ nodi: s, uno del tipo (i, t) per ciascun $i \in I$, e d;
- i cammini da s a d su G sono in corrispondenza biunivoca con le soluzioni ammissibili di (KP): i significato dell'arco orizzontale $((i, t), (i+1, t))$ è che l'oggetto i non viene inserito nello zaino, che pertanto mantiene l'occupazione corrente t, mentre l'arco diagonale $((i, t), (i+1, t+a_i))$ significa che l'oggetto i viene inserito nello zaino, la cui occupazione passa pertanto da t a $t+a_i$;
- il costo di ciascun s a d su G è pari al costo della corrispondente soluzione ammissibile.

Pertanto (KP) è equivalente al problema del cammino massimo da s a d su G. Poichè G è aciclico il problema ha sicuramente soluzione ottima finita che può essere determinata semplicemente visitandolo (mentre lo si costruisce), mantenendo aggiornate le opportune etichette $d[\cdot]$ dei nodi. Poiché G ha al più $n(b+1)+2$ nodi e ciascun nodo ha al più due archi uscenti, la cardinalità di A è $O(nb)$ e quindi questo è il costo di risolvere il problema del cammino massimo, ossia di risolvere (KP).

Esempio 7.9. Costruzione del grafo della PD per (KP)
Consideriamo un'istanza del problema dello zaino ed il corrispondente grafo G:

$$\begin{array}{lccccccccccccc} \max & 7x_1 & + & 2x_2 & + & 4x_3 & + & 5x_4 & + & 4x_5 & + & x_6 & & \\ & 5x_1 & + & 3x_2 & + & 2x_3 & + & 3x_4 & + & x_5 & + & x_6 & \leq & 8 \\ & x_1 & , & x_2 & , & x_3 & , & x_4 & , & x_5 & , & x_6 & \in & \{0, 1\} \end{array}$$



I nodi del grafo sono posizionati su un piano cartesiano la cui ascissa corrisponde ad i e la cui ordinata corrisponde a t, il che giustifica gli aggettivi "orizzontale" e "diagonale" ai due diversi tipi di archi. La riga tratteggiata orizzontale corrisponde a $t = 8$, che è la "massima altezza dei nodi del grafo". Per efficacia di rappresentazione non sono mostrati tutti i costi degli archi: quelli orizzontali sono sempre 0, quelli diagonali al livello i hanno tutti lo stesso costo c_i, mostrato in basso. È evidente come qualsiasi cammino sul grafo da s a d corrisponda ad una sequenza di decisioni su tutti gli oggetti che non viola la capacità dello zaino, e quindi ad una soluzione ammissibile di (KP). In particolare, gli archi tratteggiati descrivono la soluzione ottima $x^* = [\,1\,,\,0\,,\,1\,,\,0\,,\,1\,,\,0\,]$, con peso totale 8 e costo $cx^* = 15$.

Esercizio 7.18. Si scriva formalmente, in pseudo-codice, una procedura che risolve il problema (KP) costruendo e visitando il grafo G sopra descritto. L'algoritmo deve produrre, al termine, una descrizione della soluzione ottima di (KP) sotto forma del classico vettore $x \in \{\,0\,,\,1\,\}^n$. Si discuta la complessità sia in tempo che in memoria dell'algoritmo. Si discuta infine se la complessità in memoria potrebbe essere ridotta qualora l'output dell'algoritmo dovesse essere solamente il *valore ottimo* di (KP) e non la sua soluzione ottima (abbiamo visto nell'Esempio 4.8 che il problema di determinare il solo valore ottimo è polinomialmente equivalente a (KP)).

È interessante discutere la relazione tra il grafo G così costruito e gli alberi delle decisioni per il problema (KP). In particolare, si consideri l'albero T delle decisioni "naïve" in cui al livello i si prendono sempre ed unicamente decisioni sulla variabile x_i, analogamente a quello mostrato nell'Esempio 7.1. È facile verificare che G è un "ripiegamento" (folding) di T ottenuto nel seguente modo: tutti i nodi del livello i di T ai quali corrisponde un'occupazione dello zaino pari a $t \leq b$ sono rappresentati dal singolo nodo $(\,i\,,\,t\,)$ di G. Invece, i nodi di T ai quali corrisponde un'occupazione dello zaino superiore a b non sono rappresentati in G. Questi due aspetti giustificano che G sia "più compatto" di T: infatti, nell'Esempio 7.9 G ha 39 nodi, mentre un albero binario completo con 7 livelli (compresa la radice) ne avrebbe 127. Quindi, visitare G può essere più efficiente che non visitare T.

Un'importante differenza tra G e T, però, è che il secondo è un albero mentre il primo no: è possibile raggiungere lo stesso nodo $(\,i\,,\,t\,)$ con più cammini, ed infatti la determinazione della soluzione ottima in G non può essere fatta con una normale visita (ancorché l'Algoritmo 2.2.6 abbia fondamentalmente la stessa complessità). È però possibile facilmente decidere quale dei cammini che raggiungono $(\,i\,,\,t\,)$ sia preferibile: semplicemente, è quello col costo maggiore (poiché il problema è di massimizzazione). Ciò in essenza corrisponde all'applicazione di una *regola di dominanza* (cf. §7.2.3) tra nodi di T che permette di evitare di visitarne alcuni. La regola è la seguente: tra due nodi n' ed n'' allo stesso livello i di T, e quindi nei quali si sono prese decisioni sullo stesso insieme $\{\,1\,,\,2\,,\,\ldots\,,\,i\,\}$ di oggetti, *se le soluzioni parziali di n' ed n'' hanno la stessa occupazione t* allora è possibile evitare di visitare quello tra n' ed n'' la cui soluzione parziale ha costo minore (se le due soluzioni parziali hanno lo stesso costo è possibile scegliere arbitrariamente). Questo è facile da giustificare: qualsiasi modo ammissibile di completare la soluzione parziale di n' scegliendo tra gli oggetti rimanenti $\{\,i+1\,,\,i+2\,,\,\ldots\,,\,n\,\}$ produce una soluzione ammissibile anche partendo dalla soluzione parziale di n'', e l'incremento del valore della funzione obiettivo è lo stesso. Pertanto, se (ponendo) la soluzione parziale di n' ha un costo superiore a quella di n'', per qualsiasi soluzione ammissibile nel sottoalbero di n'' ne esiste una nel sottoalbero di n' che ha costo migliore. Pertanto, purché si esamini esaustivamente—anche implicitamente—il sottoalbero di radice n', non è necessario esaminare il sottoalbero di radice n''.

In altri termini, l'algoritmo di PD sopra descritto può essere considerato un algoritmo di B&B per (KP) in cui l'albero delle decisioni è organizzato in modo tale da rendere efficiente il controllo della regola di dominanza sopra descritta. In questo caso, l'algoritmo non utilizza in effetti alcun rilassamento o euristica, e quindi visiterebbe tutto T se non fosse per l'applicazione della regola di dominanza; questa da sola è in grado (se "b è piccolo", come discusso nel seguito), di rendere efficiente la determinazione

di una soluzione ottima. È anche possibile costruire versioni più sofisticate dell'approccio in cui vengono utilizzati sia rilassamenti ed euristiche che la regola di dominanza, ma non discuteremo questa possibilità nel dettaglio.

Il principale limite della tecnica illustrata è ovviamente la richiesta che b, e quindi gli a_i siano numeri interi e "piccoli". L'interezza di per sé non sarebbe un problema, dato che i numeri reali sui computer vengono tipicamente rappresentati in virgola mobile, e quindi sono di fatto numeri razionali. Pertanto, dati gli $a_i = r_i / d_i \in \mathbb{Q}$, ossia $r_i \in \mathbb{Z}$ e $d_i \in \mathbb{Z}$, è possibile costruire un'istanza di (KP) equivalente i cui pesi $\bar{a}_i$ sono interi semplicemente calcolando il Minimo Comune Multiplo (MCM) dei d_i e moltiplicando tutti gli a_i e b per quello; a quel punto, se b è frazionario può essere arrotondato all'intero inferiore. Questa operazione è possibile, ma i pesi $\bar{a}_i$ possono facilmente essere "molto grandi". Un modo leggermente diverso di vedere il problema è quello di immaginare che i pesi a_i derivino in qualche modo da "misure nel mondo reale", e occorra stabilire quante cifre decimali della misura siano necessarie per garantire che la soluzione ottenuta sia significativa per la situazione reale che (KP) rappresenta. Se sono necessarie k cifre decimali occorre moltiplicare tutti i pesi per 10^k per renderli interi, il che facilmente produce numeri interi di dimensione molto elevata. Queste considerazioni portano naturalmente alla seguente domanda: se gli $\bar{a}_i \in \mathbb{N}$ utilizzati hanno un "piccolo errore" rispetto a quelli che si dovrebbero utilizzare (ad esempio, hanno solo $h \ll k$ cifre significative corrette), che tipo di errore ci si può attendere sul valore ottimo del problema? È possibile mostrare che un "piccolo errore nei pesi" si traduce in un "piccolo errore nella soluzione"? La risposta a questo quesito è parzialmente positiva, come brevemente discusso nel seguito.

Prima di affrontare la domanda appena formulata notiamo che la tecnica vista per (KP) può essere utilizzata in modi diversi, e per diversi problemi.

Esercizio 7.19. Assumendo che i costi c_i di (KP) siano interi si sviluppi un algoritmo di PD, analogo a quello visto, che determina una soluzione ottima di (KP) in $O(\, n \sum_{i \in I} c_i \,)$

Esercizio 7.20. Si sviluppi un algoritmo di PD per il problema del Cammino Minimo Vincolato (CSP) in cui le lunghezze l_{ij} degli archi siano numeri interi, e pertanto anche L possa essere assunto tale (suggerimento: si costruisca un opportuno grafo aciclico ottenuto replicando $L + 1$ volte il grafo del problema).

Esercizio 7.21. Si estenda l'algoritmo di PD dell'esercizio precedente al problema (CSP) con vincoli multipli, ossia per il quale esista per ogni arco un vettore $[\, l_{ij}^k \,]_{k \in K}$ di lunghezze, ed il cammino selezionato non possa avere per ciascun k lunghezza k-esima superiore ad un valore dato L^k. Si discuta in particolare l'effetto di $|K|$ sulla complessità dell'approccio.

Per di terminare questa sezione discutiamo brevemente alcune interessanti implicazioni generali dei concetti appena visti, che rispondono parzialmente alla domanda precedentemente formulata. Innanzi tutto, la complessità $O(\, nb \,)$ dell'approccio di PD non è polinomiale, poiché un'istanza di (KP) può essere descritta con $O(\, n \log(\, b \,) \,)$ bits. Una tale complessità è *pseudopolinomiale*, ed indica che la difficoltà del problema è potenzialmente influenzata dalla magnitudine dei numeri che appaiono come costanti nella dimensione dell'istanza; in pratica, questo approccio può risultare efficiente solamente se "b è piccolo" (e, quindi, anche i pesi a_i lo sono). L'esistenza di algoritmi pseudopolinomiali per un problema di ottimizzazione $\mathcal{NP}$-arduo però non è affatto ovvia: in effetti questa indica che (KP) è in qualche senso "tra i più facili dei problemi difficili". Ciò di deduce in particolare dal seguente risultato: *l'esistenza di un algoritmo pseudopolinomiale per (KP) implica l'esistenza di uno schema di approssimazione pienamente polinomiale per il problema.*

Per dimostrare questo risultato conviene utilizzare non l'algoritmo di PD che abbiamo visto, ma quello dell'Esercizio 7.19. L'idea del procedimento è semplice: data una qualsiasi istanza di (KP) i cui costi non siano interi (e neanche i pesi), si definisce una nuova istanza di (KP) con gli stessi pesi a_i e volume b, ma costi modificati

$$\bar{c}_i = \lfloor nc_i \,/\, (\, \varepsilon C \,) \rfloor \qquad i \in I$$

in cui $C = \max\{\, c_i \,:\, i \in I \,\}$ e $\varepsilon > 0$ è l'accuratezza richiesta dalla soluzione. La motivazione di questa

scelta è che, a meno di un fattore di scala, "i costi cambiano poco": infatti

$$c_i(\, n \,/\, (\, \varepsilon C \,)\,) \leq \lfloor n c_i \,/\, (\, \varepsilon C \,) \rfloor + 1 \leq \bar{c}_i + 1 \ ,$$

da cui

$$c_i - \bar{c}_i(\, \varepsilon C \,/\, n \,) \leq \varepsilon C \,/\, n \ . \tag{7.7}$$

Definiti i $\bar{c}_i$, si ottiene una soluzione ottima della nuova istanza mediante l'algoritmo DP che sfrutta l'integralità dei costi. È possibile vedere che:

- la complessità dell'algoritmo è $O(\, n^3 \,/\, \varepsilon \,)$;
- la soluzione intera determinata è affetta da un errore relativo di al più ε.

La prima proprietà è banale: poiché $\bar{c}_i \leq n \,/\, \varepsilon$, $\sum_{i \in I} \bar{c}_i \leq n^2 \,/\, \varepsilon$. Per quanto riguarda la seconda proprietà, chiamiamo $\bar{x}$ la soluzione ammissibile determinata dall'approccio e x^* la reale soluzione ottima del problema: ovviamente vale $cx^* \geq c\bar{x}$. Ma è anche vero che

$$\bar{c}\bar{x} = \textstyle\sum_{i \in I} \bar{c}_i \bar{x}_i = \sum_{i \in I} \lfloor n c_i \,/\, (\, \varepsilon C \,) \rfloor \bar{x}_i \leq [\, n \,/\, (\, \varepsilon C \,)\,] \sum_{i \in I} c_i \bar{x}_i = [\, n \,/\, (\, \varepsilon C \,)\,] c\bar{x} \ .$$

Dalla relazione precedente si ottiene

$$cx^* - c\bar{x} \leq cx^* - [\, \varepsilon C \,/\, n \,] \bar{c}\bar{x} \ .$$

Ma $\bar{x}$ è ottima per il problema con i costi modificati, il che significa che $\bar{c}\bar{x} \geq \bar{c}x^*$: di conseguenza

$$cx^* - c\bar{x} \leq cx^* - [\, \varepsilon C \,/\, n \,] \bar{c}x^* = \textstyle\sum_{i \in I} (\, c_i - [\, \varepsilon C \,/\, n \,] \bar{c}_i \,) x_i^* \ ,$$

che usando (7.7) fornisce

$$cx^* - c\bar{x} \leq \textstyle\sum_{i \in I} x_i^* [\, \varepsilon C \,/\, n \,] \leq \varepsilon C \ .$$

Poiché ovviamente $cx^* \geq C$ (la soluzione in cui è selezionato il singolo oggetto di costo massimo è ammissibile), si ottiene infine

$$(\, cx^* - c\bar{x} \,) \,/\, cx^* \leq \varepsilon$$

come annunciato.

Il risultato appena descritto non è valido solo per (KP): è infatti possibile mostrare che, sotto opportune ipotesi, l'esistenza di uno schema di approssimazione pienamente polinomiale è in effetti *equivalente* a quella di un algoritmo pseudopolinomiale. I problemi per cui queste due condizioni sono verificate sono "i più facili dei problemi difficili", e vengono chiamati *debolmente $\mathcal{NP}$-ardui*. È facile vedere che molti problemi $\mathcal{NP}$-ardui non hanno questa proprietà (a meno che $\mathcal{P} = \mathcal{NP}$), e quindi sono "più difficili" dei problemi debolmente $\mathcal{NP}$-ardui. Si pensi ad esempio ai "problemi senza numeri" come Max-SAT (cf. Esempio 1.9): il valore della funzione obiettivo è limitato polinomialmente nelle dimensioni del problema (in particolare, dal numero m di clausole). Poiché il valore ottimo è un numero intero, qualsiasi schema di approssimazione con errore $\varepsilon < 1 \,/\, m$ risolverebbe il problema all'ottimo, ma se esso fosse pienamente polinomiale questo avrebbe complessità polinomiale nelle dimensioni del problema, e quindi sarebbe $\mathcal{P} = \mathcal{NP}$. Per tali problemi non è quindi ragionevole attendersi l'esistenza di algoritmi pseudopoliomiali, o equivalentemente schemi di approssimazione pienamente polinomiali. In effetti, si dimostra che molti problemi $\mathcal{NP}$-ardui non hanno neanche schemi di approssimazione: esiste un valore fissato ε tale per cui ottenere una soluzione con errore inferiore è un problema $\mathcal{NP}$-arduo. Non possiamo in questa sede addentrarci ulteriormente in questi affascinanti argomenti, anche se ci auguriamo che le brevi note che abbiamo incluso suscitino la curiosità dei lettori, che potrà essere soddisfatta consultando i riferimenti bibliografici o seguendo corsi avanzati dell'area della Ricerca Operativa o degli Algoritmi.

Riferimenti Bibliografici

M. Conforti, G. Cornuéjols, G. Zambelli "Integer Programming" *Springer*, 2014

G. Gamrath, T. Koch, A. Martin, M. Miltenberger, D. Weninger "Progress in presolving for mixed integer programming" *Mathematical Programming Computation* 7, 367—398, 2015

T. Koch, T. Berthold, J. Pedersen, C. Vanaret "Progress in mathematical programming solvers from 2001 to 2020" *EURO Journal on Computational Optimization* 10, 100031, 2022

T. Koch, T. Ralphs, Y. Shinano "Could we use a million cores to solve an integer program?" *Mathematical Methods of Operations Research* 76, 67-93, 2012

T. Ralphs, Y. Shinano, T. Berthold, T. Koch "Parallel Solvers for Mixed Integer Linear Optimization", in *Handbook of Parallel Constraint Reasoning*, 283—336, 2018

L. Wolsey "Integer Programming" *Wiley*, 1998

K. Marriott, P.J. Stuckey "Programming with Constraints: An Introduction" *MIT Press*, 1998

Conclusioni

Queste dispense hanno cercato di fornire al lettore un primo sguardo alle metodologie della Ricerca Operativa. Ciò richiede di compiere scelte arbitrarie e sicuramente criticabili, in quanto la Ricerca Operativa è caratterizzata dalla grande eterogeneità sia degli strumenti matematici utilizzati che delle applicazioni che di essi traggono beneficio. Gli argomenti presentati sono quelli generalmente considerati adatti ai corsi di base della materia: Programmazione Lineare, algoritmi su grafi, algoritmi di enumerazione implicita. Il taglio della presentazione privilegia l'aspetto algoritmico, coerentemente con l'orientamento verso studenti dell'area informatica, senza rinunciare ad un certo livello di approfondimento degli aspetti più teorici e matematici, utile per studenti di corsi di laurea in matematica, ed agli aspetti legati alla modellazione di situazioni (ir)reali, di interesse per studenti di ambiti più aplicativi quali le ingegnerie.

Il materiale presentato non ha certamente la pretesa di essere esaustivo, neanche per gli argomenti che vengono maggiormente approfonditi. Aspetti che non sono minimamente affrontati, e che invece sono di grande rilevanza pratica, sono ad esempio:

- la presenza di elementi nonlineari nei modelli di ottimizzazione;
- le problematiche relative all'incertezza dei dati di input;
- gli aspetti pratici dell'utilizzo di strumenti software per la soluzione di problemi di ottimizzazione (solutori, linguaggi e sistemi di modellazione);
- una descrizione dettagliata dei modelli che derivano dall'applicazioe delle metodologie descritte in moltissimi campi dell'attività umana quali i trasporti e la logistica, le telecomunicazioni, l'energia, l'economia e la finanza, l'informatica, l'intelligenza artificiale, la scienze fisiche e biologiche, e molti altri.

Ciò nonostante gli estensori sperano che le dispense possano servire sia a fornire un utile materiale di riferimento che a stimolare l'interesse dei lettori verso i molteplici aspetti di questa interessante branca della matematica applicata, che ha il suo principale punto di interesse nel beneficiare di metodologie provenienti da ambiti molto diversi della matematica e delle scienze (analisi, geometria, teoria dei numeri, combinatoria, probabilità, informatica, fisica, ...) e, a sua volta, di prestare le sue metodologie alle più diverse applicazioni. I lettori interessati ad approfondire questi argomenti possono utilizzare, tra i molti altri, i riferimenti bibliografici forniti ed i corsi più avanzati dell'area della Ricerca Operativa.

Come tutti gli aspetti della conoscenza umana la Ricerca Operativa è in continua evoluzione, e gli aspetti che possono al momento attuale sembrare assodati potrebbero in futuro essere profondamente rimessi in discussione. Questo è particolarmente vero in un campo in cui molta della ricerca si basa sull'ipotesi che $\mathcal{P} \neq \mathcal{NP}$; il solo fatto che di questo non si sia in grado di dare una dimostrazione, in un senso o nell'altro, mostra quanto ancora ci sia da capire a da scoprire. In particolare, negli anni più recenti sono emerse prepotentemente due linee di ricerca i cui risultati potrebbero influire in modo estremamente significativo sulle metodologie per la soluzione di problemi di ottimizzazione:

- la computazione quantistica;
- l'apprendimento automatico / intelligenza artificiale.

Allo stato attuale nessuna delle due pare essere in grado di mettere profondamente in discussione il fatto che "la maggioranza dei problemi di ottimizzazione siano molto difficili da risolvere", ma ciascuna delle due potrebbe portare (individualmente, o anche in sinergia) a cambiamenti radicali nella pratica della soluzione di questi problemi. Del resto, negli oltre 20 anni trascorsi dalla prima stesura di queste dispense l'evoluzione delle metodologie è stata comunque estremamente significativa. Come avvenuto in passato, comunque, ci si può aspettare che la Ricerca Operativa beneficerà degli sviluppi in queste (ed altre) branche della scienza e della tecnologia, ma anche che le metodologie della Ricerca Operativa saranno per esse utili; questo è già vero oggi. Per quanto il futuro sia incerto, e nessuno possa prevedere quali sviluppi avrà l'avventura della conoscenza umana, non pare irragionevole aspettarsi che la Ricerca Operativa continuerà ad avere il proprio—piccolo, ma significativo—ruolo in essa.

Appendice A

Algoritmi e complessità

In questa appendice vogliamo brevemente richiamare alcuni concetti fondamentali della teoria della complessità computazionale, utili per meglio comprendere la diversa "difficoltà" della soluzione dei problemi di ottimizzazione.

A.1 Modelli computazionali

Una volta che un problema P sia stato formulato, deve essere risolto: siamo quindi interessati alla messa a punto di strumenti di calcolo che, data una qualsiasi istanza p, siano in grado di fornirne una soluzione in un tempo finito. Tali strumenti di calcolo si chiamano *algoritmi*. Un algoritmo che risolve P può essere definito come una sequenza finita di istruzioni che, applicata ad una qualsiasi istanza p di P, si arresta dopo un numero finito di passi (ovvero di computazioni elementari), fornendo una soluzione di p oppure indicando che p non ha soluzioni ammissibili. Per poter studiare gli algoritmi dal punto di vista della loro efficienza, o complessità computazionale, è necessario definire un modello computazionale: classici modelli computazionali sono la Macchina di Turing (storicamente il primo proposto), la R.A.M. (Random Access Machine), la Macchina a Registri (MR), etc.

La Macchina a Registri è un buon compromesso tra semplicità e versatilità: una MR consiste di un numero (finito ma non limitato) di registri, ciascuno dei quali può contenere un singolo numero intero, e di un programma, ossia di una sequenza finita di istruzioni del tipo

- incrementa il registro k e salta all'istruzione j;
- decrementa il registro k e salta all'istruzione j;
- se il registro k contiene 0 salta all'instruzione j, altrimenti salta all'istruzione h.

Si tratta di una macchina *sequenziale e deterministica*, poiché il comportamento futuro della macchina è univocamente determinato dalla sua configurazione presente. Una MR è un buon modello astratto di un calcolatore elettronico, ed è quindi in grado di compiere tutte le computazioni possibili in un qualunque sistema di calcolo attualmente noto (se si eccettuano i computer quantistici). D'altra parte, si può dimostrare che la classe delle funzioni computabili da una MR è equivalente alla classe delle funzioni computabili da una Macchina di Turing, il che, secondo la *Tesi di Church*, implica che le MR siano presumibilmente in grado di calcolare qualsiasi funzione effettivamente computabile con procedimenti algoritmici.

A.2 Misure di complessità

Dato un problema P, una sua istanza p, e un algoritmo A che risolve P, indichiamo con costo (o complessità) di A applicato a p una misura delle risorse utilizzate dalle computazioni che A esegue su una macchina MR per determinare la soluzione di p. Le risorse, in principio, sono di due tipi, *memoria occupata e tempo di calcolo*: nell'ipotesi che tutte le operazioni elementari abbiano la stessa durata, il tempo di calcolo può essere espresso come numero di operazioni elementari effettuate dall'algoritmo.

Poiché molto spesso la risorsa più critica è il tempo di calcolo, nel seguito useremo soprattutto questa come misura della complessità degli algoritmi.
Dato un algoritmo, è opportuno disporre di una misura di complessità che consenta una valutazione sintetica della sua bontà ed eventualmente un suo agevole confronto con algoritmi alternativi. Conoscere la complessità di A per ognuna delle istanze di P non è possibile (l'insieme delle istanze di un problema è normalmente infinito), né sarebbe di utilità pratica: si cerca allora di esprimere la complessità come una funzione $g(\,n\,)$ della dimensione, n, dell'istanza cui viene applicato l'algoritmo. Poiché, per ogni dimensione, si hanno in generale molte istanze di quella dimensione, si sceglie $g(n)$ come il costo necessario per risolvere la più difficile tra le istanze di dimensione n: si parla allora di *complessità nel caso peggiore*.

Bisogna naturalmente definire in modo preciso il significato di dimensione di una istanza: chiameremo dimensione di p una misura del numero di bit necessari per rappresentare, con una codifica "ragionevolmente" compatta, i dati che definiscono p, cioè una misura della lunghezza del suo input. Per esempio, in un grafo con n nodi e m archi i nodi possono essere rappresentati dagli interi tra 1 ed n e gli archi per mezzo di una lista contenente m coppie di interi (l'arco che collega i nodi i e j è rappresentato dalla coppia $(\,i\,,\,j\,)$: trascurando le costanti moltiplicative, potremo allora assumere come misura della dimensione della codifica del grafo, al variare del numero dei nodi e degli archi, la funzione $m \log n$, dato che interi positivi e non superiori ad n possono essere rappresentati con $\log n$ bit. Nel seguito, per semplicità, oltre alle costanti moltiplicative trascureremo anche le funzioni sublineari, come la funzione logaritmo; diremo allora che m è la lunghezza dell'input per un grafo con m archi. Nelle ipotesi fatte, la misura della lunghezza dell'input non varia se usiamo una codifica in base $b > 2$: se invece si usasse una codifica unaria, la lunghezza dell'input nell'esempio in questione diventerebbe nm, aumentando considerevolmente.

A questo punto la funzione $g(\,n\,)$, introdotta precedentemente, risulta definita in modo sufficientemente rigoroso: in pratica essa continua però ad essere di difficile uso come misura della complessità, dato che risulta difficile, se non praticamente impossibile, la valutazione di $g(\,n\,)$ per ogni dato valore di n. Questo problema si risolve sostituendo alla $g(\,n\,)$ il suo ordine di grandezza: si parlerà allora di *complessità asintotica*. Data una funzione $g(\,x\,)$, diremo che:

1. $g(\,x\,)$ è $O(\,f(\,x\,)\,)$ se esistono due costanti c_1 e c_2 per cui, per ogni x, si ha $g(\,x\,) \leq c_1 f(\,x\,) + c_2$;
2. $g(\,x\,)$ è $\Omega(\,f(\,x\,)\,)$ se $f(\,x\,)$ è $O(\,g(\,x\,)\,)$;
3. $g(\,x\,)$ è $\Theta(\,f(\,x\,)\,)$ se $g(\,x\,)$ è allo stesso tempo $O(\,f(\,x\,)\,)$ e $\Omega(\,f(\,x\,)\,)$.

Sia $g(\,x\,)$ il numero di operazioni elementari che vengono effettuate dall'algoritmo A applicato alla più difficile istanza, tra tutte quelle che hanno lunghezza di input x, di un dato problema P: diremo che la complessità di A è un $O(\,f(\,x\,)\,)$ se $g(\,x\,)$ è un $O(\,f(\,x\,)\,)$; analogamente, diremo che la complessità di A è un $\Omega(\,f(\,x\,)\,)$ o un $\Theta(\,f(\,x\,)\,)$ se $g(\,x\,)$ è un $\Omega(\,f(\,x\,)\,)$ o un $\Theta(\,f(\,x\,)\,)$.

A.3 Problemi trattabili e problemi intrattabili

Chiameremo *trattabili* i problemi per cui esistono algoritmi la cui complessità sia un $O(\,p(\,x\,)\,)$, con $p(\,x\,)$ un polinomio in x, e *intrattabili* i problemi per cui un tale algoritmo non esiste: le seguenti tabelle chiariscono il perché di questa distinzione. In questa tabella vengono forniti, per diverse funzioni di complessità f, i tempi di esecuzione (in secondi, ove non diversamente specificato) per alcuni valori di n su un calcolatore che richieda $1e^{-6}$ secondi per effettuare un'operazione elementare.

f	10	20	40	60
n	$1e^{-5}$	$2e^{-5}$	$4e^{-5}$	$6e^{-5}$
n^3	$1e^{-3}$	$8e^{-3}$	$7e^{-2}$	$2e^{-1}$
n^5	$1e^{-1}$	3.2	1.7 min.	13 min.
2^n	$1e^{-3}$	1	13 giorni	36600 anni
3^n	$6e^{-2}$	1 ora	$4e^5$ anni	$1e^{13}$ anni

In questa tabella vengono invece indicati i miglioramenti ottenibili, in termini di dimensioni delle istanze risolvibili, per diverse funzioni di complessità, al migliorare della tecnologia dei calcolatori: con x_i abbiamo indicato la dimensione di un'istanza risolvibile oggi in un minuto per la i-esima funzione di complessità.

f	computer odierno	100 volte più veloce	10000 volte più veloce
n	x_1	$100x_1$	$10000x_1$
n^3	x_2	$4.6x_2$	$21.5x_2$
n^5	x_3	$2.5x_3$	$6.3x_3$
2^n	x_4	$x_4 + 6.6$	$x_4 + 13.2$
3^n	x_5	$x_5 + 4.2$	$x_5 + 8.4$

Molti problemi di rilevante importanza pratica sono trattabili: sono problemi per i quali disponiamo di efficienti algoritmi di complessità polinomiale. Per potere effettuare una più rigorosa classificazione dei diversi problemi, facciamo riferimento a problemi in forma decisionale.

A.3.1 Le classi $\mathcal{P}$ e $\mathcal{NP}$

Una prima importante classe di problemi è la classe $\mathcal{NP}$, costituita da tutti i problemi decisionali il cui problema di certificato associato può essere risolto in tempo polinomiale. In altri termini, i problemi in $\mathcal{NP}$ sono quelli per cui è possibile verificare efficientemente una risposta "si", perché è possibile decidere in tempo polinomiale se una soluzione x è ammissibile per il problema. Ad esempio, il problema della soddisfattibilità proposizionale (SAT), introdotto nell'Esempio 1.9, è un problema in $\mathcal{NP}$: dato un qualunque assegnamento di valori di verità, è possibile verificare se tale assegnamento rende vera la formula in tempo lineare nella sua dimensione.

Equivalentemente, si può definire $\mathcal{NP}$ come la classe di tutti i problemi decisionali risolubili in tempo polinomiale da una MR nondeterministica ($\mathcal{NP}$ va infatti inteso come $\mathcal{P}$olinomiale nel calcolo $\mathcal{N}$ondeterministico). Una MR nondeterministica è il modello di calcolo (astratto) in cui una MR, qualora si trovi ad affrontare un'operazione di salto condizionale, può eseguire *contemporaneamente* entrambi rami dell'operazione, e questo ricorsivamente per un qualsiasi numero di operazioni (si veda il §7.2.6. In altre parole, i problemi in $\mathcal{NP}$ sono quelli per cui esiste una computazione di lunghezza polinomiale che può portare a costruire una soluzione ammissibile, se esiste, ma questa computazione può essere "nascosta" entro un insieme esponenziale di computazioni analoghe tra le quali, in generale, non si sa come discriminare.

Ad esempio, per SAT si può immaginare una MR nondeterministica che costruisca l'assegnamento di valori di verità ai letterali con una computazione in cui, sequenzialmente, viene assegnato il valore di verità a ciascun letterale, in un certo ordine prestabilito, in base ad una certa condizione logica. Questo identifica un *albero di computazione* (si veda il §7.1. che descrive tutte le possibili esecuzioni della MR, e le cui foglie sono tutti i 2^n possibili assegnamenti di valori di verità agli n letterali della formula. Se la formula è soddisfattibile, allora esiste un cammino nell'albero di computazione, di lunghezza lineare nel numero di letterali, che porta a costruire esattamente un certificato del problema, ossia un assegnamento di valori di verità che soddisfa la formula.

Un sottoinsieme della classe $\mathcal{NP}$ è la classe $\mathcal{P}$, costituita da tutti i *(problemi polinomiali)*, quei problemi decisionali per i quali esistono algoritmi di complessità polinomiale che li risolvono. Una domanda particolarmente importante è se esistano problemi in $\mathcal{NP}$ che non appartengano anche a $\mathcal{P}$. A questa domanda non si è a tutt'oggi stati in grado di rispondere, ma si ritiene fortemente probabile sia effettivamente $\mathcal{P} \neq \mathcal{NP}$; una breve giustificazione di questo sarà data nel paragrafo successivo.

A.3.2 Problemi $\mathcal{NP}$-completi e problemi $\mathcal{NP}$-ardui

Molti problemi, anche se apparentemente notevolmente diversi, possono tuttavia essere ricondotti l'uno all'altro; dati due problemi decisionali, P e Q, diciamo che P *si riduce in tempo polinomiale a* Q, e scriveremo $P \propto Q$, se, supponendo l'esistenza di un algoritmo A_Q che risolva Q in tempo costante (indipendente dalla lunghezza dell'input), esiste un algoritmo che risolve P in tempo polinomiale

utilizzando come sottoprogramma A_Q: parliamo in tal caso di *riduzione polinomiale di* P a Q. Ad esempio, SAT si riduce polinomialmente alla *PLI*, come mostrato nell'Esempio 1.9. Quindi, se esistesse un algoritmo polinomiale per la *PLI* allora esisterebbe un algoritmo polinomiale per SAT: data la formula in forma normale congiuntiva, basterebbe produrre (in tempo polinomiale) il problema di *PLI* corrispondente, applicare l'algoritmo a tale problema e rispondere "si" o "no" a seconda della risposta ottenuta. Possiamo quindi affermare che $SAT \propto PLI$. È facile verificare che la relazione $\propto$ ha le seguenti proprietà:

1. è riflessiva: $A \propto A$;
2. è transitiva: $A \propto B$ e $B \propto C \Rightarrow A \propto C$;
3. se $A \propto B$ e $A \notin P$, allora $B \notin P$;
4. se $A \propto B$ e $B \in P$, allora $A \in P$.

Possiamo definire adesso la classe dei problemi $\mathcal{NP}$-completi: un problema A è detto *$\mathcal{NP}$-completo* se $A \in \mathcal{NP}$ e se per ogni $B \in \mathcal{NP}$ si ha che $B \propto A$. La classe dei problemi $\mathcal{NP}$-completi costituisce un sottoinsieme di $\mathcal{NP}$ di particolare importanza; un fondamentale teorema, dovuto a Cook, garantisce che ogni problema $P \in \mathcal{NP}$ si riduce polinomialmente a SAT, ossia che tale classe non è vuota. I problemi $\mathcal{NP}$-completi hanno la proprietà che se esiste per uno di essi un algoritmo polinomiale, allora necessariamente tutti i problemi in $\mathcal{NP}$ sono risolubili polinomialmente, e quindi è $\mathcal{P} = \mathcal{NP}$; in un certo senso, tali problemi sono i "più difficili" tra i problemi in $\mathcal{NP}$. Un problema che abbia come caso particolare un problema $\mathcal{NP}$-completo ha la proprietà di essere almeno tanto difficile quanto i problemi $\mathcal{NP}$-completi (a meno di una funzione moltiplicativa polinomiale): un problema di questo tipo si dice $\mathcal{NP}$-arduo. Si noti che un problema $\mathcal{NP}$-arduo può anche non appartenere a $\mathcal{NP}$. Ad esempio, sono $\mathcal{NP}$-ardui i problemi di ottimizzazione la cui versione decisionale è un problema $\mathcal{NP}$-completo: infatti, come si è già visto, è sempre possibile ricondurre un problema decisionale ad un problema di ottimizzazione con una opportuna scelta della funzione obiettivo. Ad esempio, il problema della *PLI* è $\mathcal{NP}$-arduo, poichè ad esso si riduce SAT, che è $\mathcal{NP}$-completo.

Fino ad oggi, sono stati trovati moltissimi problemi $\mathcal{NP}$-ardui; si può affermare che la grande maggioranza dei problemi combinatori siano $\mathcal{NP}$-ardui. Nonostante tutti gli sforzi dei ricercatori, non è stato possibile determinare per nessuno di essi un algoritmo polinomiale (il che avrebbe fornito algoritmi polinomiali per tutti i problemi della classe); questo fa ritenere che non esistano algoritmi polinomiali per i problemi $\mathcal{NP}$-completi, ossia che sia $\mathcal{P} \neq \mathcal{NP}$. Ad oggi non si conosce nessuna dimostrazione formale di questa ipotesi, ma essa è suffragata da molti indizi. Ad esempio, $\mathcal{P}$ ed $\mathcal{NP}$ sono solo i primi elementi di una *gerarchia polinomiale* di classi di problemi, sempre "più difficili", che collasserebbero tutte sulla sola classe $\mathcal{P}$ qualora fosse $\mathcal{P} = \mathcal{NP}$, il che fa ritenere questa eventualità altamente improbabile.

Per riassumere, possiamo dire che, allo stato delle conoscenze attuali, tutti i problemi $\mathcal{NP}$-ardui sono presumibilmente intrattabili. Naturalmente, ciò non significa che non sia in molti casi possibile costruire algoritmi in grado di risolvere efficientemente istanze di problemi $\mathcal{NP}$-ardui di dimensione significativa (quella richiesta dalle applicazioni reali); questo non rientra comunque nel campo della teoria della complessità computazionale ma piuttosto nel campo dell'Ottimizzazione Combinatoria (si veda il Capitolo 4 e seguenti).

A.3.3 Complessità ed approssimazione

Esistono molti altri risultati interessanti della teoria della complessità computazionale che non rientrano nello scopo di queste dispense. Risultati molto importanti permettono, ad esempio, di caratterizzare classi di problemi per cui non solo il problema originario, ma anche ottenere una soluzione approssimata sia un problema intrattabile.

Ricordiamo che un algoritmo euristico si dice *ε-approssimato* se produce una soluzione ε-ottima per ogni istanza. Uno *schema di approssimazione* è un algoritmo che, oltre all'istanza p di P, prende in input anche un parametro $\varepsilon > 0$ e produce una soluzione $\bar{x}$ che è ε-ottima; sostanzialmente uno schema di approssimazione è famiglia infinita di algoritmi ε-approssimati, uno per ogni valore di ε.

Tipicamente, la complessità di un tale algoritmo sarà anche una funzione dell'approssimazione ε voluta (crescente con $1 / \varepsilon$): uno schema di approssimazione è detto:

- *polinomiale*, se la sua complessità è polinomiale nella dimensione dell'istanza p (ma può essere esponenziale in $1 / \varepsilon$);
- *pienamente polinomiale*, se la sua complessità è polinomiale nella dimensione di p ed in $1 / \varepsilon$.

Accenniamo brevemente qui ad un noto risultato che riguarda la difficoltà di costruire schemi di approssimazione pienamente polinomiali per un problema di ottimizzazione.

Esistono problemi $\mathcal{NP}$-completi che sono risolvibili polinomialmente nella lunghezza dell'input se codificati in unario, mentre non lo sono con codifiche più compatte: un esempio è il problema dello zaino (Esempio 1.5). Tali problemi sono in un certo senso più facili di quelli che richiedono algoritmi esponenziali indipendentemente dalla codifica dell'input. Viene detto *pseudopolinomiale* un algoritmo che risolva uno di tali problemi in tempo polinomiale nella lunghezza dell'input codificato in unario; un problema è quindi detto *pseudopolinomiale* se può essere risolto per mezzo di un algoritmo pseudopolinomiale. Viene chiamato $\mathcal{NP}$-completo *in senso forte* un problema $\mathcal{NP}$-completo per cui non esistono algoritmi pseudopolinomiali; analogamente, viene chiamato $\mathcal{NP}$-arduo *in senso forte* un problema di ottimizzazione che abbia come versione decisionale un problema $\mathcal{NP}$-completo in senso forte. I problemi $\mathcal{NP}$-completi in senso forte sono tipicamente quelli che "non contengono numeri": ad esempio, SAT è un problema $\mathcal{NP}$-completo in senso forte. Un diverso esempio è il problema del commesso viaggiatore (TSP) (si veda l'Esempio 1.7), che resta $\mathcal{NP}$-arduo anche se i costi degli archi sono limitati a due soli possibili valori, ad esempio 0 e 1. Infatti, dato un algoritmo per TSP con costi in $\{ 0 , 1 \}$ è possibile risolvere (in modo ovvio) il problema del ciclo Hamiltoniano su un grafo, che è notoriamente $\mathcal{NP}$-completo.

È possibile dimostrare che l'esistenza di algoritmi pseudopolinomiali per un problema è equivalente all'esistenza di *schemi di approssimazione pienamente polinomiali* per il problema; in altri termini, nessun problema $\mathcal{NP}$-arduo in senso forte (la grande maggioranza) ammette schemi di approssimazione pienamente polinomiali (a meno che $\mathcal{P} = \mathcal{NP}$). Di conseguenza, per la maggior parte dei problemi $\mathcal{NP}$-ardui è difficile non solo risolvere il problema originario, ma anche una sua approssimazione arbitraria.

Risultati di questo tipo sono stati dimostrati anche per approssimazioni con errore relativo fissato. Ad esempio, è stata definita un'operazione di riduzione simile a $\propto$ che preserva l'approssimabilità, ossia tale che se esiste un algoritmo ε-approssimato per un un certo problema A ed un altro problema B si riduce ad A, allora esiste un algoritmo ε-approssimato (possibilmente per un diverso ε) anche per B. Sfruttando alcuni risultati che mostrano come per certi problemi non possano esistere algoritmi ε-approssimati con ε più piccolo di una specifica soglia (a meno che $\mathcal{P} = \mathcal{NP}$), si dimostra che per una grande classe di problemi di ottimizzazione non possono esistere algoritmi di approssimazione polinomiali con precisione arbitrariamente piccola.

Riferimenti Bibliografici

M.R. Garey, D.S. Johnson "Computers and Intractability; A Guide to the Theory of $\mathcal{NP}$-Completeness" *W.H. Freeman & Co.*, 1990

Appendice B

Grafi e Reti

In questa appendice richiamiamo i principali concetti relativi a grafi e reti; descriviamo inoltre alcune classi di strutture dati che possono essere utilizzate per implementare efficientemente algoritmi su grafi e reti.

B.1 I grafi: notazione e nomenclatura

Un grafo *orientato* (o *diretto*) $G = (N, A)$, è una coppia di insiemi: N è un insieme finito e non vuoto di elementi, mentre A è un insieme finito di coppie di elementi distinti di N.

B.1.1 Nodi, archi

N è detto *insieme dei nodi*, e usualmente viene indicato mediante i primi $n = |N|$ numeri naturali: $N = \{1, 2, \ldots, n\}$. L'insieme A, detto *insieme degli archi*, è in generale indicato con $A = \{a_1, a_2, \ldots, a_m\}$ $(m = |A|)$. Nel seguito un arco verrà indifferentemente denotato o da un nome che lo individua, ad esempio a_k, oppure da una coppia di nodi, e quindi come (i, j). La coppie in A sono *ordinate*, ossia gli archi sono *orientati*: i nodi i e j sono detti *estremi* dell'arco (i, j), che è *incidente* in essi; in questo caso si dirà che i nodi i e j sono *adiacenti*. Dato l'arco (i, j), i è la sua *coda* e j è la sua *testa*; l'arco è *uscente* da i ed *entrante* in j.

Il fatto che i e j abbiano ruoli distinti in (i, j), ossia che vi sia un orientamento nell'arco, è spesso utile ma non sempre necessario. In molti casi tutto ciò che interessa è che due nodi i e j sono "collegati". Ciò porta alla definizione di *lato*, ossia arco *non orientato*, come una coppia non ordinata $\{i, j\}$ di nodi. Valgono per questo caso molte delle definizioni precedenti, quale quella di estremi, adiacenza, ecc.. Un grafo i cui archi sono tutti non orientati è detto *non orientato* o *simmetrico*. Di solito si indica come (V, E), dove V è detto insieme dei *vertici* ed E insieme dei *lati*. In un certo senso i grafi orientati sono una generalizzazione dei grafi non orientati: infatti un arco non orientato può essere rappresentato per mezzo di una coppia di archi orientati, come indicato in Figura B.1. In Figura B.2 sono rappresentati un grafo non orientato (a) ed un grafo orientato (b). Nel seguito pertanto i principali concetti verranno dati solo in termini di grafi orientati in quanto la loro estensione ai grafi non orientati è in genere immediata.

Figura B.1: Equivalenza tra un lato ed una coppia di archi

Esercizio B.1. Indicare i nodi adiacenti al nodo 5 e gli archi incidenti nel nodo 1 del grafo in figura B.2(a).

Esercizio B.2. Indicare i nodi adiacenti al nodo 5 e gli archi incidenti nel nodo 1 del grafo in figura B.2(b).

Una *rete* è un grafo ai cui nodi e/o archi sono associati dei pesi. Tali pesi possono avere significati diversi a seconda del contesto; ad esempio, se il grafo rappresenta una rete idraulica, i pesi associati

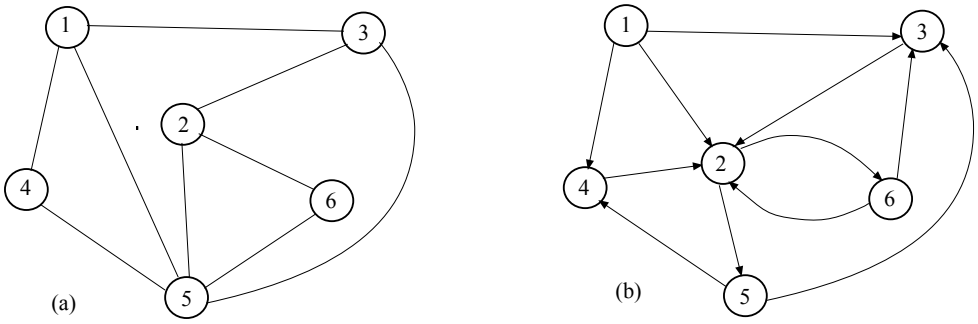


Figura B.2: Esempi di grafi

agli archi possono rappresentare la portata (numero di litri per unità di tempo) e il flusso (effettiva quantità di acqua che fluisce nell'arco), mentre i pesi associati ai nodi possono rappresentare la quantità di acqua immessa nella rete o da essa estratta. I grafi rappresentano la struttura topologica delle reti; nel seguito faremo uso indifferentemente dei termini "grafo" e "rete".

Dato un grafo (orientato o non orientato), per ogni nodo $i \in N$, si indica con $N(\,i\,)$ l'*insieme dei nodi (vertici) adiacenti* ad i e con $S(\,i\,)$ l'*insieme degli archi (lati) incidenti* in i; $|\,S(\,i\,)\,|$ è detto il *grado* del nodo i ed è indicato con g_i. Dato un grafo orientato, per ogni nodo $i \in N$, si indica con $FN(\,i\,)$ e $BN(\,i\,)$ rispettivamente l'*insieme dei nodi successori* e l'*insieme dei nodi predecessori*

$$FN(\,i\,) = \{\, j \in N \,:\, \exists (\,i\,,\,j\,) \in A \,\} \qquad , \qquad BN(\,i\,) = \{\, j \in N \,:\, \exists (\,j\,,\,i\,) \in A \,\} \ ,$$

mentre con $FS(\,i\,)$ e $BS(\,i\,)$ si indicano rispettivamente l'*insiemi degli archi uscenti* da i, o *stella uscente* di i, e l'*insieme degli archi entranti* in i, o *stella entrante* di i:

$$FS(\,i\,) = \{\, (\,x\,,\,y\,) \in A \,:\, x = i \,\} \qquad , \qquad BS(\,i\,) = \{\, (\,x\,,\,y\,) \in A \,:\, y = i \,\} \ .$$

Valgono le relazioni $FN(\,i\,) \cup BN(\,i\,) = N(\,i\,)$ e $FS(\,i\,) \cup BS(\,i\,) = S(\,i\,)$; $|\,FN(\,i\,)\,| = |\,FS(\,i\,)\,|$ e $|\,BN(\,i\,)\,| = |\,BS(\,i\,)\,|$ sono detti rispettivamente *grado uscente* e *grado entrante* di i.

Esercizio B.3. Indicare la stella uscente e la stella entrante del nodo 2 del grafo in Figura B.2(b).

Dato un grafo $G = (\,N\,,\,A\,)$, il grafo $G' = (\,N\,,\,A'\,)$, con $A' \subset A$, è detto *grafo parziale* di G; il grafo $G'' = (\,N''\,,\,A''\,)$, con $N'' \subset N$ e $A'' = \{\, (\,i\,,\,j\,) \in A \,:\, i \in N'' \ ,\ j \in N'' \,\}$, è detto *grafo indotto* da N''. Un grafo $G^* = (\,N^*\,,\,A^*\,)$, è un *sottografo* di G se A^* è contenuto in A e N^* contiene tutti i nodi estremi degli archi di A^*.

Esercizio B.4. Disegnare il grafo parziale $G' = (\,N\,,\,A'\,)$ del grafo $G = (\,N\,,\,A\,)$ in Figura B.2(b), dove $A' = \{\, (\,1\,,\,2\,)\,,\,(\,2\,,\,5\,)\,,\,(\,2\,,\,6\,)\,,\,(\,4\,,\,2\,)\,,\,(\,5\,,\,3\,)\,,\,(\,7\,,\,3\,),\}$.

Esercizio B.5. Disegnare il grafo $G'' = (\,N''\,,\,A''\,)$ indotto da $N'' = \{\,2\,,\,4\,,\,5\,,\,6\,\}$ sul grafo $G = (\,N\,,\,A\,)$ in Figura B.2(b).

Esercizio B.6. Disegnare il sottografo $G^* = (\,N^*\,,\,A^*\,)$ del grafo $G = (\,N\,,\,A\,)$ in Figura B.2(b), dove $A^* = \{\, (\,1\,,\,3\,)\,,\,(\,1\,,\,2\,)\,,\,(\,1\,,\,4\,)\,\}$.

B.1.2 Cammini, cicli

Dato un grafo orientato G, un *cammino* tra il nodo i_0 ed il nodo i_q è una sequenza di nodi e di archi

$$C = \{\, i_0 \ ,\ a_1 \ ,\ i_1 \ ,\ a_2 \ ,\ i_2 \ ,\ \ldots \ ,\ i_{q-1} \ ,\ a_q \ ,\ i_q \,\}$$

in cui per ogni $h = 1, \ldots, q$, a_h è incidente in i_{h-1} e i_h. Se, per $h = 1, \ldots, q$, è $a_h = (\,i_{h-1}\,,\,i_h\,)$, allora C è un *cammino orientato* da i_0 a i_q. Il nodo i_0 è detto l'*origine* e i_q è detto la *destinazione* di C. Una sottosequenza di C è detta *sottocammino*. Se $i_0 = i_q$, C è detto un *ciclo* (eventualmente orientato). Un cammino (ciclo) non contenente cicli come sottocammini (propri) è detto un *cammino* (*ciclo*) *semplice* (eventualmente orientato). Nei cammini (cicli) che non sono semplici vi è ripetizione di nodi; essi possono avere anche ripetizioni di archi (si veda la Figura B.3). Quando non si ingenerano equivoci, un cammino può essere descritto mediante la sola sequenze dei suoi nodi o dei suoi archi.

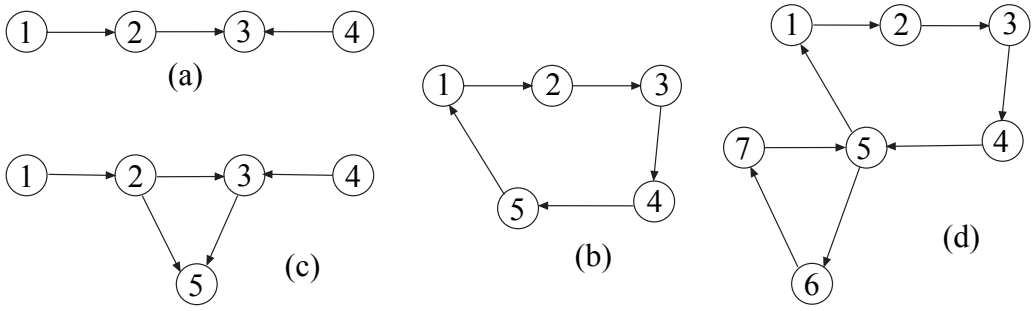


Figura B.3: (a) un cammino semplice; (b) un ciclo orientato semplice; (c) un cammino non semplice con ripetizione di archi; (d) un ciclo orientato non semplice senza ripetizione di archi

Esercizio B.7. Individuare un cammino tra i nodi 2 e 4 sul grafo $G = (N, A)$ in Figura B.2(a); dire se è semplice.

Esercizio B.8. Individuare un ciclo orientato semplice formato da tre nodi e tre archi sul grafo $G = (N, A)$ in Figura B.2(b).

Un ciclo semplice formato da n archi è detto *ciclo Hamiltoniano*; esso passa per ogni nodo del grafo una e una sola volta. Un ciclo senza ripetizione di archi formato da m archi è detto emphciclo Euleriano; esso passa attraverso ciascun arco del grafo una e una sola volta. Si può dimostrare che un grafo possiede un ciclo Euleriano se e solo se il grado di ogni nodo è pari, e che un grafo orientato possiede un ciclo Euleriano orientato se e solo se le cardinalità della stella uscente e della stella entrante di ogni nodo sono uguali.

Esercizio B.9. Dimostrare l'affermazione precedente.

Esercizio B.10. Individuare, se esiste, un ciclo Hamiltoniano sul grafo $G = (N, A)$ in Figura B.2(a).

Due nodi i e j sono *connessi* se esiste un cammino tra di essi; in un grafo orientato si dice che j è *connesso a* i se esiste un cammino orientato da i a j.

Esercizio B.11. Elencare tutti i nodi che sono connessi al nodo 2 sul grafo $G = (N, A)$ in Figura B.2(b).

B.1.3 Tagli e connettività

Dato un grafo $G = (N, A)$, una partizione di N in due sottoinsiemi non vuoti N' e N'' è detta un *taglio* del grafo e viene indicata con (N', N''). I due insiemi N' ed N'' sono detti le *rive* del taglio; l'insieme degli archi aventi un estremo in N' e l'altro in N''

$$A(N', N'') = \{(i, j) \in A : i \in N', j \in N'' \text{ oppure } j \in N', i \in N''\}$$

è detto *insieme degli archi del taglio.*

Esercizio B.12. Sia dato il taglio $(\{1, 3, 5\}, \{2, 4, 6\})$ del grafo $G = (N, A)$ in Digura B.2(a), fornire l'insieme degli archi del taglio.

Se il grafo è orientato, per ogni taglio si possono distinguere due insiemi di archi del taglio, l'insieme degli *archi diretti* del taglio

$$A^+(N', N'') = \{(i, j) \in A : i \in N', j \in N''\}$$

e l'insieme degli *archi inversi* del taglio:

$$A^-(N', N'') = \{(i, j) \in A : j \in N', i \in N''\} \ .$$

Ovviamente, $A(N', N'') = A^+(N', N'') \cup A^-(N', N'')$.

Esercizio B.13. Sia dato il taglio $(N', N'') = (\{1, 3, 5, 7\}, \{2, 4, 6\})$ del grafo $G = (N, A)$ in Figura B.2(b), fornire gli insiemi $A^+(N', N'')$ e $A^-(N', N'')$.

Utilizzando il concetto di taglio possiamo ridefinire la relazione di connessione fra nodi. Due nodi i e j sono connessi se non esiste un taglio (N', N'') tale che sia $i \in N'$, $j \in N''$ e $A(N', N'') = \emptyset$.

Esercizio B.14. Dimostrare l'equivalenza delle due definizioni di connessione, basate sui cammini e sui tagli.

Analogamente, in un grafo orientato j è connesso a i se non esiste un taglio (N', N'') tale che sia $i \in N'$, $j \in N''$ e $A^+(N', N'') = \emptyset$. Un *grafo connesso* è un grafo in cui tutte le coppie di nodi sono connesse, altrimenti è detto *non connesso*. Un *grafo fortemente connesso* è un grafo in cui per ogni coppia di nodi i, j si ha che j è connesso a i, cioè esiste un cammino orientato da i a j. Ad esempio tutti i grafi in Figura B.3 sono connessi, mentre solamente i grafi (b) e (d) sono fortemente connessi. Dato un grafo non connesso $G = (N, A)$, ogni grafo connesso indotto da un sottoinsieme di N, massimale rispetto alla connessione, è detto *componente connessa* di G. Ad esempio il grafo di Figura B.4 contiene due componenti connesse $G' = (N', A')$ e $G'' = (N'', A'')$ che sono i sottografi indotti da $N' = \{1, 2, 3\}$ e $N'' = \{4, 5, 6, 7\}$.

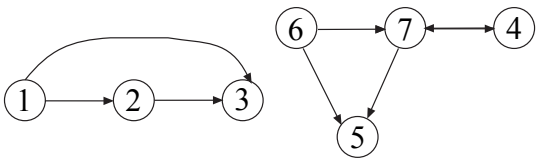


Figura B.4: Due componenti connesse

Un grafo è detto *completo* se esiste un arco tra ogni coppia di nodi. Pertanto un grafo non orientato completo possiede $m = n(n-1)/2$ archi, mentre un grafo orientato completo possiede $m = n(n-1)$ archi.

Esercizio B.15. Disegnare un grafo completo non orientato con 4 nodi e un grafo completo orientato con tre nodi.

Un grafo $G = (N, A)$ è detto *bipartito* se esiste un taglio (N', N'') il cui insieme degli archi $A(N', N'')$ coincide con A, cioè se è possibile partizionare N in N' e N'' in modo tale che tutti gli archi abbiano un estremo in N' e l'altro in N''. Un grafo (orientato) bipartito è detto completo se esiste un arco (orientato) per ogni coppia di nodi non appartenenti al medesimo sottoinsieme del taglio; indicando con $n' = |N'|$ e $n'' = |N''|$ le cardinalità dei due sottoinsiemi, il numero di archi è $m = n'n''$ se il grafo è non orientato, mentre è $m = 2n'n''$ nel caso sia orientato.

Esercizio B.16. Dimostrare le affermazioni fatte sul numero di archi dei grafi completi e dei grafi bipartiti completi (orientati e non).

Esercizio B.17. Disegnare un grafo bipartito completo non orientato con 8 nodi in cui $n' = 3$ e $n'' = 5$.

B.1.4 Alberi

Un grafo connesso e privo di cicli è detto *albero*. Un albero $T = (N, A)$, con $|N| = n$, è tale che $|A| = n - 1$. Sono equivalenti alla precedente le seguenti definizioni: un albero è un grafo connesso con $n - 1$ archi; un albero è un grafo privo di cicli con $n - 1$ archi.

Esercizio B.18. Dimostrare l'equivalenza delle definizioni date.

Un *albero radicato* è un albero in cui sia stato selezionato un nodo, detto *radice dell'albero*; in un tale albero i nodi possono essere ordinati per "livelli" in modo ricorsivo: la radice è posta al livello 0 e i nodi adiacenti ad essa sono posti al livello 1; al livello $k + 1$ appartengono i nodi che non appartengono al livello $k - 1$ e che sono adiacenti ai nodi del livello k. Nel grafo di Figura B.5, se si assume il nodo 1 come radice, al livello 0 appartiene il nodo 1, al livello 1 si trovano i nodi 2 e 3, a livello 2 si trovano i nodi 4, 5, 6 e 7, mentre a livello 3 si trovano i nodi 8 e 9.

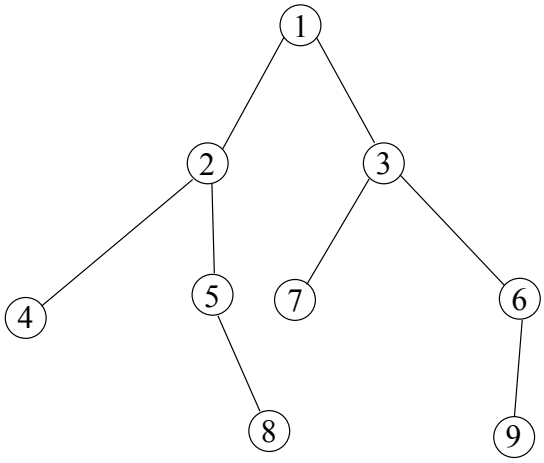


Figura B.5: Un albero

Esercizio B.19. Disegnare l'albero in Figura B.5 radicato nel nodo 3; individuare i nodi a livello 2.

Ogni arco di un albero radicato connette due nodi appartenenti a livelli adiacenti. Per ciascun arco (i, j) con i a

livello k e j a livello $k+1$, i è detto *padre* di j e questi *figlio* di i; la radice non ha padre. Nodi aventi lo stesso padre sono detti *fratelli*. Un nodo senza figli è detto *foglia* dell'albero radicato.

Esiste un solo cammino tra la radice e qualsiasi nodo dell'albero; la lunghezza (in numero di archi) di tale cammino è uguale al livello cui appartiene il nodo destinazione del cammino. Un nodo i che appartiene al cammino dalla radice ad un nodo j è detto un *antenato* di j e questi un *discendente* di i (ogni nodo è antenato e discendente di sé stesso). Un sottoalbero $T(\,j\,)$ di un albero radicato è il grafo indotto dall'insieme dei nodi discendenti di j; in altri termini $T(\,j\,)$ è formato da j, da tutti gli altri suoi discendenti e da tutti gli archi dell'albero tra questi nodi. Ad esempio, nell'albero di Figura B.5, considerato radicato in 1, il padre di 3 è 1; i nodi 4 e 5 sono fratelli; l'insieme delle foglie è $\{\,4\,,\,7\,,\,8\,,\,9\,\}$; l'insieme degli antenati di 3 è $\{\,1\,,\,3\,\}$ e quello dei discendenti è $\{\,3\,,\,6\,,\,7\,,\,9\,\}$. Inoltre, il sottoalbero $T(\,2\,)$, disegnato in Figura B.6, dell'albero di Figura B.5, radicato in 1, contiene i nodi 2, 4, 5 e 8.

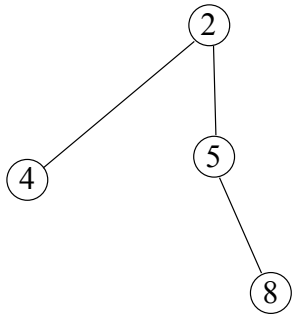


Figura B.6: Un sottoalbero

Esercizio B.20. Individuare sull'albero disegnato per l'esercizio precedente i figli della radice, gli antenati di 6 e i discendenti di 1; disegnare il sottoalbero $T(\,6\,)$.

Un albero radicato, i cui archi sono orientati, è detto *orientato* se tutti i suoi archi sono orientati dal padre verso il figlio (o dal figlio verso il padre). Dato un grafo $G = (\,N\,,\,A\,)$, un suo grafo parziale $T = (\,N\,,\,A_T\,)$ che sia un albero è detto *albero di copertura* (*spanning tree*) di G; nel grafo di Figura B.7 è evidenziato un albero di copertura. Ogni arco $(\,i\,,\,j\,) \in A$, non appartenente a T, forma con gli archi di T un unico ciclo che indicheremo con $C_T(\,i\,,\,j\,)$. Inoltre, l'eliminazione di un arco $(\,i\,,\,j\,) \in A_T$ divide l'albero T in due sottoalberi $T_i = (\,N_i\,,\,A_i\,)$ e $T_j = (\,N_j\,,\,A_j\,)$, individuando un taglio $(\,N_i\,,\,N_j\,)$. Gli archi del taglio sono quelli dell'insieme

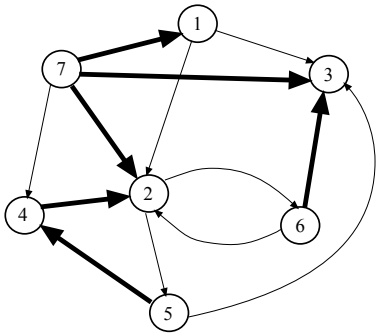


Figura B.7: Un albero di copertura

$$A(\,N_i\,,\,N_j\,) = \{\,(\,k\,,\,l\,) \in A\,:\,k \in N_i\,,\,l \in N_j \text{ oppure } l \in N_i\,,\,k \in N_j\,\}\;;$$

cioè, essi sono $(\,i\,,\,j\,)$ stesso e tutti gli archi non appartenenti a T che, quando aggiunti all'albero, formano un ciclo contenente $(\,i\,,\,j\,)$. Un grafo le cui componenti connesse sono alberi è detto *foresta*.

Esercizio B.21. Disegnare una foresta del grafo in Figura B.4.

B.2 Rappresentazione di grafi ed alberi

In questo paragrafo verranno presentate le strutture dei dati fondamentali per la rappresentazione dei grafi e delle reti, e verranno introdotti alcuni algoritmi elementari che serviranno come strumenti di base per la costruzione di algoritmi per problemi di ottimizzazione su reti.

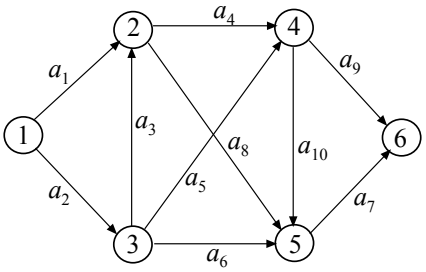
B.2.1 Matrici di incidenza e liste di adiacenza

Dato un grafo orientato $G = (\,N\,,\,A\,)$, la sua *matrice di incidenza* $E = [\,e_{ik}\,]$ è una matrice $n \times m$ (le righe corrispondono ai nodi e le colonne agli archi), così definita:

$$e_{ik} = \begin{cases} -1 & \text{se } i \text{ è la coda dell'arco } a_k \\ 1 & \text{se } i \text{ è la testa dell'arco } a_k \\ 0 & \text{altrimenti} \end{cases} .$$

La matrice di incidenza ha due soli elementi diversi da 0 per ogni colonna: un -1 ed un 1. Un esempio di matrice di incidenza è riportato in Figura B.8. La *lista di adiacenza* per stelle uscenti di un grafo orientato è la sequenza $\{\,FS(\,1\,)\,,\,FS(\,2\,)\,,\,\ldots\,,\,FS(\,n\,)\,\}$ degli insiemi degli archi uscenti dai nodi del grafo. Nell'esempio in Figura B.8, la lista di adiacenza è $\{\,\{\,a_1\,,\,a_2\,\}\,,\,\{\,a_4\,,\,a_8\,\}\,,\,\{\,a_3\,,\,a_5\,,\,a_6\,\}\,,\,\{\,a_9\,,\,a_{10}\,\}\,,\,\{\,a_7\,\}\,,\,\emptyset\,\}$. Analoga è la lista di adiacenza per stelle entranti $\{\,BS(\,1\,)\,,\,BS(\,2\,)\,,\,\ldots\,,\,BS(\,n\,)\,\}$.

Le liste di adiacenza consentono una efficiente memorizzazione dei grafi. Ad esempio in figura B.9 è riportata una possibile rappresentazione del grafo di figura B.8 che utilizza un sistema di liste. In



	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9	a_{10}
1	-1	-1	0	0	0	0	0	0	0	0
2	1	0	1	-1	0	0	0	-1	0	0
3	0	1	-1	0	-1	-1	0	0	0	0
4	0	0	0	1	1	0	0	0	-1	-1
5	0	0	0	0	0	1	-1	1	0	1
6	0	0	0	0	0	0	1	0	1	0

Figura B.8: Un grafo e la sua matrice di incidenza

particolare:

1. I nodi sono rappresentati per mezzo di una lista in cui ogni record corrisponde ad un nodo e contiene quattro campi: il nome del nodo, il puntatore al nodo successivo nella lista, il puntatore al primo arco della stella uscente, il puntatore al primo arco della stella entrante; un puntatore al primo nodo consente di iniziare la scansione.
2. Gli archi sono rappresentati per mezzo di una lista in cui ogni record corrisponde ad un arco e contiene sei campi: il nome dell'arco, il puntatore all'arco successivo, la coda, la testa, il puntatore all'arco successivo della stella uscente cui esso appartiene, il puntatore all'arco successivo della stella entrante cui esso appartiene; anche qui c'è un puntatore al primo arco.

Tale struttura dati permette agevolmente sia la scansione dei nodi e degli archi, sia la scansione degli archi di una stessa stella uscente od entrante quando si conosca il nodo relativo. Essa permette anche inserimenti e rimozioni di nodi e/o di archi. La struttura dati può essere ulteriormente ampliata: ad esempio, a ciascun record di nodo e/o di arco possono essere aggiunti nuovi campi contenenti i pesi o le variabili di interesse. Un altro esempio di ampliamento della struttura consiste nell'introduzione di puntatori inversi dei puntatori sopra definiti; la presenza di tali puntatori consente la rimozione di un nodo o di un arco senza dover scorrere la lista alla ricerca del record precedente.

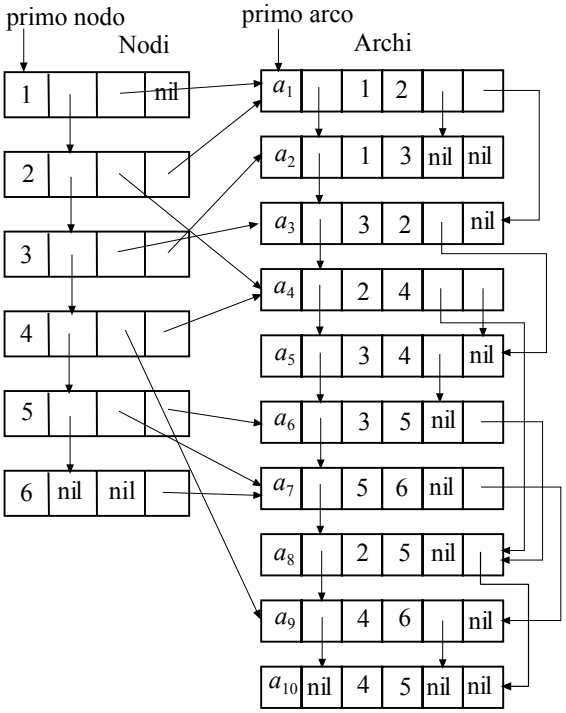


Figura B.9: Liste di adiacenza

Esercizio B.22. Scrivere una procedura che, utilizzando la struttura per liste di adiacenza descritta sopra, calcoli il grado entrante ed il grado uscente di ciascun nodo; se ne valuti la complessità.

Esercizio B.23. Scrivere una procedura che, utilizzando la struttura per liste di adiacenza descritta sopra, inserisca un nuovo nodo $n + 1$; se ne valuti la complessità.

Esercizio B.24. Scrivere una procedura che, utilizzando la struttura per liste di adiacenza descritta sopra, inserisca un nuovo arco a_{m+1}; se ne valuti la complessità.

Esercizio B.25. Scrivere una procedura che, utilizzando la struttura per liste di adiacenza descritta sopra, rimuova un nodo i e tutti gli archi incidenti in esso; se ne valuti la complessità.

Esercizio B.26. Risolvere il problema dell'esercizio precedente utilizzando una struttura ampliata con i puntatori inversi.

Spesso, nella risoluzione di problemi su grafi, le rimozioni di nodi ed archi sono temporanee in quanto un particolare sottoproblema da risolvere è relativo ad una data porzione del grafo originario e sottoproblemi successivi sono relativi ad altre porzioni; non si vuole pertanto distruggere la struttura di dati che descrive il grafo originario. In tal caso, la rimozione fisica dei record relativi a nodi ed

archi risulta inutilmente costosa. È più conveniente invece affiancare, in ciascun record, ai puntatori "statici" che definiscono il grafo originario, nuovi puntatori "dinamici" che definiscono il sottografo corrente, ed operare gli opportuni aggiornamenti su di essi.

Liste di adiacenza mediante vettori di puntatori

La struttura per liste di adiacenza può essere semplificata quando non si prevedono aggiunte o rimozioni di nodi e/o archi. In tal caso le liste a puntatori dei nodi e degli archi possono essere agevolmente realizzate mediante vettori facendo corrispondere l'indice del nodo o dell'arco con l'indice della componente del vettore contenente le informazioni relative al nodo o all'arco. Per realizzare la lista di adiacenza per stelle uscenti è sufficiente disporre di un vettore $P\_FS[\cdot]$ ad $n+1$ componenti, una per ogni nodo più una ausiliaria, e di un vettore $H\_Arc[\cdot]$ ad m componenti, una per ogni arco. L'elemento i-esimo del primo vettore, $i = 1, \ldots, n$, contiene il puntatore al primo arco della stella uscente del nodo i, mentre l'elemento $n+1$ punta all'arco fittizio $m+1$. L'elemento k-esimo del secondo vettore, $k = 1, \ldots, m$, contiene il nodo testa dell'arco k; gli archi sono ordinati per stelle uscenti. Per conoscere la stella uscente del nodo i basta effettuare una scansione del vettore $H\_Arc[\cdot]$ tra la posizione $P\_FS[i]$ e la posizione $P\_FS[\,i+1\,]-1$, ottenendo le teste degli archi aventi i come coda. La stella uscente è vuota se $P\_FS[\,i\,] = P\_FS[\,i+1\,]$. L'occupazione di memoria di questa rappresentazione della lista di adiacenza è $m+n+1$. Ad esempio per rappresentare il grafo di Figura B.8 i due vettori $P\_FS[\cdot]$ e $H\_Arc[\cdot]$ assumono i seguenti valori:

i	1	2	3	4	5	6	7
$P\_FS[\,i\,]$	1	3	5	8	10	11	11

k	1	2	3	4	5	6	7	8	9	10
$H\_Arc[\,k\,]$	2	3	4	5	2	4	5	6	5	6

Analogamente si può realizzare la lista di adiacenza per stelle entranti.

Esercizio B.27. Costruire la lista di adiacenza (per stelle entranti) del grafo in figura B.8.

B.2.2 Rappresentazione di alberi: la funzione predecessore

Un albero radicato di radice r può essere convenientemente rappresentato mediante la funzione predecessore p:

$$p_j = \begin{cases} i & \text{se } i \text{ è padre di } j \\ r & \text{se } j \text{ è la radice } (j = r) \end{cases}$$

La funzione predecessore consente di rappresentare in modo compatto (usando solamente n locazioni di memoria) tutti gli $n-1$ cammini, ciascuno di lunghezza fino ad $n-1$ archi, tra ciascun nodo dell'albero e la radice. Per ricostruire il cammino corrispondente al nodo i è infatti sufficiente risalire da i al suo predecessore p_i, da questo al suo predecessore, e così via finché non si raggiunge il nodo radice r (il cui predecessore è se stesso, indicando che non è più necessario alcun altro passo). Se gli archi dell'albero radicato sono orientati, e ci interessa memorizzare l'orientamento dell'arco che connette un nodo con il padre, basta porre $p_j = -i$ se i è padre di j e l'arco tra essi è $(\,j\,,\,i\,)$. Osserviamo che la funzione predecessore può essere convenientemente inserita nella struttura dati descritta nel §B.2.1, inserendo ulteriori campi in ciascun record della lista corrispondente all'insieme dei nodi. Infatti, per il record corrispondente al nodo i è sufficiente aggiungere un campo contenente il puntatore al nodo p_i, un campo (booleano) per l'orientamento dell'arco $(\,i\,,\,p_i\,)$ o $(\,p_i\,,\,i\,)$ ed eventualmente un campo contenente il puntatore a tale arco nell'insieme degli archi.

B.2.3 Visite di un albero

Un'operazione particolarmente rilevante è la visita di un albero. A seconda dell'ordine con cui i nodi (e gli archi) vengono visitati avremo diversi tipi di visita. Si dice *visita anticipata* di un albero $T = (\,N_T\,,\,A_T\,)$, di radice r e definito dalla funzione predecessore p, una visita dei nodi secondo la regola "un nodo i viene visitato solo se tutti i nodi appartenenti all'unico cammino in T tra r e i sono stati visitati", cioè un nodo può essere visitato solo dopo che sono stati visitati tutti i suoi antenati. Pertanto la visita inizia dalla radice dell'albero e termina in una sua foglia. Osserviamo che la visita anticipata visita anche gli archi di T. Infatti, quando viene visitato un nodo $i \neq r$, viene anche implicitamente visitato l'arco $(\,i\,,\,-p_i\,)$ (o $(\,p_i\,,\,i\,)$); quindi la visita anticipata induce un ordinamento

sui nodi e sugli archi di T. Una visita anticipata è definita per mezzo di una funzione, $va(\cdot)$, che associa ad ogni nodo i il nodo che verrà visitato dopo i attraverso una visita anticipata di T; inoltre, $va(\cdot)$ associa all'ultimo nodo visitato il primo della visita. Dato un albero ci sono diverse funzioni $va(\cdot)$ che realizzano una visita anticipata. In Figura B.10 viene fornito un esempio di visita anticipata; essa permette di visitare consecutivamente i nodi di ciascun sottoalbero.

Definiamo *visita posticipata* di T una visita dei nodi secondo la seguente regola: "un nodo i viene visitato solo se tutti i suoi nodi figli sono stati visitati", cioè un nodo può essere visitato solo dopo che sono stati visitati tutti gli altri suoi discendenti. Analogamente alla funzione $va(\cdot)$, possiamo definire la funzione di visita posticipata, $vp(\cdot)$. Una particolare visita posticipata è data dalla funzione inversa di $va(\cdot)$: $va^{-1}(\,j\,) = i \Leftrightarrow va(\,i\,) = j$. Con riferimento all'esempio in Figura B.10 si ha:

i	1	2	3	4	5	6	7	8	9
$va(i)$	2	4	7	5	8	9	6	3	1
$va^{-1}(i)$	9	1	8	2	4	7	3	5	6

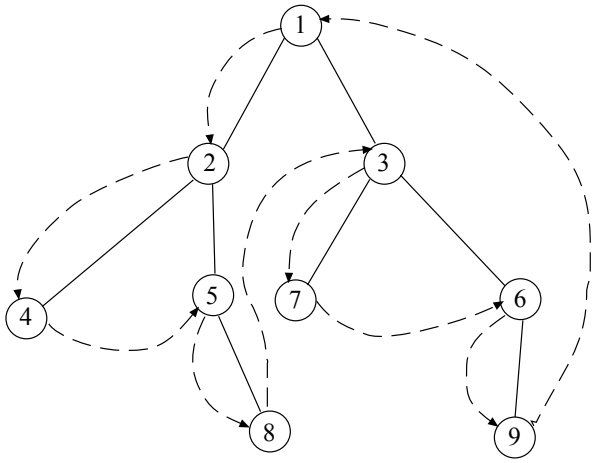


Figura B.10: La funzione $va(\cdot)$ della visita anticipata

B.2.4 Livello dei nodi di un albero

La funzione *livello*, $lev(\cdot)$, dei nodi di un albero associa ad ogni nodo i il suo livello, cioè il numero di archi dell'unico cammino nell'albero tra la radice r e i; $lev(\cdot)$ può essere facilmente calcolata, date le funzioni predecessore p e visita anticipata $va(\cdot)$, mediante la seguente procedura Livello:

```
procedure lev = Livello( r , p , va ) {
  lev[ r ] = 0; u = va[ r ];
  while( u ≠ r ) do { lev[ u ] = lev[ p[ u ] ] + 1; u = va[ u ]; }
  }
```

Procedura 2.1: Calcolo della funzione *livello*

Esercizio B.28. Determinare $lev(\cdot)$ per l'albero in Figura B.10.

B.3 Visita di un grafo

Gli *algoritmi di visita* sono strumenti che consentono di individuare degli insiemi di nodi o delle porzioni di grafo che soddisfano particolari proprietà. Nel seguito descriveremo prima la versione base della procedura di visita, che risolve il problema di determinare, dato un grafo orientato $G = (\,N\,,\,A\,)$, l'insieme dei nodi raggiungibili per mezzo di un cammino orientato a partire da un dato nodo r. Mostreremo poi come tale versione di base possa essere utilizzata o adattata per risolvere problemi diversi. La procedura *Visita* riceve in input il grafo orientato $G = (\,N\,,\,A\,)$ ed un nodo origine o radice r, e determina i nodi raggiungibili da r per mezzo di cammini orientati. Tali cammini individuano un albero orientato $T_r = (\,N_r\,,\,A_r\,)$ che viene fornito in output per mezzo di un vettore di predecessori $p[\cdot]$ (o funzione predecessore, si veda il §B.2.2). Al termine della procedura, ogni nodo i tale che $p[\,i\,] = 0$ non è stato raggiunto nella visita. Per il suo funzionamento la procedura si avvale di un insieme, Q, che contiene i *nodi candidati*, cioè quei nodi che sono già stati raggiunti nell'esplorazione ma ancora non sono stati utilizzati per proseguirla. Ad ogni passo la procedura seleziona uno dei nodi in Q e prosegue la visita del grafo a partire da tale nodo; la correttezza dell'algoritmo non dipende dal modo in cui il nodo è selezionato, ossia dall'implementazione della funzione *Select*.

```
procedure p = Visita ( G , r )  {
  foreach( i ∈ N ) do p[ i ] = 0;
  Q = { r }; p[ r ] = r
  do {  i = Select( Q ); Q = Q \ { i };
        foreach( ( i , j ) ∈ FS[ i ] ) do
          if( p[ j ] == 0 ) then { p[ j ] = i; Q = Q ∪ { j }; }
        } while( Q ≠ ∅ );
  }
```

Procedura 2.2: Visita di un grafo

Il predecessore di ogni nodo j viene inizializzato a 0, e posto al valore i del nodo da cui j è visto per la prima volta (se questo accade); si noti che stiamo assumendo che i nomi dei nodi siano gli interi $1, \ldots, n$ (altrimenti si può usare al posto di 0 qualsiasi valore "che sia chiaramente non il nome di un nodo", ad esempio un valore negativo). Pertanto, ogni nodo i del grafo viene inserito in Q solamente la prima volta che viene raggiunto (se questo accade). Quindi non si avranno più di n inserzioni e rimozioni di nodi da Q, e ogni arco (i, j) verrà esaminato al più una volta, se e quando i viene estratto da Q. Si noti che per questo motivo il predecessore della radice viene posto al valore $r\,(\neq 0)$ in fase di inizializzazione; in questo modo r non viene mai re-inserito in Q durante il ciclo. Pertanto il numero globale di ripetizioni delle operazioni effettuate nel ciclo "**do** ... **while**" è limitato superiormente da m. Supponendo che le operazioni di gestione di Q abbiano complessità $O(1)$, si ha che la complessità di *Visita* è $O(m)$.

Esercizio B.29. Realizzare la procedura *Visita* per grafi memorizzati mediante liste di adiacenza.

B.3.1 Implementazioni della procedura di visita

La correttezza della procedura di visita descritta nel paragrafo precedente è indipendente da:

- l'ordine con cui vengono esaminati gli archi della $FS()$ del nodo i estratto da Q, e
- l'ordine con cui vengono estratti i nodi da Q, ossia in che modo l'insieme viene implementato.

Questo significa che, indipendentemente dall'implementazione di queste due operazioni, si ottengono comunque tutti i nodi raggiungibili da r per mezzo di cammini orientati. Implementazioni diverse possono però fornire, al termine della procedura, insiemi di cammini diversi. Non ci soffermeremo sull'effetto dell'ordinamento dei nodi nelle $FS()$, che di solito, in pratica, dipende dai dettagli dell'implementazione delle strutture dati con cui è rappresentato il grafo, e spesso, in ultima analisi, dall'ordine con cui sono descritti gli archi nell'input della procedura. Per semplicità, nel seguito assumeremo che le $FS()$ siano ordinate in senso crescente degli indici dei nodi testa.

Per quanto riguarda l'implementazione di Q, scelte diverse possono avere un impatto rilevante sull'insieme dei cammini individuati. In particolare, le implementazioni di Q come fila (*queue*) e pila (*stack*) corrispondono rispettivamente alle strategie di esplorazione del grafo note come *visita a ventaglio* (bfs, da *breadth-first search*) e *visita a scandaglio* (dfs, da *depth-first search*). Si noti che tutti i nodi inseriti in Q in quanto raggiungibili da uno stesso nodo i saranno "figli" di i nell'insieme di cammini determinato. In una visita *bfs*, tutti i figli di uno stesso nodo i vengono inseriti consecutivamente in Q, e quindi estratti consecutivamente da Q: di conseguenza, i discendenti di tali nodi possono essere estratti solamente dopo che l'ultimo di questi nodi è stato estratto, ossia i "fratelli vengono visitati prima dei figli". In una visita *dfs*, i "figli" del nodo estratto i vengono inseriti in cima alla pila, e quindi saranno estratti (visitati) prima dei "fratelli" di i che sono ancora in Q al momento in cui i viene estratto. Queste due strategie tendono a costruire insiemi di cammini con proprietà abbastanza diverse: in particolare, la visita a ventaglio (Q implementato come fila) tende a costruire cammini "corti" (in termini di numero di archi che li compongono), mentre la visita a scandaglio (Q implementato come pila) tende a costruire cammini "lunghi".

Esempio B.1. Diversi tipi di visita di un grafo

Applichiamo la procedura *Visita* al grafo in Figura B.11(a) partendo dal nodo $r = 1$.

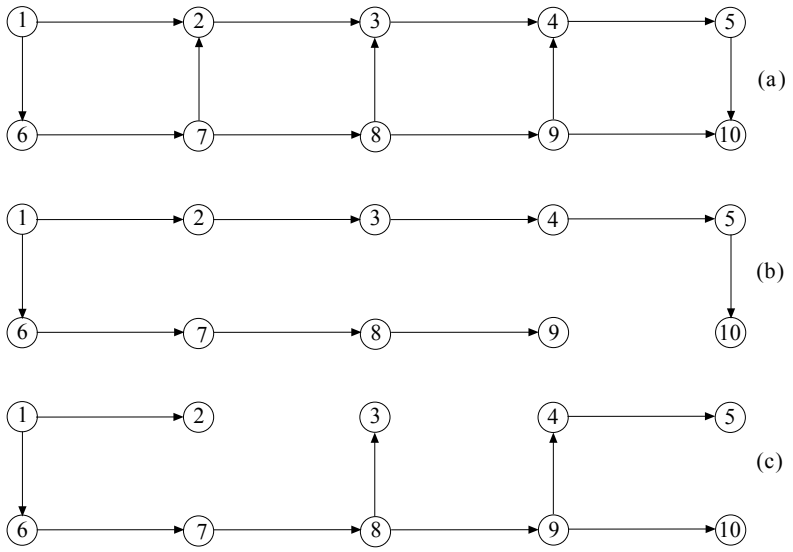


Figura B.11: Applicazioni della procedura *Visita*

Se Q è implementato mediante una fila (*queue*), il risultato, in termini di cammini determinati, è mostrato in Figura B.11(b), L'ordine di inserzione in (e rimozione da) Q è: 1, 2, 6, 3, 7, 4, 8, 5, 9, 10. La rimozione di 1 da Q causa l'inserimento di 2 e 6 mediante gli archi $(1, 2)$ e $(1, 6)$; la rimozione di 2 causa l'inserimento di 3 mediante l'arco $(2, 3)$; la rimozione di 6 causa l'inserimento di 7 mediante l'arco $(6, 7)$, ecc. La funzione predecessore è definita dal vettore $p = [1, 1, 2, 3, 4, 1, 6, 7, 8, 5]$. In questo caso il risultato è che tutti i nodi sono raggiungibili da r. Diverso è il risultato, mostrato in Figura B.11(c), se Q è implementato mediante una pila (*stack*). In questo caso 2 e 6, "figli" di 1, vengono esaminati in ordine inverso (rispetto al caso precedente) in quanto 6 è inserito in Q dopo 2, e quindi ne è estratto prima. Inoltre, quando 6 viene rimosso da Q segue l'inserzione di 7 in cima alla pila, e quindi l'esplorazione prosegue da 7, "figlio" di 6, piuttosto che da suo "fratello" 2, e così via. Si noti come i cammini prodotti in questo caso possano essere diversi e più lunghi di quelli prodotti nel caso precedente; in particolare, nel caso (c) i cammini da 1 a 4 e da 1 a 5 sono formati rispettivamente da 5 e 6 archi, mentre nel caso (b) erano formati rispettivamente da 3 e 4 archi.

In effetti, è possibile dimostrare il seguente interessante risultato:

Teorema B.1. La procedura di visita in cui Q è implementato come fila determina, per ogni nodo i raggiungibile da r, un cammino orientato da r a i di lunghezza minima in termini di numero di archi.

Dimostrazione. Per ogni nodo i raggiungibile da r, denotiamo con $d(i)$ la "distanza" di i da r, ossia la lunghezza (in termini di numero di archi) del più corto cammino tra i ed r ($d(r) = 0$). Dimostriamo per induzione che ogni nodo i con $d(i) = k > 0$ è inserito in Q—e quindi, siccome Q è una fila, estratto da Q—dopo tutti i nodi h con $d(h) = k - 1$ e prima di qualsiasi nodo j con $d(j) > k$ (se ne esistono); inoltre, al momento di essere inseriti in Q il loro predecessore $p[i] = h$ ha $d(h) = k - 1$ (ossia il cammino determinato dalla visita ha lunghezza pari a k). Questo è certamente vero per $k = 1$: un nodo i ha distanza 1 da r se e solo se $(r, i) \in FS(r)$, e tutti questi nodi sono esaminati e posti in Q alla prima iterazione della visita, quando viene esaminato r. Per il passo induttivo, supponiamo che la proprietà sia vera per k e dimostriamo che è vera per $k + 1$. Dall'ipotesi induttiva segue immediatamente che quando il *primo* nodo i con $d(i) = k$ è estratto da Q, tutti i nodi h che erano stati estratti da Q in precedenza avevano $d(h) < k$: quindi, in quel momento tutti i nodi j con $d(j) > k$ non sono ancora stati inseriti in Q, e quindi hanno $p[j] = 0$. Un nodo j ha $d(j) = k + 1$ se e solo se esiste almeno un nodo i con $d(i) = k$ tale che $(i, j) \in A$; siccome Q è una fila, dall'ipotesi induttiva segue che tutti i nodi con $d(j) = k + 1$ sono inseriti in coda alla fila nelle iterazioni della visita in cui sono estratti da Q ed esaminati tutti i nodi i con $d(i) = k$, e che il loro predecessore è uno di tali nodi. Da questo segue che la proprietà è vera anche per $k + 1$, e quindi il teorema. □

Quindi, la procedura di visita è, in alcuni casi, in grado di calcolare insiemi di cammini che utilizzano il minimo numero di archi: il §2.2 mostra come affrontare il problema generale dei cammini minimi.

B.3.2 Usi della procedura di visita

La procedura sopra descritta può essere modificata per risolvere anche altri problemi, tra i quali citiamo i seguenti:

- determinare l'insieme dei nodi raggiungibili per mezzo di un cammino orientato a partire da un dato *insieme* $R \subset N$ di nodi;
- determinare l'insieme dei nodi a partire dai quali un dato nodo r è raggiungibile per mezzo di un cammino orientato;
- determinare l'insieme dei nodi raggiungibili per mezzo di un cammino *non orientato* a partire da un dato nodo r, o, equivalentemente, determinare l'insieme dei nodi raggiungibili a partire da un dato nodo r su un grafo *non orientato*;
- individuare se un grafo è aciclico e, se non lo è, determinare un ciclo del grafo;
- determinare le componenti connesse di un grafo;
- determinare se un grafo è bipartito.

Tali problemi sono risolubili con piccole modifiche alla procedura *Visita*, e/o applicando la procedura ad un opportuno grafo ottenuto a partire da quello originario.
Ad esempio, supponiamo di voler determinare l'insieme dei nodi raggiungibili per mezzo di un cammino orientato a partire da un dato insieme R di nodi; è facile verificare che questo problema può essere risolto mediante un'applicazione della procedura *Visita* al grafo $G' = (N' , A')$ in cui $N' = N \cup \{ s \}$, dove s è un nodo fittizio che funge da "super-radice", e $A' = A \cup \{ (s , r) : r \in R \}$. Per il problema di determinare l'insieme dei nodi a partire dai quali r è raggiungibile per mezzo di un cammino orientato, invece, è sufficiente applicare la procedura *Visita*, con la stessa radice, al grafo $G' = (N , A')$ che ha gli stessi nodi del grafo originario G ma i cui archi sono "invertiti" rispetto a quelli di G, ossia tale che $A' = \{ (j , i) : (i , j) \in A \}$. Analogamente, se si vuole determinare l'insieme dei nodi raggiungibili da r per mezzo di un cammino non orientato, è sufficiente applicare la procedura al grafo $G' = (N , A')$ in cui $A' = A \cup \{ (j , i) : (i , j) \in A \}$.
Si noti che, negli esempi precedenti, è possibile evitare di costruire una rappresentazione del grafo G' modificando opportunamente la procedura *Visita* in modo che possa lavorare direttamente sulle strutture dati che descrivono il grafo originario G. Ad esempio, nel caso della determinazione dell'insieme dei nodi raggiungibili per mezzo di un cammino orientato a partire da un dato insieme R di nodi, è solamente necessario sostituire le istruzioni

$$p[r] = r;\ Q = \{ r \}; \qquad \text{con} \qquad \textbf{for each}(\ r \in R\ )\ \textbf{do}\ p[r] = r;\ Q = R;$$

In altri termini, basta inizializzare tutti i nodi di R come nodi radice e porli tutti in Q all'inizio dell'algoritmo. Per il problema di determinare l'insieme dei nodi a partire dai quali r è raggiungibile per mezzo di un cammino orientato, invece, è sufficiente modificare l'istruzione

$$\textbf{for each}(\ (i , j) \in FS(i)\ )\ \textbf{do}\ \ldots \qquad \text{in} \qquad \textbf{for each}(\ (j , i) \in BS(i)\ )\ \textbf{do}\ \ldots$$

Analogamente, per il problema della raggiungibilità attraverso cammini non orientati è sufficiente esaminare sia gli archi $(i , j) \in FS(i)$ che gli archi $(j , i) \in BS(i)$ corrispondenti al nodo i estratto da Q.

Esercizio B.30. Ovviamente, le operazioni precedenti si possono combinare: ad esempio, si discuta come modificare la procedura di visita per determinare l'insieme di nodi a partire dai quali almeno uno dei nodi in $R \subset N$ è raggiungibile mediante un cammino orientato.

Altri problemi possono essere risolti con l'uso ripetuto della procedura di visita o con modifiche minori alla stessa. Alcuni di questi problemi sono descritti nei seguenti esercizi.

Esercizio B.31. Si proponga un algoritmo di complessità $O(m)$, basato sulla procedura di visita, che determini il numero di componenti connesse di un grafo non orientato, fornendone lo pseudo-codice. Si

noti che in un grafo non orientato $FS(i)$ e $BS(i)$ non sono definite, è definita solamente la stella $S(i)$ degli archi incidenti nel nodo i (suggerimento: la visita a partire da un qualunque nodo i determina la componente connessa di cui i fa parte; al termine della visita i nodi delle altre componenti connesse hanno predecessore nullo).

Esercizio B.32. Si modifichi l'algoritmo dell'esercizio precedente in modo tale che, con la stessa complessità, produca un vettore $cc[\cdot]$ tale che $cc[i] = k$ se e solo se il nodo i appartiene alla k-esima componente connessa.

Esercizio B.33. Si costruisca una versione modificata della procedura di visita (fornendo lo in pseudo-codice) che risolva il problema di determinare se un dato grafo non orientato e connesso sia aciclico, ossia se contenga oppure no cicli. Nel caso che il grafo non sia aciclico, si richiede che la procedura produca in output (come "certificato") un ciclo del grafo. Si discuta la complessità di tale procedura.

Esercizio B.34. Si adatti l'algoritmo dell'esercizio precedente al caso di un grafo non connesso. Si discuta la complessità di tali procedure.

Esercizio B.35. Fornire un algoritmo di visita, di complessità $O(m)$, per verificare se un dato grafo non orientato, eventualmente non connesso, sia bipartito.

Esercizio B.36. Fornire un algoritmo di visita, di complessità $O(m)$, per verificare se un grafo, orientato e connesso, sia fortemente connesso (suggerimento: è sufficiente verificare che un arbitrario nodo r del grafo è connesso a tutti gli altri nodi e questi sono connessi ad r mediante cammini orientati).