



Appunti di Ricerca Operativa

**Giancarlo Bigi, Antonio Frangioni,
Giorgio Gallo, Stefano Pallottino,
Maria Grazia Scutellà**

Versione 1.1.0

Corso di Ricerca Operativa
Laurea Triennale in Informatica

Università di Pisa

25 agosto 2024

Dipartimento di Informatica
Università di Pisa

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 (programmazione nonlineare), né quelle relative al fatto che alcuni dati del problema possano non essere noti con certezza (programmazione stocastica o robusta), 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. Questa particolare versione è realizzata ad-hoc per il corso di Ricerca Operativa della Laurea Triennale in Informatica, e contiene la selezione del materiale strettamente corrispondente al programma del corso. Per non ingenerare confusione ed equivoci non dovrebbe essere distribuita a di fuori degli studenti del corso stesso. La versione completa delle dispense può sempre essere ottenuta come descritto nel seguito.

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 (in ordine cronologico) Giorgio Gallo, Stefano Pallottino, Maria Grazia Scutellà, Antonio Frangioni, e Giancarlo Bigi. Un aiuto particolare alla stesura e al miglioramento delle dispense è stato dato da Paola Cappanera, Maria Paola Scaparra, e Luca Mencarelli. 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 contribuito 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.

Questo materiale è distribuito sotto la licenza Creative Commons

Attribution-NonCommercial-NoDerivatives 4.0 International (CC BY-NC-ND 4.0)

<https://creativecommons.org/licenses/by-nc-nd/4.0>



La scelta della licenza corrisponde al fatto che l'utilizzo di questo materiale in corsi di studio diversi da quelli tenuti dagli estensori del documento è permesso ed incoraggiato, a condizione che il materiale sia distribuito nella sua forma originaria, sia opportunamente citata la fonte, e non venga tratto profitto dal fornire il materiale agli studenti. La modalità di distribuzione consigliata è quella di fare riferimento alla pagina web dei Corsi di Ricerca Operativa presso il Dipartimento di Informatica

<https://commalab.di.unipi.it/courses>

in cui si trovano le versioni più aggiornate del testo, insieme ad altro materiale che può risultare utile per gli studenti. Gli autori sono grati a tutti coloro che segnalino il loro utilizzo di queste dispense nei propri corsi, ed a maggior ragione che segnalino errori o suggeriscano possibili miglioramenti. Chiunque fosse intenzionato a contribuire alle dispense, ferma restando la licenza, può contattare gli estensori per accordarsi a tal proposito.

Indice

1	Problemi e Modelli	0
1.1	Modelli e Problemi	1
1.1.1	Problemi, istanze, soluzioni	2
1.1.2	Algoritmi esatti, euristici, approssimati, e rilassamenti	6
1.2	Tecniche di Modellazione	7
1.2.1	Programmazione Lineare	9
1.2.2	Variabili logiche	12
1.2.3	Relazioni binarie	15
1.2.4	Vincoli di assegnamento e semiassegnamento	17
1.2.5	Selezione di sottoinsiemi	21
1.2.6	Problemi di flusso e di cammino	24
1.2.7	Variabili a valori discreti	28
1.2.8	Variabili semicontinue e funzioni lineari a tratti	29
1.2.9	Funzioni lineari a tratti “facili”	31
1.2.10	Vincoli disgiuntivi	35
1.2.11	Esempi ed esercizi di modellazione	38
2	Grafi e reti di flusso	48
2.1	Flussi su reti	48
2.1.1	Alcuni modelli di flusso	50
2.1.2	Trasformazioni equivalenti	51
2.2	Cammini di costo minimo	52
2.2.1	Il problema	53
2.2.2	Alberi, etichette e condizioni di ottimo	55
2.2.3	L'algoritmo <i>SPT</i>	56
2.2.4	Algoritmi a coda di priorità	59
2.2.5	Algoritmi a selezione su lista	61
2.2.6	Cammini minimi su grafi aciclici	64
2.2.7	Cammini minimi con radici multiple	67
2.3	Albero di copertura di costo minimo	67
2.3.1	Algoritmo di Kruskal	69
2.3.2	Algoritmo di Prim	70
2.4	Il problema di flusso massimo	72
2.4.1	Tagli, cammini aumentanti e condizioni di ottimo	73
2.4.2	Algoritmo per cammini aumentanti	75
2.4.3	Flusso massimo con più sorgenti / pozzi	77
2.5	Il problema del Flusso di Costo Minimo	79
2.5.1	Cammini, cicli aumentanti e condizioni di ottimo	79
2.5.2	Algoritmo basato su cancellazione di cicli	81
2.5.3	Algoritmo basato su cammini minimi successivi	83
2.6	Problemi di accoppiamento	87
2.6.1	Accoppiamento di massima cardinalità	88

2.6.2	Assegnamento di Costo Minimo	90
3	Programmazione Lineare	94
3.1	Problemi di Programmazione Lineare	94
3.1.1	Geometria della Programmazione Lineare	97
3.2	Teoria della Dualità	108
3.2.1	Coppie di problemi duali	108
3.2.2	I teoremi debole e forte della dualità	111
3.2.3	Il teorema degli scarti complementari	113
3.2.4	Soluzioni complementari e basi	116
3.3	Algoritmi del Simplexso	119
3.3.1	L'algoritmo del Simplexso Primale	119
3.3.2	L'algoritmo del Simplexso Duale	130
3.3.3	Versioni specializzate degli algoritmi del simplexso	137
3.3.4	Analisi post-ottimale	138
4	Ottimizzazione Combinatoria	144
4.1	Introduzione	144
4.2	Programmazione Lineare Intera (Mista)	146
4.2.1	Il rilassamento continuo	146
4.2.2	Formulazioni di PL equivalenti per la PLI	149
4.2.3	Diseguaglianze valide	151
4.3	Dimostrazioni di ottimalità	155
5	Algoritmi enumerativi	158
5.1	Algoritmi di enumerazione implicita	158
5.2	Implementare un algoritmo enumerativo	165
5.2.1	Rilassamento ed euristica	165
5.2.2	La strategia di visita	165
5.2.3	Regole di branching	166
5.2.4	Preprocessing	168
5.3	Esempi di algoritmi enumerativi	169
5.3.1	Il problema dello zaino	169
5.3.2	Il problema del commesso viaggiatore	172
5.3.3	Il problema del cammino minimo vincolato	173
5.4	Tecniche poliedrali	175
5.4.1	I tagli di Chvátal	175
5.4.2	I tagli di Gomory	176
A	Algoritmi e complessità	182
A.1	Modelli computazionali	182
A.2	Misure di complessità	182
A.3	Problemi trattabili e problemi intrattabili	183
A.3.1	Le classi \mathcal{P} e \mathcal{NP}	184
A.3.2	Problemi \mathcal{NP} -completi e problemi \mathcal{NP} -ardui	184
A.3.3	Complessità ed approssimazione	185
A.4	Algoritmi	186
A.4.1	Algoritmi greedy	186
A.4.2	Algoritmi di ricerca locale	189
B	Grafi e Reti	191
B.1	I grafi: notazione e nomenclatura	191
B.1.1	Nodi, archi	191

B.1.2	Cammini, cicli	192
B.1.3	Tagli e connettività	193
B.1.4	Alberi	194
B.2	Rappresentazione di grafi ed alberi	195
B.2.1	Matrici di incidenza e liste di adiacenza	195
B.2.2	Rappresentazione di alberi: la funzione predecessore	197
B.2.3	Visite di un albero	197
B.2.4	Livello dei nodi di un albero	198
B.3	Visita di un grafo	198
B.3.1	Implementazioni della procedura di visita	199
B.3.2	Usi della procedura di visita	200

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.

1.1.1 Problemi, istanze, soluzioni

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*

$$c : F \rightarrow \mathbb{R}$$

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

$$(P) \quad \min\{c(x) : x \in F\} . \quad (1.1)$$

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

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

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, \dots, 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 *reformulazione*. 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

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

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

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

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 $I = \{1, 2, \dots, n\}$ 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:

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

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

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

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

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:

- 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.
- 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 $z(P) = -\infty$ ($+\infty$).
- 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

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

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.

1.1.2 Algoritmi esatti, euristici, approssimati, e rilassamenti

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 $\mathcal{A} : P \rightarrow \mathbb{R}$. 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*, 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

$$(\bar{P}) \quad \min \{ \bar{c}(x) : x \in \bar{F} \} \quad (1.8)$$

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.

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

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

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 $\bar{R}_{\bar{x}} \leq \varepsilon$, 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 peggio, per molti problemi di ottimizzazione; si veda il Capitolo 5.

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 descri-

zione 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 nel Capitolo 5; 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 $f : \mathbb{R}^n \rightarrow \mathbb{R}$, 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.

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

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

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:

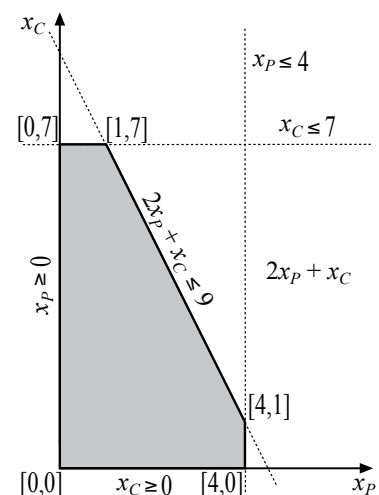


Figura 1.1: Insieme ammissibile per il problema della *Pintel*

$$x_P = w_P \cdot 300 \cdot 0.5 = w_P \cdot 150 \quad , \quad x_C = w_C \cdot 500 \cdot 0.6 = w_C \cdot 300 \quad . \quad (1.10)$$

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

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

Quindi, l'insieme ammissibile per il problema è

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

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:

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

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

$$\begin{array}{rcll} \max & 500x_P & + & 200x_C \\ & x_P & & \leq 400000 \\ & & & x_C \leq 700000 \\ & 2x_P & + & x_C \leq 900000 \\ & x_P & , & x_C \geq 0 \end{array} \quad (1.11)$$

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 $[w_P, w_C] = [2666.\bar{6}, 333.\bar{3}]$. 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 *rilasamento* (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 peggio dare un ricavo inferiore di 10000\$ rispetto a quella ottima, il che porta ad una stima (superiore) sull'errore relativo di $10000/219990000 \approx 0.00045$, 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 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:

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

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 nell'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 di 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”, come discusso nel Capitolo 4.

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 $b > 0$. Poiché $b < \sum_i a_i$, ossia non può rubare tutto quello che vorrebbe, deve decidere quale sottoinsieme dei reperti prendere, e di conseguenza quale lasciare. Ovviamente ha scartato a priori i reperti per cui vale $a_i > b$, ossia che non che può comunque trasportare. 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 $I = \{1, 2, \dots, n\}$ degli indici degli elementi tra cui scegliere (reperti), il che porta immediatamente alla definizione combinatoria

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

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

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

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 la capacità delle linee è sufficientemente grande per cui ciò non è necessario: è possibile (se più conveniente) collegare gruppi di filiali "vicine" tra loro, e solo una di esse al CEU. Il problema è quindi determinare quali linee affittare 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 richiami sui grafi si veda l'Appendice B). Nel grafo, ogni vertice rappresenta una filiale, tranne un vertice "radice" che rappresenta il CEU, ed i lati 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 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

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

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* $(S, N \setminus S)$; si veda ancora l'Appendice B). Pertanto, imponendo i seguenti *vincoli di connessione*

$$\sum_{i \in S, j \notin S} x_{ij} \geq 1 \quad \emptyset \subset S \subset V \quad (1.12)$$

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

$$\sum_{\{i, j\} \in E} c_{ij} x_{ij} , \quad (1.13)$$

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

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

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 momenti 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. 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:

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

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

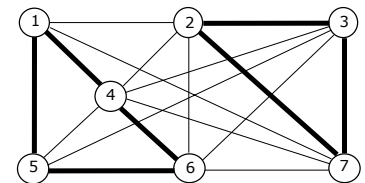


Figura 1.2: una copertura per cicli (archi evidenziati)

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

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 §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 $a \implies b$ (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:

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

Infatti, le due prime disuguaglianze 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:

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

Infatti, le due prime disuguaglianze 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:

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

Infatti, le prime due disuguaglianze 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 open-source 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

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

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

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

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*

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

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

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

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, \dots, P_n che renda vera la formula F . Tale problema può essere rappresentato come *PLI* in modo particolarmente semplice. Introduciamo gli insiemi $J = \{1, 2, \dots, m\}$ degli indici delle clausole e $I = \{1, 2, \dots, n\}$ 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, \dots, 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

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

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

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

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, \dots, 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).

La relazione tra ottimizzazione e calcolo logico è profonda. (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* (si veda il §5.4).

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 diverse. 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, \dots, 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*

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

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 nel §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 c'è simmetria nel senso che ciascun elemento di un insieme deve 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 dà 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 $I = \{1, 2, \dots, n\}$. In tal caso, i vincoli di assegnamento tra I ed I impongono un ordinamento dei lavori: $x_{ij} = 1$ 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.

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 inevitabile. Si parla in questo caso di *vincoli di semiassegnamento*. Il modello che segue 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 aventi 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 qualora 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 vertici sono le antenne ed esiste il lato $\{i, j\} \in E$ se e solo se le due antenne i e j sono interferenti. 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 assegnata frequenza $f \in F$, con la condizione che siano tutte abbastanza lontane 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

$$\sum_{f \in F} x_{if} = 1 \quad i \in S. \quad (1.16)$$

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

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

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 $\neg x_{if} - x_{jf} \geq 1 - 2$. 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

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

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

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

$$ny_f \geq \sum_{i \in F} x_{if} \quad f \in F \quad (1.19)$$

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 " $x_{if} \implies y_f$ ", 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

$$\begin{aligned} \min \quad & \sum_{f \in F} y_f \\ \text{(GC)} \quad & \sum_{f \in F} x_{if} = 1 \quad i \in S \\ & x_{if} + x_{jf} \leq 1 \quad (i, j) \in E, \quad f \in F \\ & y_f \geq x_{if} \quad i \in S, \quad f \in F \\ & x_{if} \in \{0, 1\} \quad i \in S, \quad f \in F \\ & y_f \in \{0, 1\} \quad f \in F \end{aligned}$$

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 $y_f \geq x_{if} = 1$ 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 però 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 $m \geq 2$) 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 con 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 $M = \{1, 2, \dots, p\}$ di moli spaziali che può usare per le operazioni di carico e scarico dei trasporti. Per ogni giorno è noto l'insieme $T = \{1, 2, \dots, n\}$ 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 HSCo 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 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*:

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

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

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

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, $K = \{1, 2, \dots, k\}$, 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 e

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

una famiglia di m suoi sottoinsiemi, ovvero $F_j \subseteq E$ per $j = 1, \dots, m$. Con un piccolo abuso di notazione, frequentemente utilizzato, possiamo alternativamente considerare $F = \{1, 2, \dots, m\}$ 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, \dots, x_m , col significato che $x_j = 1$ se $F_j \in S$, e $x_j = 0$ altrimenti, per $j = 1, \dots, m$. 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*

$$\sum_{j: i \in F_j} x_j \geq 1 \quad i \in E, \quad (1.20)$$

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

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

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

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

- *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*

$$\sum_{j:i \in F_j} x_j = 1 \quad i \in E, \quad (1.21)$$

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):

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

- *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*

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

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

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

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 sembrano “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 paghettoni 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 $c \in C = \{1, \dots, 8\}$ che indicano se vengono montate le corrispondenti combinazioni, le variabili y_a (interi 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:

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

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 $d \rightarrow i \rightarrow d$ per $i \in I$ oppure $d \rightarrow e \rightarrow d$ per $e \in E$;
- P_2 tutti i cammini che toccano esattamente due utenti, ossia della forma $d \rightarrow i_1 \rightarrow i_2 \rightarrow d$ per $i_1, i_2 \in I$ oppure $d \rightarrow i \rightarrow e \rightarrow d$ per $i \in I$, $e \in E$ oppure $d \rightarrow e_1 \rightarrow e_2 \rightarrow d$ 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 $d \rightarrow i_1 \rightarrow i_2 \rightarrow e \rightarrow d$ per $i_1, i_2 \in I$ ed $e \in E$, oppure $d \rightarrow i \rightarrow e_1 \rightarrow e_2 \rightarrow d$ 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 importatore, quindi i cammini corrispondenti sono esclusi);

- P_4 tutti i cammini che toccano esattamente quattro utenti, ossia della forma $d \rightarrow i_1 \rightarrow i_2 \rightarrow e_1 \rightarrow e_2 \rightarrow d$ 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:

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

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 Tartarughe 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 condotta 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*.

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 capacità 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 provenendo 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

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

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

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

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

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

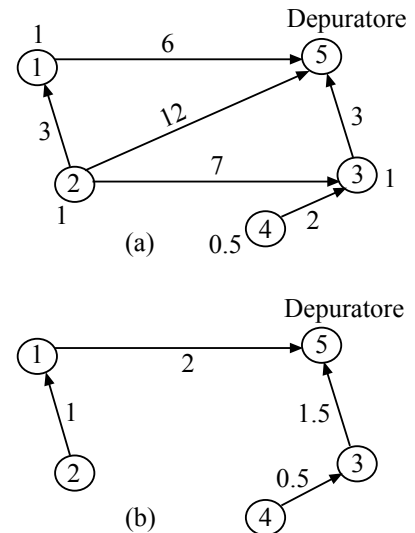


Figura 1.3: Progetto dell'espansione della rete fognaria di New York

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

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

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*

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

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

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

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" $(i, j) = ((s_i, t_i), (s_j, t_j)) \in A$ 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:

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

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 disuguaglianze) su entrambi i lati ed osservando che

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

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 nei Capitoli 2 e 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 là 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 $x \in V = \{v_1, v_2, \dots, v_k\}$. 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, \dots, 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

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

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

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

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_{ij}^1, y_{ij}^2 e y_{ij}^3 e sostituiamo il vincolo $y_{ij} \in V$ con

$$0.7y_{ij}^1 + 1.4y_{ij}^2 + 3y_{ij}^3 = y_{ij} \quad (1.23)$$

$$y_{ij}^1 + y_{ij}^2 + y_{ij}^3 \leq 1 \quad (1.24)$$

$$y_{ij}^1, y_{ij}^2, y_{ij}^3 \in \{0, 1\}$$

Utilizzando (1.23) possiamo poi sostituire y_{ij} ovunque con l'espressione (lineare) $0.7y_{ij}^1 + 1.4y_{ij}^2 + 3y_{ij}^3$, 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 *convessa* 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

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

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.

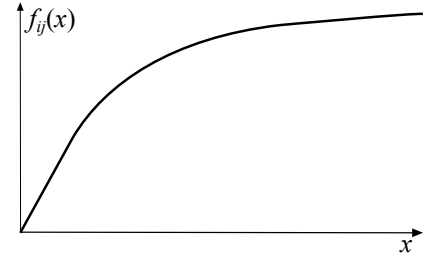


Figura 1.4: Funzione costo per il problema di progetto di rete

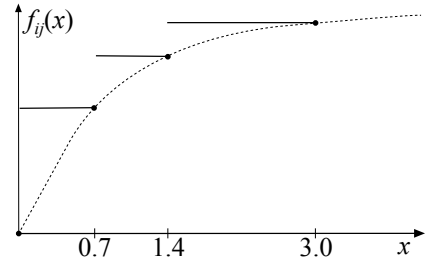


Figura 1.5: Una funzione costo a gradini

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

$$ly \leq x \leq uy \quad , \quad y \in \{0, 1\} \quad , \quad x \in \mathbb{R} \quad (1.25)$$

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*

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

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 $0 < x \leq 250$.

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 $0 \leq l_1 < u_1 \leq l_2 < u_2$ si traduce in

$$\begin{aligned} x &= x_1 + x_2, \quad y_1 + y_2 \leq 1, \quad [y_1, y_2] \in \{0, 1\}^2 \\ l_1 y_1 &\leq x_1 \leq u_1 y_1 \quad l_2 y_2 \leq x_2 \leq u_2 y_2 \end{aligned} \quad (1.27)$$

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

$$b_2 + c_2 a_2 \geq b_1 + c_1 a_2. \quad (1.28)$$

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

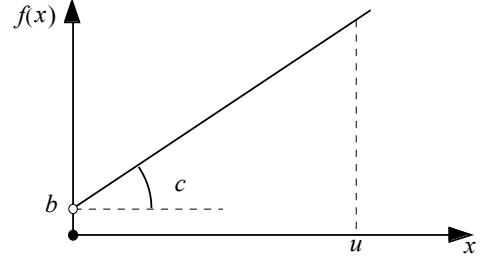


Figura 1.6: Una funzione con “carico fisso”

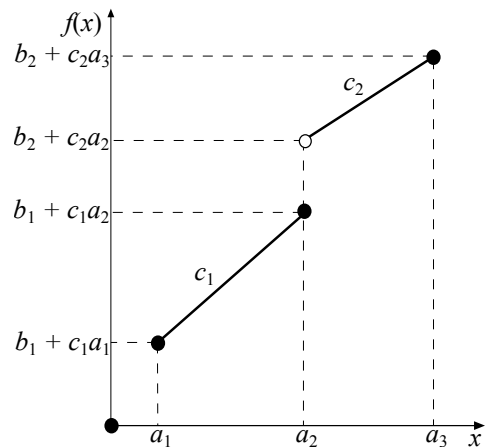


Figura 1.7: Una funzione lineare a due tratti

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

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 $x = a_2$).

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

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

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

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

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

$$\begin{aligned} \min \quad & \sum_{i=1}^n (b_i + c_i a_i) y_i + \sum_{i=1}^n c_i z_i \\ & x = \sum_{i=1}^n a_i y_i + \sum_{i=1}^n z_i \\ & 0 \leq z_i \leq (a_{i+1} - a_i) y_i \quad i = 1, \dots, n \quad . \\ & \sum_{i=1}^n y_i = 1 \\ & y_i \in \{0, 1\} \quad i = 1, \dots, n \end{aligned} \quad (1.31)$$

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, \dots, v_k ed altri due numeri reali l ed u , allora

$$\begin{aligned} l \leq \min\{v_i : i = 1, \dots, k\} &\equiv l \leq v_i \quad i = 1, \dots, k \\ u \geq \max\{v_i : i = 1, \dots, k\} &\equiv u \geq v_i \quad i = 1, \dots, k \end{aligned} \quad (1.32)$$

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

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

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 $x_j, j \in J = \{1, \dots, n\}$, semplicemente inserendo il vincolo

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

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)

$$F = \{f_i(x) = \sum_{j \in J} a_j^i x_j + b^i, i = 1, \dots, k\}$$

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

$$\begin{aligned} l \leq \min\{f_i(x) : i \in F\} &\equiv l \leq \sum_{j \in J} a_j^i x_j + b^i \quad i \in F \\ u \geq \max\{f_i(x) : i \in F\} &\equiv u \geq \sum_{j \in J} a_j^i x_j + b^i \quad i \in F \end{aligned} \quad (1.33)$$

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

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

Ossia, si può sempre trasformare un (frammento di) funzione obiettivo $f(x)$ arbitrariamente 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

$$\begin{aligned} \min\{f(x) = \max_{i \in F} \{\sum_{j \in J} a_j^i x_j + b^i\}\} &\equiv \min\{v : v \geq \sum_{j \in J} a_j^i x_j + b^i \quad i \in F\} \\ \max\{f(x) = \min_{i \in F} \{\sum_{j \in J} a_j^i x_j + b^i\}\} &\equiv \max\{v : v \leq \sum_{j \in J} a_j^i x_j + b^i \quad i \in F\} \end{aligned} \quad (1.34)$$

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 HSCo 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 HSCo 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, $T = \{1, 2, \dots, n\}$ 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 HSCo 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 $T(m)$ 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 trasporto 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

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

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 di 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 completamento 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 $\min\{|3 - 4x| : |x| \leq 2\}$.

Esercizio 1.20. Siano a_1, \dots, 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

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

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

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

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 $\|x\|_1 \leq b$ e $\|x\|_\infty \leq b$ e la loro generalizzazione $\|x - \bar{x}\|_1 \leq b$ e $\|x - \bar{x}\|_\infty \leq b$.

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,

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

è massimo di (un numero finito di) funzioni lineari e quindi convessa; infatti, (1.34) mostra che il suo *epigrafo* $\text{epi}(f) = \{(v, x) : v \geq f(x)\}$ 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:

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

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

$$\begin{aligned} \min \quad & b_1 + \sum_{i=1}^n c_i z_i \\ & x = a_1 + \sum_{i=1}^n z_i \\ & 0 \leq z_i \leq a_{i+1} - a_i \quad i = 1, \dots, n \end{aligned} \quad (1.35)$$

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;

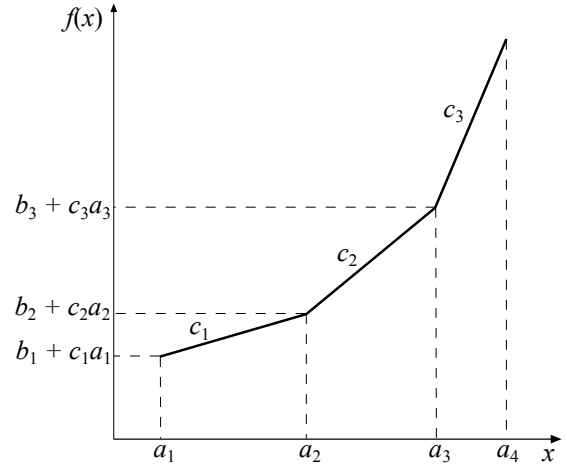


Figura 1.8: Una funzione lineare a tratti e convessa

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

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

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

$$A_i x \leq b_i \quad , \quad i = 1, \dots, m \quad (1.36)$$

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

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

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

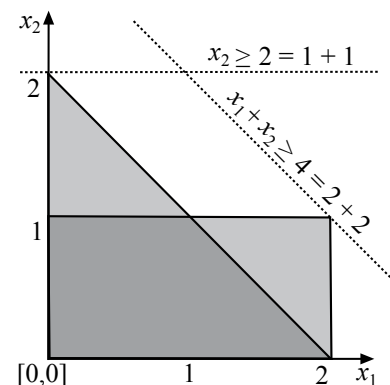


Figura 1.9: Rappresentazione di poliedri non convessi

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

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

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, \dots, S_p siano p sottoinsiemi, non necessariamente disgiunti, dell'insieme $\{1, 2, \dots, m\}$. Definiamo per ogni $h = 1, \dots, p$ l'insieme $X_h = \{x : A_i x \leq b_i \text{ } i \in S_h\}$ di tutti i punti che soddisfano i vincoli i cui indici sono in S_h , e consideriamo l'insieme $X = X_1 \cup X_2 \cup \dots \cup X_p$. 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

$$\begin{array}{rcll} A_i x - M_1 y_1 & \leq & b_i & i \in S_1 \\ A_i x & - M_2 y_2 & \leq & b_i \quad i \in S_2 \\ \vdots & & \vdots & \\ A_i x & & - M_p y_p & \leq b_i \quad i \in S_p \\ & y_1 + y_2 + \dots + y_p & \leq & p - 1 \\ & y_1, y_2, \dots, y_p & \in & \{0, 1\} \end{array}$$

Per ogni $h = 1, \dots, p$, 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 , $i = 1, \dots, p$). In questo caso è sufficiente sostituire al vincolo $y_1 + y_2 + \dots + y_p \leq p - 1$ il nuovo vincolo $y_1 + y_2 + \dots + y_p \leq p - k$.

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:

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

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 $i = 1, \dots, n$ 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

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

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

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

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 $2nr^2 \leq 1 \implies r \leq \sqrt{1/(2n)}$. 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 $i \in I = \{1, \dots, n\}$ si conosce il costo d_i di installazione, il numero massimo u_i di clienti che l’ufficio potrebbe assistere qualora fosse attivato, e per ogni cliente $j \in J = \{1, \dots, m\}$ 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

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

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

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

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

$$\sum_{j \in J} x_{ij} \leq u_i \quad i \in I , \quad (1.37)$$

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

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

Questi ultimi esprimono l'implicazione $x_{ij} > 0 \implies y_i = 1$. 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

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

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

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

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

Teucro 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 invisibili 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 modo 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

$$\begin{aligned} T(e) &= \{ (c, a) : (e, c, a) \in T \} \cup \{ (e', c) : (e', c, e) \in T \} \\ T(a) &= \{ (e, c) : (e, c, a) \in T \} & T(c) &= \{ (e, a) : (e, c, a) \in T \} \end{aligned}$$

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

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

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

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

Una formulazione del problema è la seguente:

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

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 Personalità 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 $h \in A = \{1, \dots, 5\}$, col significato che $x_{ih} = 1$ se il reparto i viene assegnato all'area h , una formulazione del problema è

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

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

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

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.

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

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.

$$\min 680x_1 + 705x_2 + 705x_3 + 705x_4 + 705x_5 + 680x_6 + 655x_7 + 125y_1 + 125y_2 + 125y_3 + 125y_4 + 125y_5 + 125y_6 + 125y_7$$

	x_2	$+x_3$	$+x_4$	$+x_5$	$+x_6$		$+y_1$							≥ 18
		x_3	$+x_4$	$+x_5$	$+x_6$	$+x_7$		$+y_2$						≥ 27
x_1			$+x_4$	$+x_5$	$+x_6$	$+x_7$			$+y_3$					≥ 22
x_1	$+x_2$			$+x_5$	$+x_6$	$+x_7$				$+y_4$				≥ 26
x_1	$+x_2$	$+x_3$			$+x_6$	$+x_7$					$+y_5$			≥ 25
x_1	$+x_2$	$+x_3$	$+x_4$			$+x_7$						$+y_6$		≥ 21
x_1	$+x_2$	$+x_3$	$+x_4$	$+x_5$									$+y_7$	≥ 19
$x_1,$	$x_2,$	$x_3,$	$x_4,$	$x_5,$	$x_6,$	$x_7,$	$y_1,$	$y_2,$	$y_3,$	$y_4,$	$y_5,$	$y_6,$	$y_7 \in \mathbb{N}$	

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

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

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:

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

La funzione obiettivo rappresenta il numero di Portali distrutti. Il secondo blocco di vincoli sono i classici vincoli di conservazione del flusso che 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 $i = 1, \dots, n$. 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. Poiché 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 $i = 1, \dots, n$.

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

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

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:

$$\begin{aligned}
\min \quad & v \\
& 0 \leq x_1 \quad , \quad 0 \leq y_1 \\
& x_{i+1} \geq x_i \quad , \quad y_{i+1} \geq y_i & i = 1, \dots, n-1 \\
& x_i + w_i \leq v \quad , \quad y_i + h_i \leq v & i = 1, \dots, n \\
& x_i + w_i \leq x_j + Wz_{ij}^{(1)} & (i, j) \in C \\
& y_i + h_i \leq y_j + Hz_{ij}^{(2)} & (i, j) \in C \\
& z_{ij}^{(1)} + z_{ij}^{(2)} \leq 1 & (i, j) \in C \\
& z_{ij}^{(h)} \in \{0, 1\} & h = 1, 2 \quad , \quad (i, j) \in C
\end{aligned}$$

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 $k/3 \leq h < k$, 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 però 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 ($\geq \alpha$) se j è la destinazione di un'Arca; altrimenti, possiamo scrivere che

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

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 $1 - p_j \leq 1 - \alpha$ 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:

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

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 c'è nessun motivo per non spedire sotto-Arche. Il vincolo critico è ottenuto a partire dalla condizione logica

$$\log(1 - p_j) \leq \log(1 - \alpha) \quad (*) .$$

Si noti innanzi tutto che, data la natura binaria di y_{ij} , le espressioni $\log(1 - y_{ij} P_{ij})$ 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_2 : 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 mischi 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. Il 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 $\{a_1, \dots, a_7\}$, 4 clubs politici $\{C_1, \dots, C_4\}$, e 3 partiti politici $\{P_1, \dots, P_3\}$. 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 , $i = 1, \dots, n$, 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 difforni 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, $i = 1, \dots, n$; inoltre il Capolimone conosce l'insieme $D(i)$ dei volontari in grado di collaborare col volontario i , $i = 1, \dots, n$. 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. 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 nel Capitolo 5.

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

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

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)

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

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*

$$l_{ij} \leq x_{ij} \leq u_{ij} \quad (i, j) \in A. \quad (2.3)$$

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

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

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

$$(MCF) \quad \min \{ cx : Ex = b, l \leq x \leq u \} \quad (2.4)$$

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 $l \leq x \leq u$ 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).

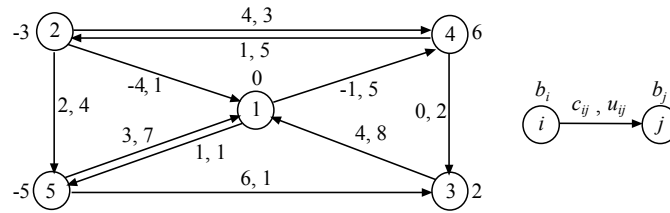


Figura 2.1: Un grafo con domande, offerte, costi e capacità

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

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

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

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

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 , $i = 1, \dots, 12$, 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 , $i = 1, \dots, 12$. 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'inizio 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, \dots , 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.

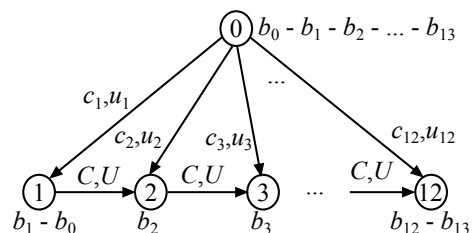


Figura 2.2: Il problema della PereCani

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 , $i = 1, \dots, n$, 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 , $i = 1, \dots, n$, 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 Andamamento Lento possenga 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 , $i = 1, \dots, n$. 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 $i = 1, \dots, n$, con costo c_i^I e capacità 1; analogamente, esistono archi (i'', t) per ogni $i = 1, \dots, n$, 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 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) .

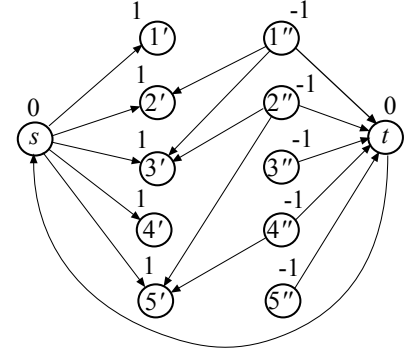


Figura 2.3: Schedulazione di viaggi

2.1.2 Trasformazioni equivalenti

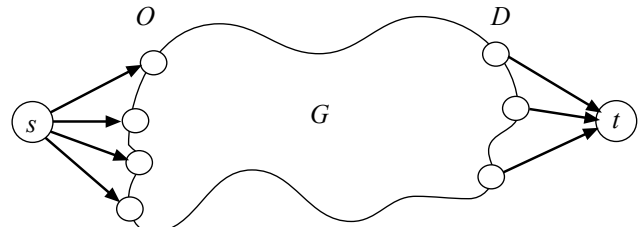
Molti problemi reali possono essere formulati come problemi di flusso di costo minimo; questo è in parte dovuto alla grande flessibilità del modello stesso. Infatti, sui problemi di (MCF) si possono fare alcune assunzioni che ne semplificano la descrizione e l'analisi: se tali assunzioni non sono rispettate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

- *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

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

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

Figura 2.4: La rete ampliata G'

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

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

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 ha 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 $N = \{0, 1, \dots, n\}$ 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 $n = 4$.

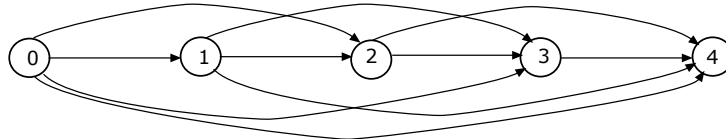


Figura 2.5: Grafo associato al problema delle ispezioni

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

$$(SP) \quad \min \{ C(P) : P \in \mathcal{P}_{rt} \} . \quad (2.5)$$

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

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

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 cammino 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 cammino Hamiltoniano minimo (qualora ne esista uno).

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

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

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

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

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 2.3. 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:

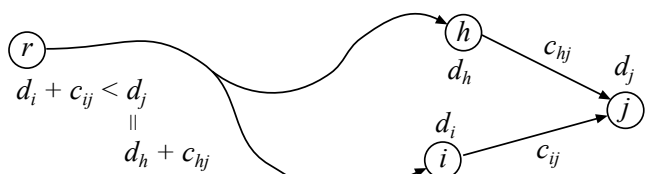


Figura 2.6: Un cammino di costo inferiore per j

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

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

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

$$\begin{array}{rcl}
d_{j_k} & \leq & d_{j_{k-1}} + c_{j_{k-1}j_k} \\
d_{j_{k-1}} & \leq & d_{j_{k-2}} + c_{j_{k-2}j_{k-1}} \\
\vdots & & \vdots \\
d_{j_2} & \leq & d_{j_1} + c_{j_1j_2}
\end{array}$$

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

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

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

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 $C = \{j_1, j_2, \dots, j_k, j_1\}$ un ciclo negativo, e supponiamo per assurdo che esista un vettore d che rispetta (2.7): si ha

$$\begin{array}{rcl}
d_{j_k} & \leq & d_{j_{k-1}} + c_{j_{k-1}j_k} \\
d_{j_{k-1}} & \leq & d_{j_{k-2}} + c_{j_{k-2}j_{k-1}} \\
\vdots & & \vdots \\
d_{j_2} & \leq & d_{j_1} + c_{j_1j_2} \\
d_{j_1} & \leq & d_{j_k} + c_{j_kj_1}
\end{array}$$

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 ciò 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 §A.4.2) 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  $(i \in N)$  do {  $p[i] = r; d[i] = M;$  }
   $d[r] = 0; Q = \{r\};$ 
  do { select  $i$  from  $Q; Q = Q \setminus \{i\};$ 
    foreach  $((i, j) \in FS(i))$  do
      if  $(d[i] + c_{ij} < d[j])$  then {
         $d[j] = d[i] + c_{ij}; p[j] = i; Q = Q \cup \{j\};$ 
      }
    } while  $(Q \neq \emptyset);$ 
}

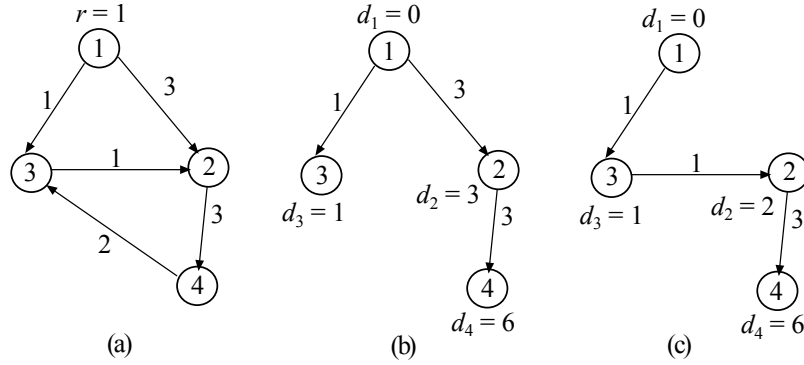
```

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 ciò 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).

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 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 stato 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 si fa 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, di cui omettiamo la dimostrazione:

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.

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.

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*.

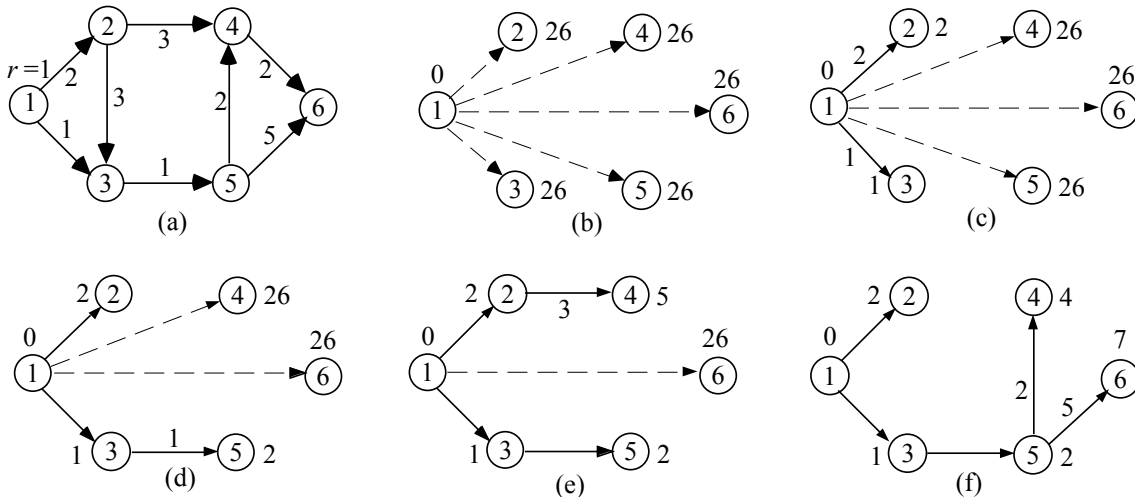
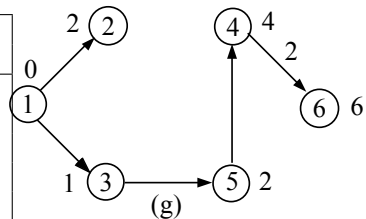


Figura 2.8: Alberi generati dall'Algoritmo *SPT.S*

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

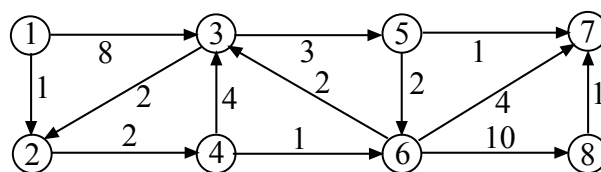


Figura 2.9: Un'istanza del problema (SPT)

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 2.5. 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 bilanciato*, 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.

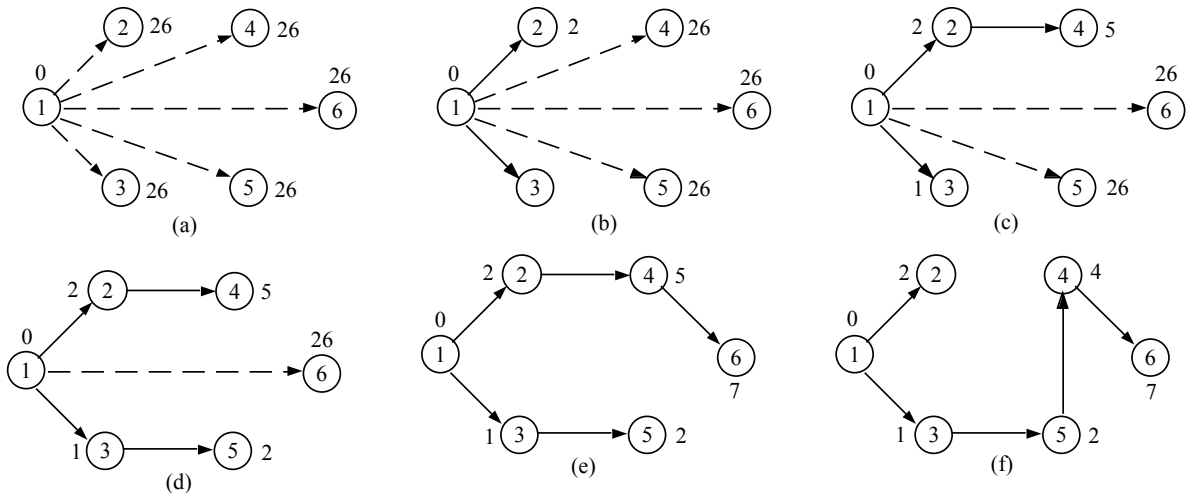
Indichiamo nel seguito con *SPT.L* le versioni di *SPT* nelle quali l'insieme Q è implementato come lista. La lista può essere realizzata in diversi modi (lista a puntatori, vettore di puntatori, lineare o circolare, semplice o doppia, ecc.), ed è sempre possibile fare in modo che le operazioni elementari 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 nella figura seguente. La simbologia nelle figure e nella tabella seguente coincide con quella utilizzata nell'Esempio 2.7.



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 2.6. 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. È possibile dimostrare un'utile conseguenza di questo tipo di visita, ossia 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 . 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 2.7. 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 2.10. 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).

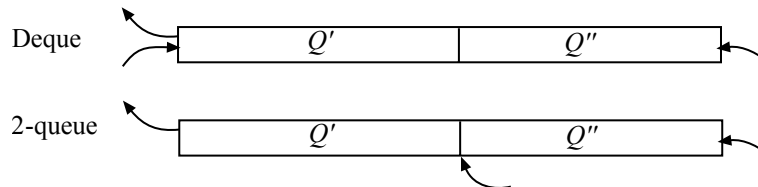


Figura 2.10: Liste a doppio ingresso

Esercizio 2.8. 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 2.10). 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 state 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

$$(i, j) \in A \implies i < j. \quad (2.8)$$

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 (2.8). 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 (2.8), può essere risolto per mezzo di una visita del grafo (si veda il paragrafo B.3) ed ha pertanto complessità $O(m)$.

Esercizio 2.9. 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 (2.8) (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 (2.8); 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] + cij < d[j]) then { d[j] = d[i] + cij; p[j] = i; }
}
```

Procedura 2.2: Algoritmo *SPT.Acyclic*

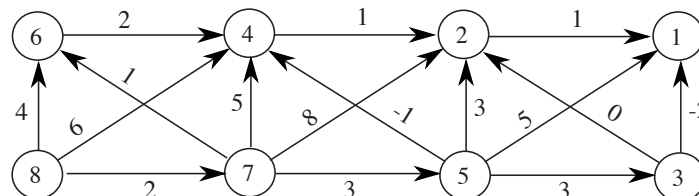
Esercizio 2.10. Applicare *SPT.Acyclic* al grafo di Figura 2.9 da cui siano stati eliminati gli archi (3, 2) e (6, 3), con radice $r = 1$, dopo aver eventualmente rinumerato i suoi nodi.

Esercizio 2.11. 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.Dequeue* 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.

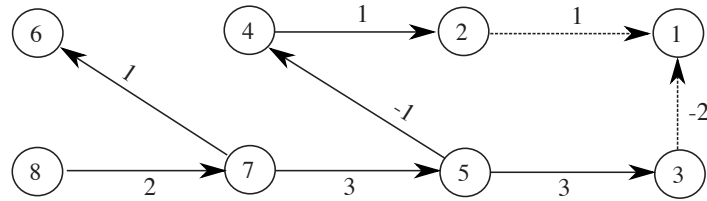


Iniziamo eseguendo *SPT.S*. Si noti che il grafo ha due archi di costo negativo, (5, 4) e (3, 1), quindi il Teorema 2.3 non si applica e non c'è garanzia che ogni nodo esca dalla coda una sola volta. Questo potrebbe in teoria portare la complessità dell'algoritmo ad essere esponenziale, ma ciò accade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

it.	i	$p[\cdot]$								$d[\cdot]$								Q
		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	$Q = \{8\}$
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 perché 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.

it.	i	$p[\cdot]$								$d[\cdot]$								Q
		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	$Q = \{8\}$
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 pessi-

mo. Anzi, *SPT.L.Queue*, che ha la migliore complessità al caso peggio, 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.

it.	i	$p[\cdot]$								$d[\cdot]$								Q
		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	$Q = \{8\}$
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 sì 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.Dequeue*. 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.

it.	i	$p[\cdot]$								$d[\cdot]$								Q
		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	$Q = \{8\}$
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 2.12. 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: iniziando $i = 1$, si cerca se esiste un nodo privo di archi entranti. Se ciò accade si dà 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 tutti 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)

it.	i	$p[\cdot]$								$d[\cdot]$							
		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

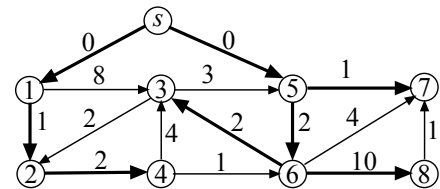
Questo algoritmo è chiaramente il più efficiente, non dovendo gestire nessuna Q e potendo anche fare un'iterazione in meno (visto che in un grafo aciclico il nodo n per definizione non può avere archi uscenti).

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

Esercizio 2.14. 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 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.

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. §A.4.1);

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  $S = \text{Greedy-MST}(G, c)$  {
   $S = \emptyset; R = \emptyset;$ 
  do applica Inserzione o Cancellazione
  while(  $S \cup R \subsetneq E$  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, di cui omettiamo la dimostrazione:

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.

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  $S = Kruskal(G, c)$  {
   $S = \emptyset$ ;  $R = \emptyset$ ;  $X = Sort(A)$ ;
  do { estrai da  $X$  il primo lato  $\{u, v\}$ ;
    if ( $Component(E, u, v)$ )
      then  $R = R \cup \{\{u, v\}\}$ ; /* cancellazione */
      else  $S = S \cup \{\{u, v\}\}$ ; /* inserzione */
    } while ( $|S| < n - 1$ );
}

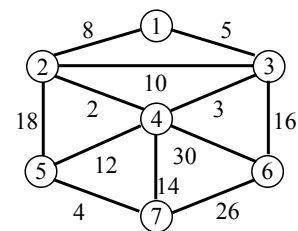
```

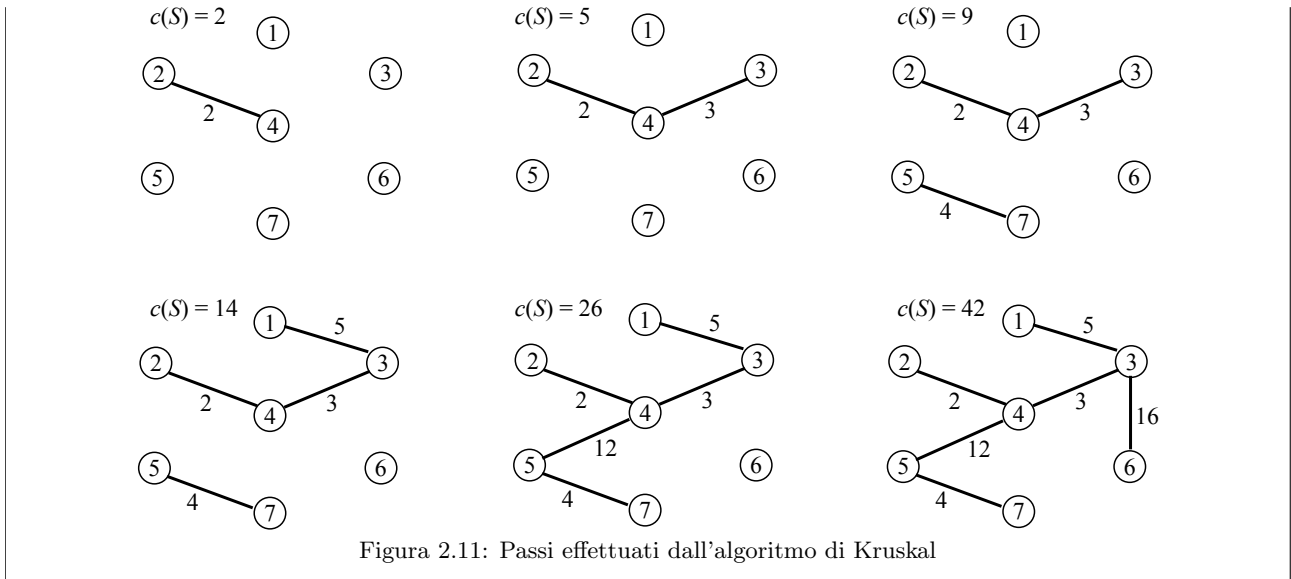
Procedura 2.4: Algoritmo di Kruskal per il problema (MST)

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 2.11 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.





Esercizio 2.15. Applicare *Kruskal* al grafo di figura 2.12; fornire ad ogni iterazione la foresta $T = (V, S)$ ed il lato esaminato indicando se viene eliminato o inserito e perché.

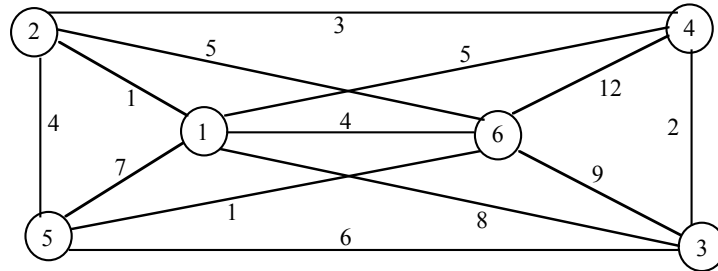


Figure 2.12: Un'istanza di (MST)

Esercizio 2.16. 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 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  $p = \text{Prim}(G, c, r)$  {
  foreach ( $i \in V$ ) do {  $p[i] = r$ ;  $d[i] = M$ ; }
   $d[r] = -M$ ;  $Q = \{r\}$ ;
  do { seleziona  $u$  in  $Q$  tale che  $d[u] = \min\{d[j] : j \in Q\}$ ;
     $d[u] = -M$ ;  $Q = Q \setminus \{u\}$ ;
    foreach ( $\{u, v\} \in S(u)$ ) do
      if ( $c_{uv} < d[v]$ ) then {  $d[v] = c_{uv}$ ;  $p[v] = u$ ;
        if ( $v \notin Q$ ) then  $Q = Q \cup \{v\}$ ;
      }
    } while ( $Q \neq \emptyset$ );
}

```

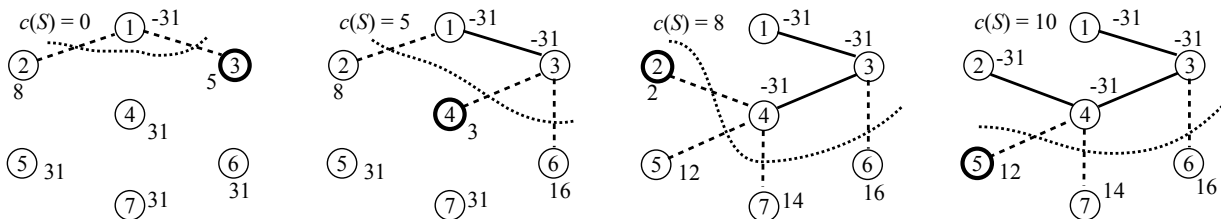
Procedura 2.5: Algoritmo di *Prim*

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 $d[i] = -M$.

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 2.13 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.



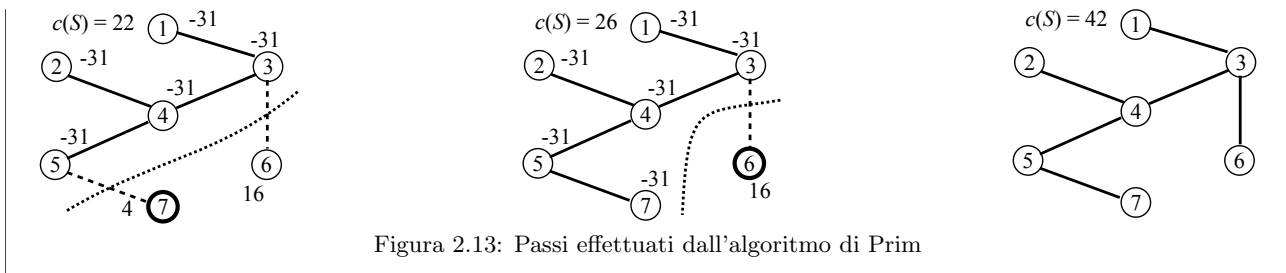


Figura 2.13: Passi effettuati dall'algoritmo di Prim

Esercizio 2.17. Applicare *Prim* al grafo di Figura 2.12; fornire ad ogni iterazione l'albero parziale $T = (V', E_T)$, l'insieme Q ed il vertice u selezionato.

Esercizio 2.18. 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 .

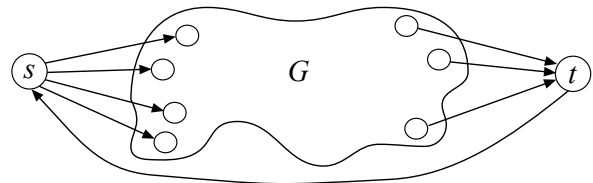
2.4 Il problema di flusso massimo

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

$$\sum_{(j,i) \in BS(i)} x_{ji} - \sum_{(i,j) \in FS(i)} x_{ij} = \begin{cases} -v & i = s \\ v & i = t \\ 0 & \text{altrimenti} \end{cases} \quad i \in N, \quad (2.9)$$

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 (2.9) 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 2.14.

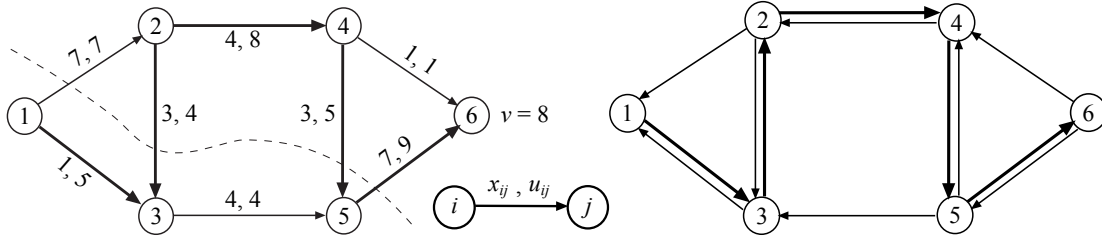
In G' i nodi sono di trasferimento, compresi s e t , cioè $b = 0$; un tale problema di flusso è detto *di circolazione*. I costi degli archi sono nulli salvo quello dell'arco (t, s) che è posto uguale a -1 : di conseguenza, minimizzare $-v$ equivale a massimizzare il valore v del flusso che transita lungo l'arco (t, s) , ossia del flusso che in G è inviato da s a t .

Figura 2.14: Un grafo con l'arco di ritorno (t, s)

Esempio 2.13. Un esempio di flusso ammissibile

Si consideri il grafo in Figura 2.15, 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.

Figura 2.15: un flusso ammissibile x di valore $v = 8$ ed il corrispondente grafo residuo

2.4.1 Tagli, cammini aumentanti e condizioni di ottimo

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

$$x_{ij}(\theta) = \begin{cases} x_{ij} + \theta & \text{se } (i, j) \in P^+ \\ x_{ij} - \theta & \text{se } (i, j) \in P^- \\ x_{ij} & \text{altrimenti} \end{cases} \quad (2.10)$$

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

$$\theta(P, x) = \min \{ \min \{ u_{ij} - x_{ij} : (i, j) \in P^+ \}, \min \{ x_{ij} : (i, j) \in P^- \} \} \quad (\geq 0), \quad (2.11)$$

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 2.15, 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 utilizzare P per inviare altre due unità di flusso da s a t , ossia costruire il nuovo flusso $x' = x \oplus 2P$; applicando la definizione (2.10) 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 2.3. 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 2.15, 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:

$$x(N_s, N_t) = \sum_{(i,j) \in A^+(N_s, N_t)} x_{ij} - \sum_{(i,j) \in A^-(N_s, N_t)} x_{ij}, \quad (2.12)$$

$$u(N_s, N_t) = \sum_{(i,j) \in A^+(N_s, N_t)} u_{ij}. \quad (2.13)$$

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 2.4. Per ogni flusso ammissibile x di valore v e per ogni taglio (N_s, N_t) vale

$$v = x(N_s, N_t) \leq u(N_s, N_t).$$

Dimostrazione La disuguaglianza deriva da (2.12), (2.13) e dal fatto che $0 \leq x_{ij} \leq u_{ij}$ per ogni $(i, j) \in A$; infatti

$$\sum_{(i,j) \in A^+(N_s, N_t)} x_{ij} \leq \sum_{(i,j) \in A^+(N_s, N_t)} u_{ij} \quad \text{e} \quad - \sum_{(i,j) \in A^-(N_s, N_t)} x_{ij} \leq 0.$$

L’uguaglianza si ottiene sommando tra loro, membro a membro, i vincoli di (2.9) 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. \diamond

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 2.19. Cercare, se esiste, un taglio (N_s, N_t) nel grafo in Figura 2.15 avente una capacità inferiore a 16.

Esercizio 2.20. Si consideri il taglio $(N_s, N_t) = (\{1, 2, 5\}, \{3, 4, 6\})$ per il grafo in Figura 2.15: si forniscano gli insiemi degli archi diretti e inversi del taglio, e si calcolino il flusso e la capacità del taglio.

Esercizio 2.21. Ripetere l’esercizio precedente per il taglio $(N_s, N_t) = (\{1, 4, 5\}, \{2, 3, 6\})$.

Il Teorema 2.4 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 2.4 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 2.5. (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  $(x, p) = \text{Cammini-Aumentanti}(G, u, s, t)$  {
  for  $(x = 0 ; ; )$  {
     $(p, \theta) = \text{Trova-Cammino}(G, s, t, x, u)$ ;
    if  $(p[t] == 0)$  then break;
     $x = \text{Aumenta-Flusso}(x, p, s, t, \theta)$ ;
  }
}

```

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 (2.10).

La terminazione dell'algoritmo è facile da mostrare sotto un'opportuna ipotesi relativa alle capacità:

Teorema 2.6. 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. \diamond

Il Teorema 2.6 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 2.22. 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 2.16. 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.

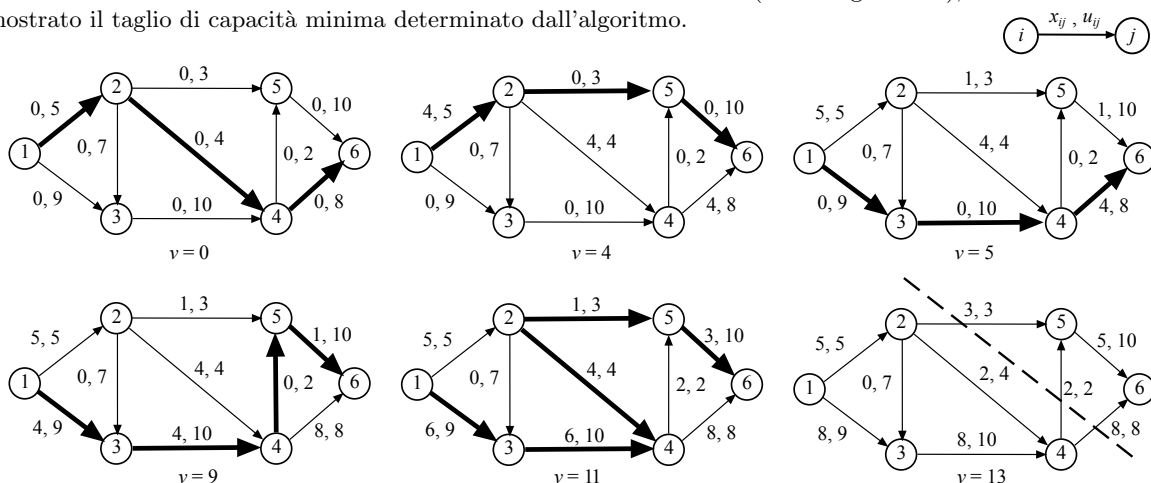


Figura 2.16: Esecuzione dell'algoritmo di Edmonds & Karp

Per l'algoritmo di Edmonds & Karp è possibile dimostrare una valutazione della complessità migliore di quella generale, di cui omettiamo la dimostrazione:

Teorema 2.7. L'algoritmo di Edmonds & Karp ha complessità $O(m^2n)$.

Si noti che questo risultato vale anche nel caso in cui le capacità non siano intere.

Esercizio 2.23. Partendo dal flusso $x = 0$, determinare il flusso massimo da 1 a 6 sul grafo in Figura 2.15, 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

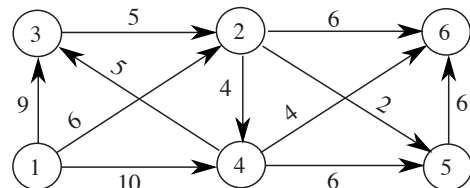
Una generalizzazione del problema di flusso massimo è quella in cui si ha un insieme S di nodi sorgente ed un insieme T di nodi pozzo (entrambi non vuoti), e si vuole individuare il massimo flusso che può essere spedito dai nodi sorgente ai nodi pozzo. Questo problema può essere facilmente risolto applicando 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 2.24. 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

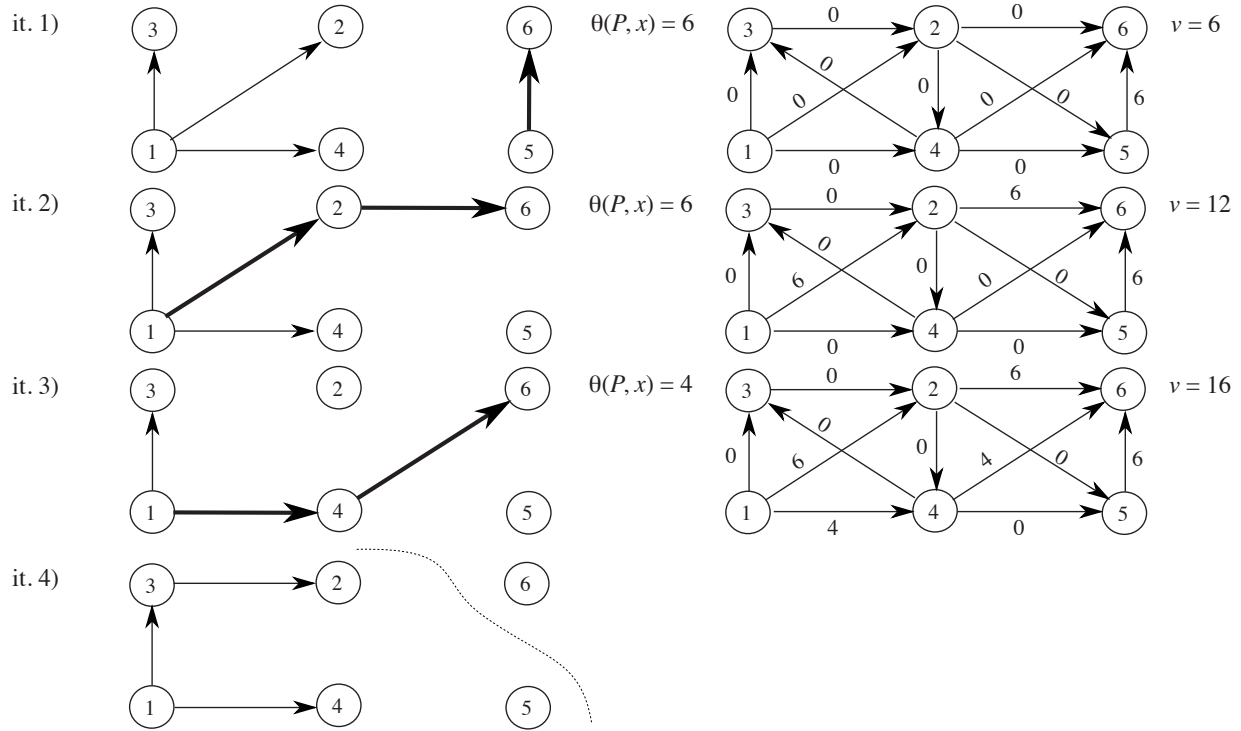
Si individui un flusso massimo dai due nodi sorgente 1 e 5 al nodo pozzo 6 sulla rete in figura, utilizzando l'algoritmo di Edmonds & Karp. Ad ogni iterazione si fornisca l'albero della visita, il cammino aumentante individuato con la relativa capacità, ed il flusso ottenuto con il relativo valore. Al 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 $u(N_s, N_t) = u_{26} + u_{46} + u_{56} = 6 + 4 + 6 = 16 = v$.



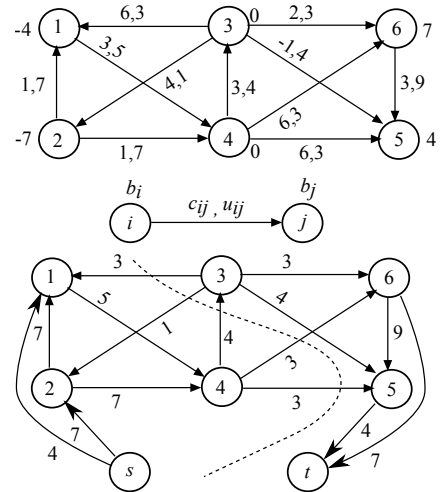
Il problema appena introdotto è particolarmente interessante anche in quanto esso coincide con il problema di determinare se esiste oppure no una soluzione ammissibile per (MCF).

Esercizio 2.25. 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 2.4. 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.



2.5 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

$$g(x) = \sum_{i \in O_x} e_x(i) \quad (= - \sum_{j \in D_x} e_x(j)) \geq 0 \quad :$$

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 (2.11), 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 (2.10), 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

$$c(P) = \sum_{(i,j) \in P^+} c_{ij} - \sum_{(i,j) \in P^-} c_{ij} . \quad (2.14)$$

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

$$cx(\theta) = c(x \oplus \theta P) = cx + \theta c(P) . \quad (2.15)$$

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 2.3:

Lemma 2.4. 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.

È possibile dimostrare (ma non lo faremo formalmente) che cammini e cicli aumentanti sono gli “strumenti base per costruire pseudoflussi, e quindi flussi”.

Teorema 2.8. [*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, \dots, 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 $i = 1, \dots, k$, $x^{k+1} = x''$, 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.

Un elemento importante della tesi del Teorema 2.8, 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.

Il Teorema 2.8 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 2.1. 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 (2.15)).

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 2.8 si ha $x' = x \oplus \theta_1 C_1 \oplus \dots \oplus \theta_k C_k$, dove C_i sono cicli aumentanti rispetto a x ; siccome tutti i θ_i sono numeri positivi, $cx' < cx$ e (2.15) implicano che $c(C_i) < 0$ per un qualche i , il che contraddice l'ipotesi. \diamond

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 2.1 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  $(x, caso) = \text{Cancella-Cicli}(G, c, b, u)$  {
   $(x, caso) = \text{Flusso-Ammissibile}(G, b, u)$ ;
  if ( $caso == \text{“vuoto”}$ ) then return;
  for ( ; ) {
     $(C, \theta) = \text{Trova-Ciclo}(G, c, u, x)$ ;
    if ( $\theta == 0$ ) then break;
     $x = \text{Cambia-Flusso}(x, C, \theta)$ ;
  }
}

```

Procedura 2.7: Algoritmo basato sulla cancellazione di cicli

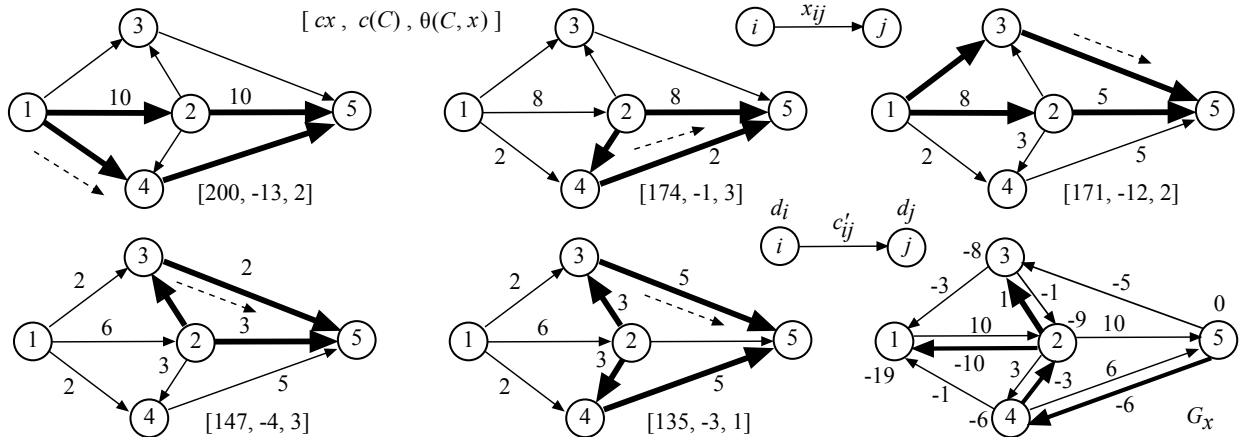
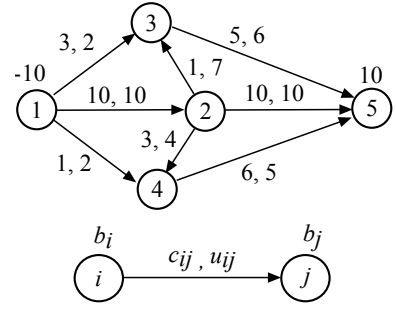
La procedura *Flusso-Ammissibile* determina, se esiste, un flusso ammissibile: in tal caso restituisce $caso = \text{“ottimo”}$ ed il flusso x , altrimenti restituisce $caso = \text{“vuoto”}$. Una possibile implementazione di questa procedura è stata discussa nel §2.4.3. La procedura *Trova-Ciclo* determina, 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 2.4 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 2.26. 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 2.20. 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 2.17. 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 2.17: Esecuzione dell'algoritmo *Cancella-Cicli*

La correttezza dell'algoritmo *Cancella-Cicli* discende direttamente dal Corollario 2.1; 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 2.6): 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 2.27. 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.

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 2.9. 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 Simplexso su Reti”, perché si possono interpretare come versioni specializzate (altamente efficienti) degli algoritmi del simplexso 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 2.10. 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 2.8 mostra che esistono k cammini aumentanti P_1, \dots, 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, \dots, C_h rispetto ad x tali che

$$x' = x \oplus \theta_1 P_1 \oplus \dots \oplus \theta_k P_k \oplus \theta_{k+1} C_1 \oplus \dots \oplus \theta_{k+h} C_h .$$

In più, deve sicuramente essere $\theta_1 + \dots + \theta_k = \theta$. 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 $c(P) \leq c(P_i)$, $i = 1, \dots, k$. Di conseguenza

$$cx' = cx + \theta_1 c(P_1) + \dots + \theta_k c(P_k) + \theta_{k+1} c(C_1) + \dots + \theta_{k+h} c(C_h) \geq cx + \theta c(P) = cx(\theta) ,$$

e quindi $x(\theta)$ è minimale. \diamond

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

$$\theta = \min\{ \theta(P, x), e_x(s), -e_x(t) \} > 0 \quad (2.16)$$

(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( $G, c, b, u$ ) {
   $x = \text{Inizializza}(c, u)$ ;  $caso = \text{"ottimo"}$ ;
  while ( $g(x) \neq 0$ ) do
    ( $p, d$ ) = Trova-Cammino-Minimo( $G_x, O_x, D_x$ );
     $t = \text{argmin} \{ d[i] : i \in D_x \}$ ;
    if ( $d[t] == \infty$ ) then {  $caso = \text{"vuoto"}$ ; break; }
     $x = \text{Aumenta-Flusso}(x, p, t)$ ;
  }
}

```

Procedura 2.8: *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 $caso = \text{"vuoto"}$, in quanto non esiste nessuna soluzione ammissibile per il problema di flusso di costo minimo.

Esercizio 2.28. 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 (2.16), 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 2.10 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 2.11. 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.

Esempio 2.21. Esecuzione dell'algoritmo *Cammini-Minimi-Successivi*

Consideriamo di nuovo l'istanza del problema (MCF) dell'Esempio 2.20; il funzionamento dell'algoritmo basato su cammini minimi successivi è mostrato in Figura 2.18. 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.

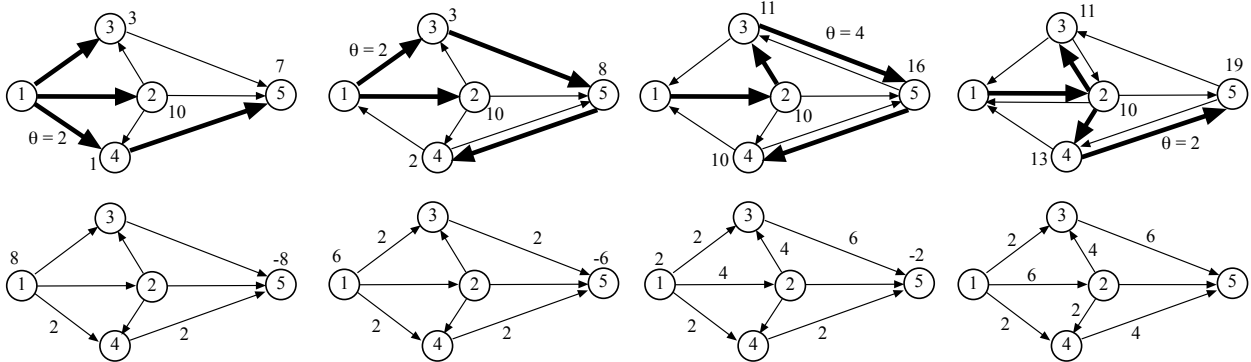


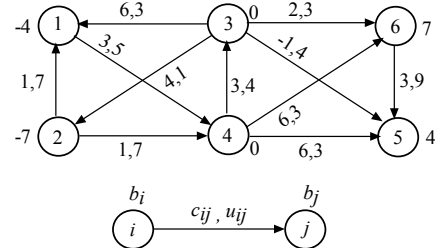
Figura 2.18: Esecuzione dell'algoritmo basato su cammini minimi successivi

Il prossimo esempio di esecuzione dell'algoritmo *Cammini-Minimi-Successivi* mostra invece cosa accade quando l'istanza di (MCF) non ha soluzione ammissibile.

Esempio 2.22. *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 2.19. 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.

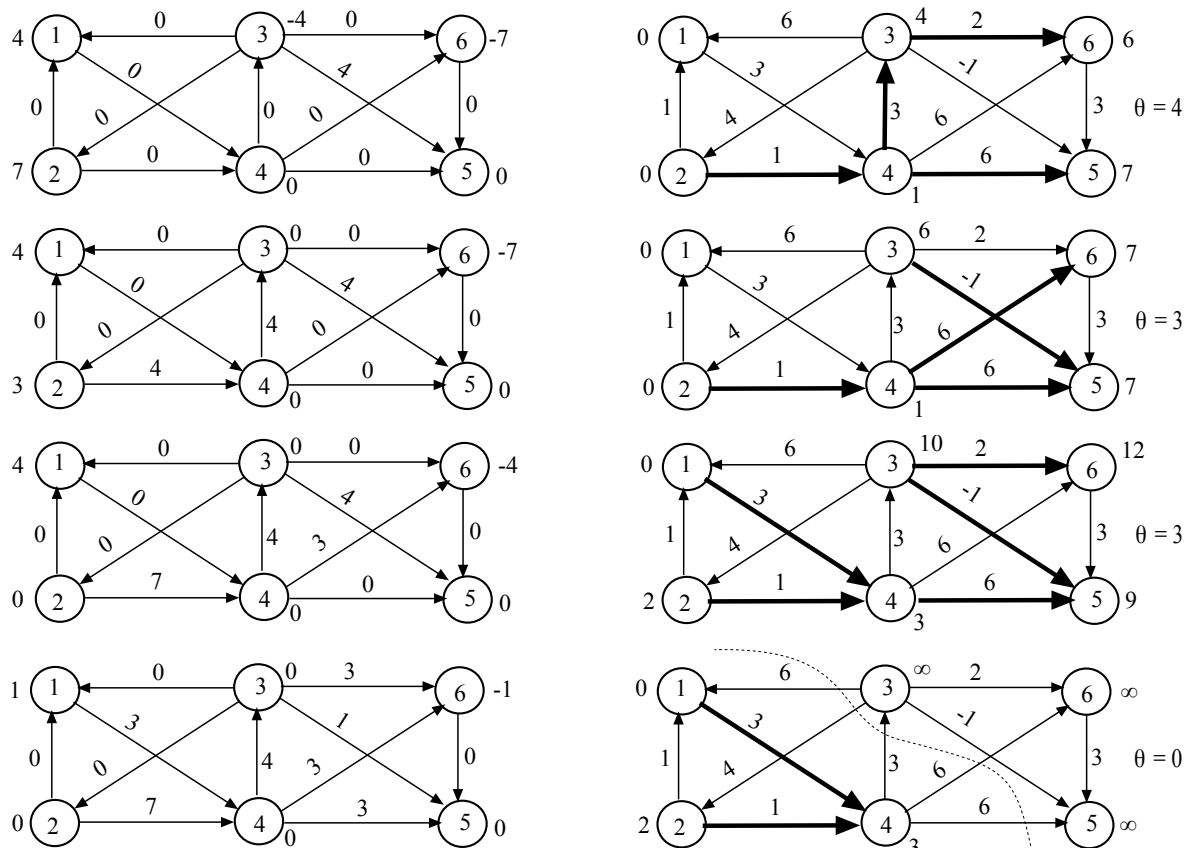


Figura 2.19: Esecuzione dell’algoritmo basato su cammini minimi successivi

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 $\bar{g} = \sum_{(i,j) \in A: c_{ij} < 0} u_{ij} + \sum_{i \in N: b_i > 0} b_i$; siccome $g(x)$ diminuisce di almeno un’unità ad ogni iterazione, il numero di iterazioni non potrà eccedere \bar{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 $O(\bar{g}mn)$.

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* $c_{ij}^\pi = c_{ij} + \pi_i - \pi_j$ di ogni arco (i, j) rispetto a π .

Teorema 2.12. Uno pseudoflusso x è minimale se e solo se esiste un vettore di potenziali π tale che

$$\begin{aligned} x_{ij} > 0 & \implies c_{ij}^\pi \leq 0 \\ x_{ij} < u_{ij} & \implies c_{ij}^\pi \geq 0 \end{aligned} \quad (2.17)$$

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 (2.17) (all'inizio dell'iterazione), allora anche $x \oplus \theta P$ e $\pi + d$ le rispettano (alla fine dell'iterazione). Pertanto, anche per questa variante dell'algoritmo si ottiene, tramite il Teorema 2.12, 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 $O(m \log n)$. 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 $O = \{1, \dots, n\}$ è l'insieme dei *nodi origine*, $D = \{n+1, \dots, n+d\}$ 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 2.20. 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 .

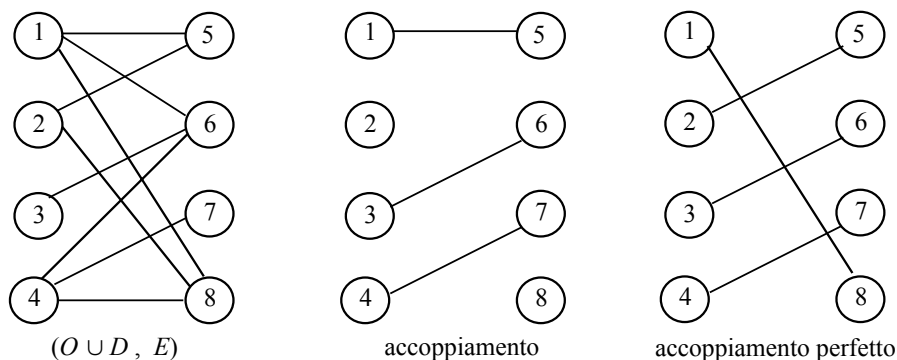


Figura 2.20: Esempi di accoppiamenti

Nel seguito studieremo i seguenti problemi:

- *Accoppiamento di Massima Cardinalità* in G .

- *Assegnamento di Costo Minimo*: si vuole determinare, tra tutti gli accoppiamenti perfetti in G , uno che abbia di costo 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 2.21, relativo al grafo G di Figura 2.20, 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).

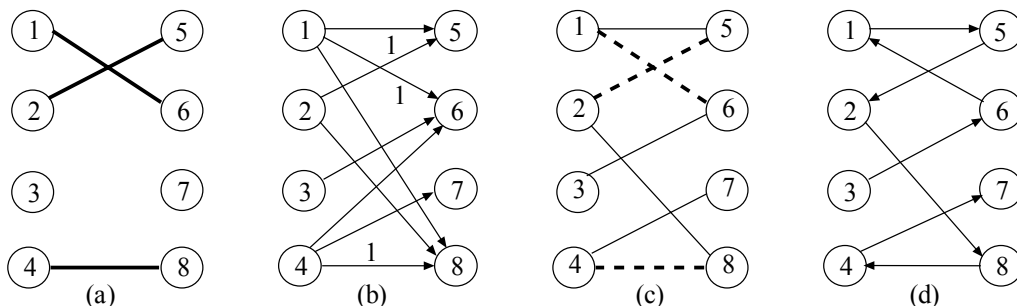


Figura 2.21: Flussi, accoppiamenti e cammini (alternanti) aumentanti

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 cam-

mini 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 2.23. Cammini alternanti aumentanti

Le affermazioni precedenti possono essere facilmente verificate nell'esempio in Figura 2.21; 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 2.24. Operazione di composizione

Proseguendo l'esempio di Figura 2.21, ed applicando l'operazione di composizione sull'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 verificare 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  $M = \text{Accoppiamento-MaxCard}(O, D, E)$  {
   $M = \emptyset$ ;
  while ( $O_M \neq \emptyset$ ) do {
     $p = \text{Cammino-Aumentante}(O, D, E, M)$ ;
    if ( $p[i] == 0$  per ogni  $i \in D_M$ ) then break;
    seleziona  $t \in D_M$  tale che  $p[t] \neq 0$ ;
     $M = \text{Cambia-Accoppiamento}(M, p, t)$ ;
  }
}

```

Procedura 2.9: 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.

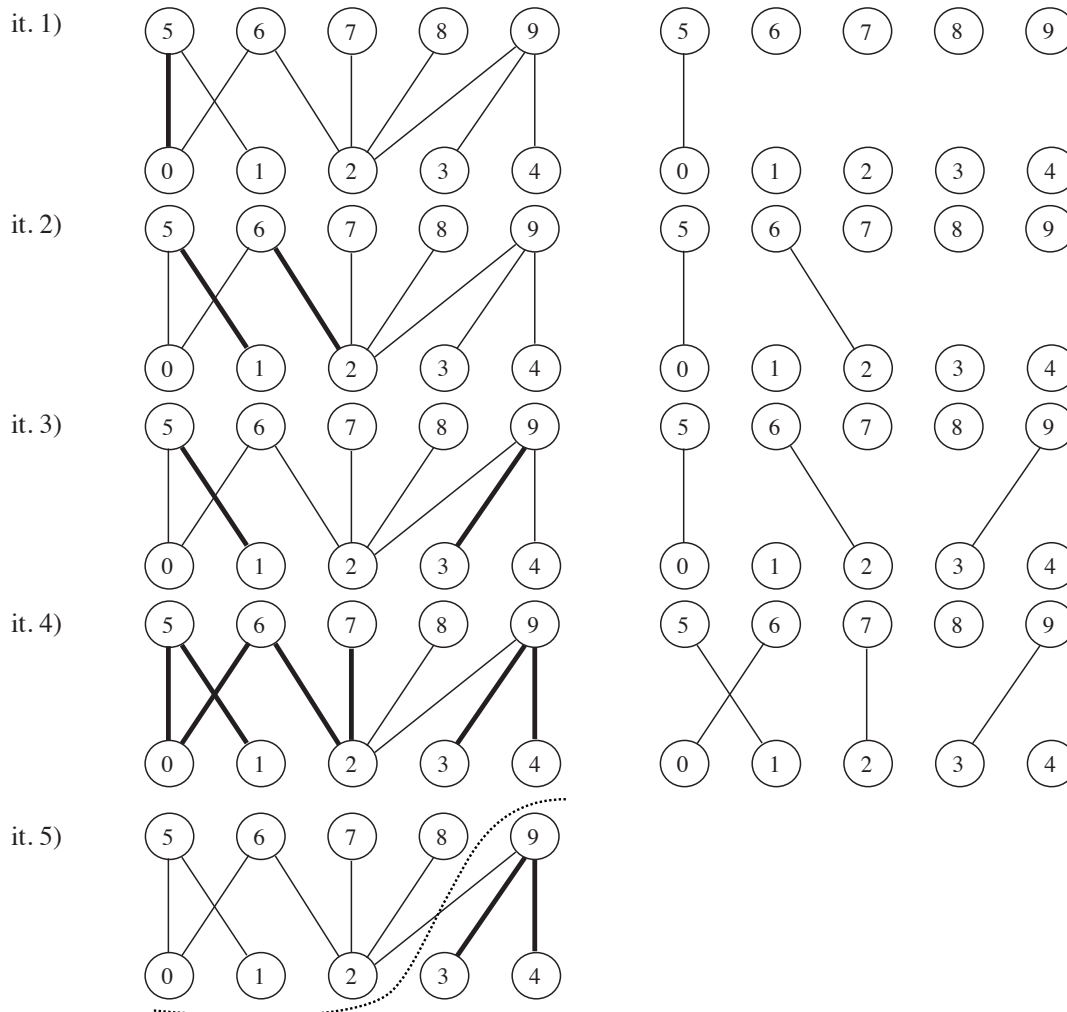
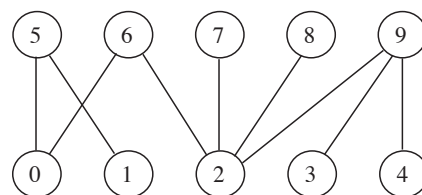
Esercizio 2.29. Si fornisca una descrizione formale, in pseudo-codice, delle procedure *Cammino-Aumentante* e *Cambia-Accoppiamento*.

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 2.25. Esecuzione di *Accoppiamento-MaxCard*

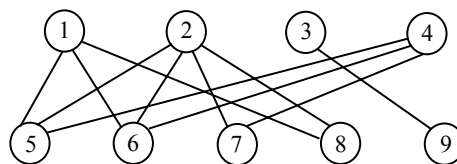
Si vuole risolvere il problema (MCM) per il grafo in figura mediante l'algoritmo dei cammini alternanti aumentanti che effettua una visita in ampiezza (ossia l'algoritmo di Edmonds&Karp). Durante la visita si esaminerà la stella uscente del vertice selezionato in ordine crescente dell'altro estremo del 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.



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 2.30. 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 2.22 (b) per l'istanza in (a) (per chiarezza di visualizzazione non sono indicate le capacità degli archi, tutte pari a 1).

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 (2.14). 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 $C(M') = C(M) + C(P)$.

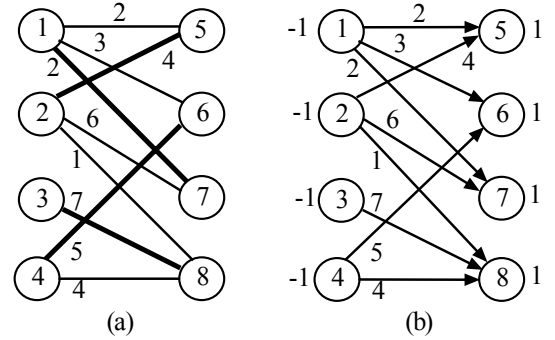
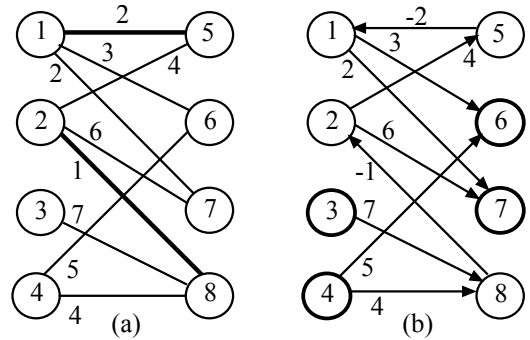


Figura 2.22: Trasformazione in un (MCF)

Esempio 2.26. 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  $M = \text{Assegnamento-MinCost}(O, D, E, c)$   $\{$ 
   $M = \emptyset;$ 
  while  $(O_M \neq \emptyset)$  do  $\{$ 
     $(p, d) = \text{Cammino-AA-Minimo}(G_M);$ 
     $t = \text{argmin} \{d[i] : i \in D_M\};$ 
    if  $(d[t] == \infty)$  then break;
     $M = \text{Cambia-Accoppiamento}(M, p, t);$ 
   $\}$ 
 $\}$ 

```

Procedura 2.10: 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 un 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 2.31. Dimostrare l'affermazione precedente.

Esempio 2.27. Due iterazioni di *Assegnamento-MinCost*

In Figura 2.23 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 2.23(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 2.23(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 2.22(a).

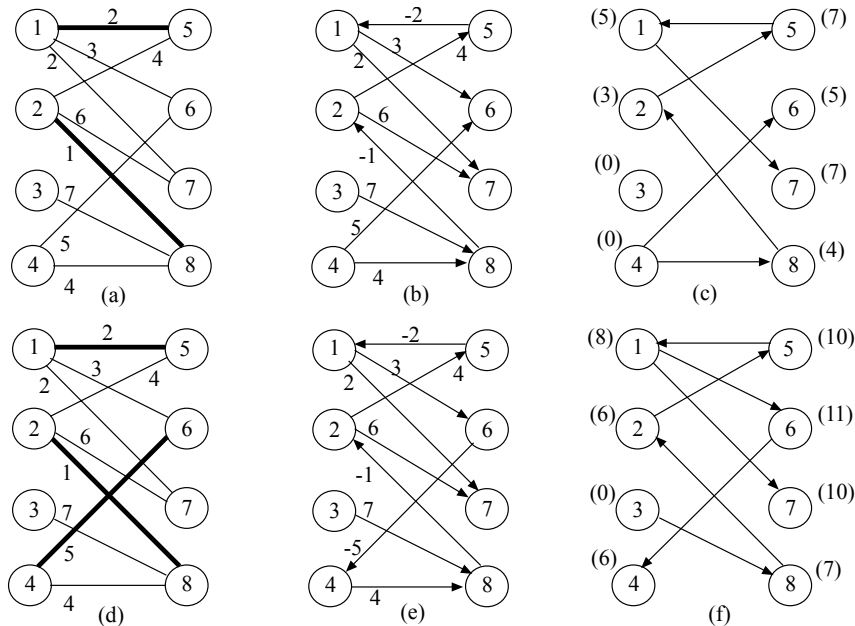
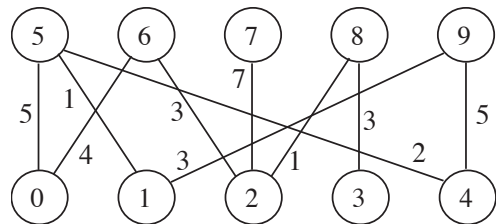


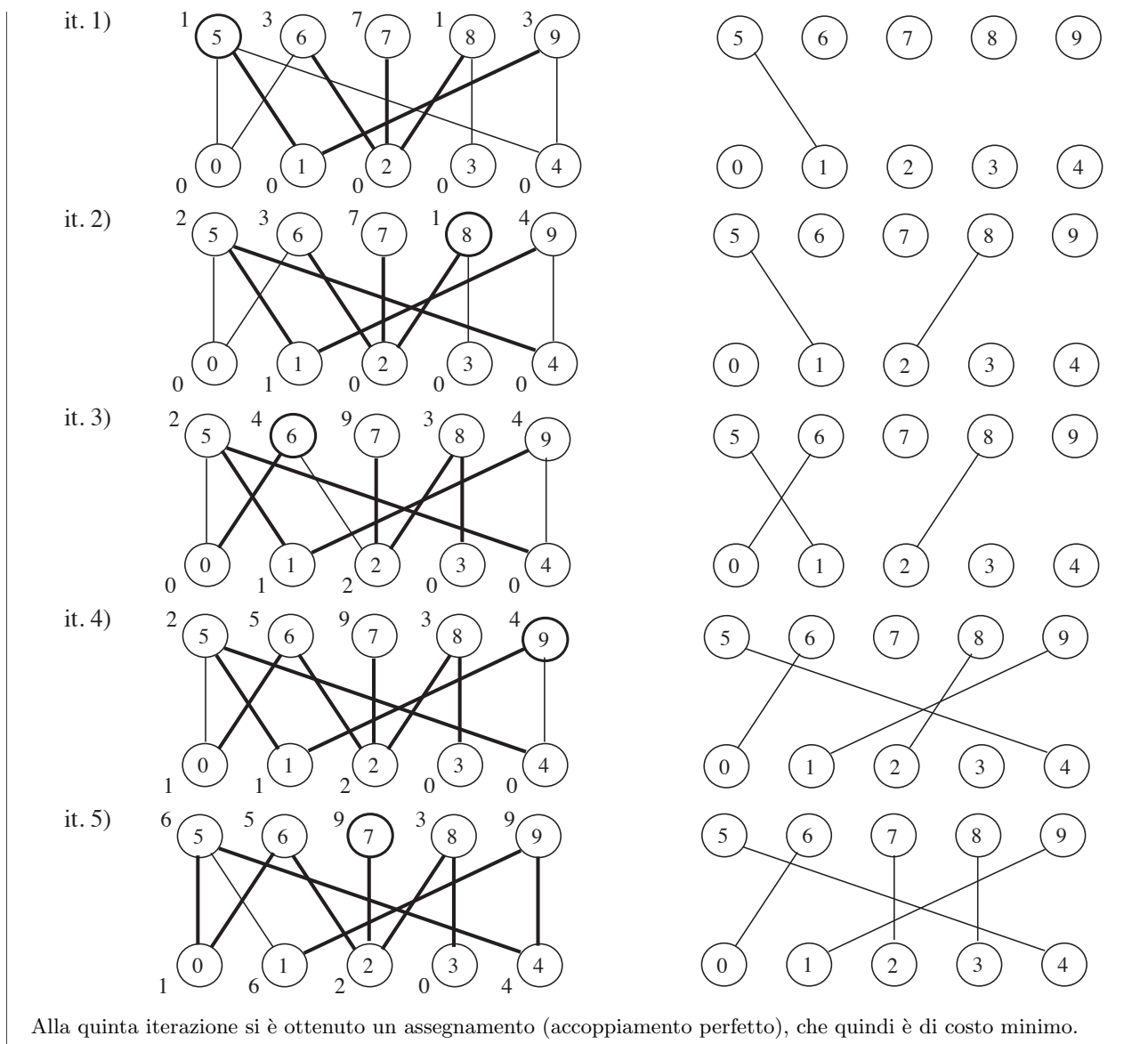
Figura 2.23: Due iterazioni di *Assegnamento-MinCost*

Esempio 2.28. 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.





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.

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 5) 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 $c(x) : \mathbb{R}^n \rightarrow \mathbb{R}$ 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 $\alpha, \beta \in \mathbb{R}$; 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

$$\max \{ cx : Ax \leq b \} \quad (3.1)$$

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 $n = 2$, $m = 5$,

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 2 & 1 \\ -1 & 0 \\ 0 & -1 \end{bmatrix}, \quad b = \begin{bmatrix} 400000 \\ 700000 \\ 900000 \\ 0 \\ 0 \end{bmatrix}, \quad c = [500, 200] .$$

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:

$$\begin{aligned} i) \quad \max \sum_{j=1}^n c_j x_j &\equiv -\min \sum_{j=1}^n (-c_j) x_j \\ ii) \quad \sum_{j=1}^n a_{ij} x_j = b_i &\equiv \begin{cases} \sum_{j=1}^n a_{ij} x_j \leq b_i \\ \sum_{j=1}^n a_{ij} x_j \geq b_i \end{cases} \\ iii) \quad \sum_{j=1}^n a_{ij} x_j \geq b_i &\equiv \sum_{j=1}^n (-a_{ij}) x_j \leq -b_i \\ iv) \quad \sum_{j=1}^n a_{ij} x_j \geq b_i &\equiv \sum_{j=1}^n a_{ij} x_j - s_i = b_i, \quad s_i \geq 0 \\ v) \quad \sum_{j=1}^n a_{ij} x_j \leq b_i &\equiv \sum_{j=1}^n a_{ij} x_j + s_i = b_i, \quad s_i \geq 0 \end{aligned} \quad (3.2)$$

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

$$x_i = x_i^+ - x_i^- \quad , \quad x_i^+ \geq 0 \quad , \quad x_i^- \geq 0 . \quad (3.3)$$

Esempio 3.2. Trasformazioni equivalenti

Il problema della Fonderia dell'Esempio 1.4

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

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

$$\begin{array}{rcllclclclcl}
-\max & -0.025x_1 & - & 0.030x_2 & - & 0.018x_3 & - & 10x_4 & & \\
& -4x_1 & - & x_2 & - & 0.6x_3 & & & \leq & -3250 \\
& 4x_1 & + & x_2 & + & 0.6x_3 & & & \leq & 5500 \\
& -0.45x_1 & - & 0.5x_2 & - & 0.4x_3 & - & 100x_4 & \leq & -450 \\
& x_1 & + & x_2 & + & x_3 & + & x_4 & \leq & 1000 \\
& -x_1 & - & x_2 & - & x_3 & - & x_4 & \leq & -1000 \\
& -x_1 & & & & & & & \leq & 0 \\
& & & -x_2 & & & & & \leq & 0 \\
& & & & & -x_3 & & & \leq & 0 \\
& & & & & & & -x_4 & \leq & 0.
\end{array}$$

Si ottiene quindi

$$c = [-0.025 \quad -0.030 \quad -0.018 \quad -10]$$

$$A = \begin{bmatrix} -4 & -1 & -0.6 & 0 \\ 4 & 1 & 0.6 & 0 \\ -0.45 & -0.5 & -0.4 & -100 \\ 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 \\ -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}, \quad b = \begin{bmatrix} -3250 \\ 5500 \\ -450 \\ 1000 \\ -1000 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

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 alternativamete utilizzare come forma standard la seguente:

$$\min\{ cx : Ax = b, x \geq 0 \} . \quad (3.4)$$

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

$$A = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 & 1 \end{bmatrix}, \quad b = \begin{bmatrix} 400000 \\ 700000 \\ 900000 \end{bmatrix}, \quad c = [-500, -200] .$$

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 facce giocano un ruolo particolarmente importante nella PL: dimostreremo infatti che, sotto oppor-

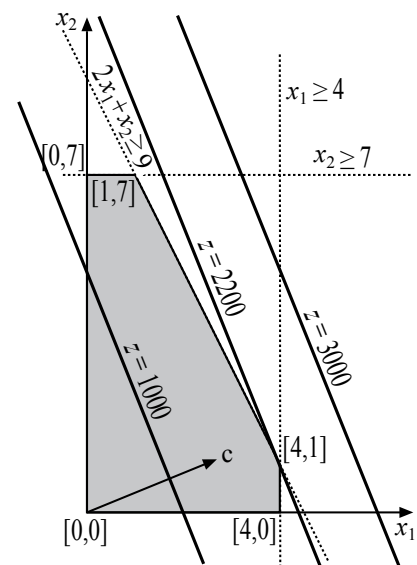


Figura 3.1: Il problema della Pintel

tune 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 la regione ammissibile è vuota.

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, spesso limitandoci ad una presentazione priva delle dimostrazioni formali, 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.

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

$$\|x\| = \sqrt{xx} = \sqrt{\sum_{i=1}^n x_i^2}$$

si ha

$$ax = \|a\| \|x\| \cos(\theta)$$

dove θ è l'angolo formato dai due vettori. Ciò significa che il prodotto scalare:

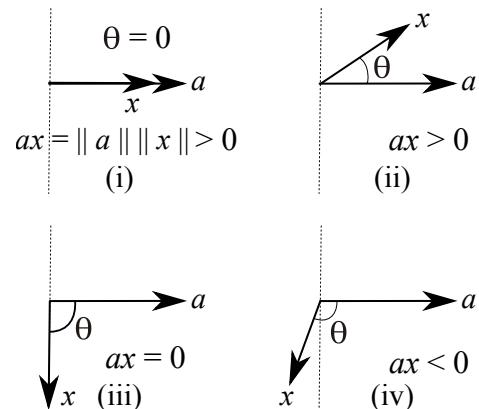


Figura 3.2: Geometria del prodotto scalare

- 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).

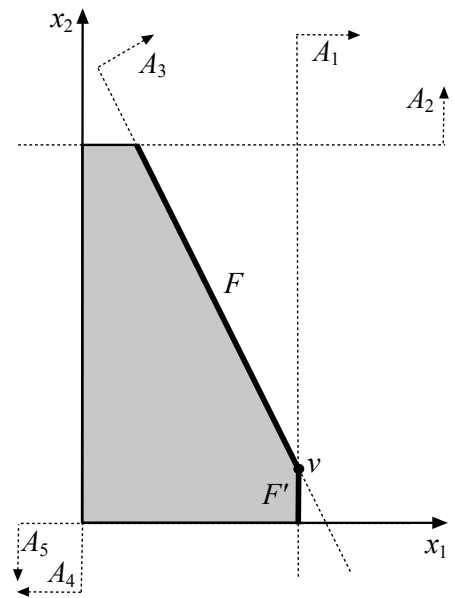
Possiamo adesso caratterizzare geometricamente i vincoli lineari, ossia ottenuti attraverso funzioni lineari. Nello spazio vettoriale \mathbb{R}^n , un vincolo lineare del tipo $A_i x = b_i$ individua l'*iperpiano affine* $P_i = \{x : A_i x = b_i\}$, che costituisce la *frontiera* del *semispazio affine* $S_i = \{x : A_i x \leq b_i\}$ 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 $A_i x \leq b_i$, $i = 1, \dots, 5$ 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 per cui $A_i x = b_i$, 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 $\text{conv}(x, y) = \{z = \alpha x + (1 - \alpha)y : \forall \alpha \in [0, 1]\}$, è formato solamente da punti di C : $\text{conv}(x, y) \subseteq C$. 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 $X = \{x^1, \dots, x^s\} \subset \mathbb{R}^n$

$$\text{conv}(X) = \left\{ x = \sum_{i=1}^s \lambda_i x^i : \sum_{i=1}^s \lambda_i = 1, \lambda_i \geq 0 \quad i = 1, \dots, s \right\} ;$$

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:

$$\begin{array}{rclcl} -x_1 & & \leq & 0 & (1) \\ & x_2 & \leq & 1 & (2) \\ & & -x_3 & \leq & 0 & (3) \\ -x_2 & & \leq & 0 & (4) \\ & x_3 & \leq & 1 & (5) \\ x_2 + x_3 & \leq & 2 & (6) \end{array} , \quad A = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} , \quad b = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 2 \end{bmatrix} .$$

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.

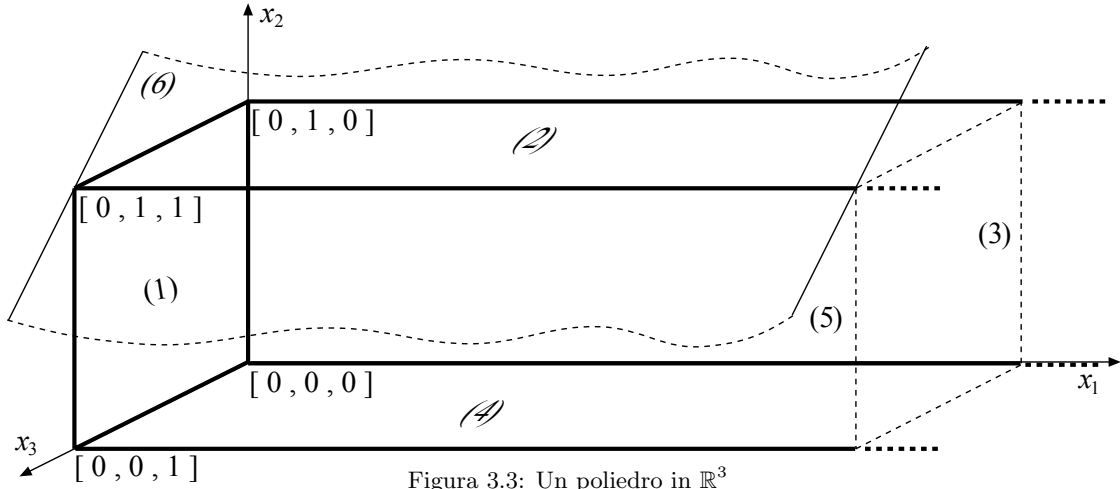


Figura 3.3: Un poliedro in \mathbb{R}^3

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*

$$x = \alpha x' + (1 - \alpha)x'' \quad , \quad \alpha \in [0, 1] \quad , \quad x' \in P \quad , \quad x'' \in P \quad \implies \quad x' = x'' \quad ,$$

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: $[0, 0, 0]$, $[0, 1, 0]$, $[0, 0, 1]$ e $[0, 1, 1]$. 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

$$\forall x \in P \quad \forall \lambda \geq 0 \quad (x + \lambda d) \in P \quad .$$

In particolare, il poliedro di Figura 3.3 ha una sola direzione, $[1, 0, 0]$, che “identifica” i quattro spigoli illimitati.

Al fine di caratterizzare matematicamente le direzioni studiamo adesso la “forma particolarmente semplice” di poliedro

$$C = \{x : Ax \leq 0\} \quad (3.5)$$

in cui il lato destro dei vincoli è tutto nullo. Ciò implica immediatamente

$$x \in C \quad , \quad \alpha \geq 0 \quad \implies \quad \alpha x \in C \quad ; \quad (3.6)$$

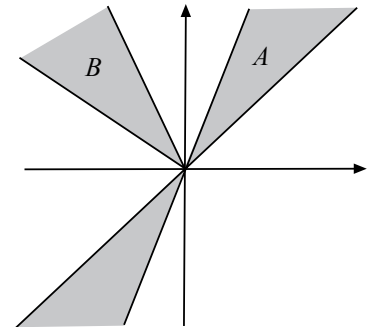


Figura 3.4: Coni convessi e non

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*.

È facile verificare che una definizione alternativa a (3.6), per il caso convesso, è

$$x, y \in C \quad , \quad \lambda, \mu \in \mathbb{R}_+ \quad \implies \quad \lambda x + \mu y \in C \quad . \quad (3.7)$$

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 $V = \{v^1, \dots, v^t\} \subset \mathbb{R}^n$ come

$$\text{cono}(V) = \left\{ v = \sum_{i=1}^t \nu_i v^i : \nu_i \geq 0 \quad i = 1, \dots, t \right\} . \quad (3.8)$$

Possiamo adesso enunciare (senza dimostrazione) la caratterizzazione dell'insieme $\text{rec}(P)$ di tutte le direzioni (di recessione) di P .

Teorema 3.1. $\text{rec}(P) = \{d \in \mathbb{R}^n : Ad \leq 0\}$.

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 $\text{rec}(P)$, anche se la proprietà era già ovvia dalla definizione. Si noti che $\text{rec}(P)$ non è mai vuoto: chiaramente, 0 vi appartiene sempre. D'altro canto, 0 non è una direzione; quindi, si ha che o $\text{rec}(P) = \{0\}$, oppure esiste un $0 \neq d \in \text{rec}(P)$. 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 $\text{rec}(P) = \{0\}$. Infatti, se $0 \neq d \in \text{rec}(P)$ 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

$$d \in \text{rec}(P) \quad , \quad -d \in \text{rec}(P) \quad ;$$

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 $[1, 0, 0]$ 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,

$$\text{lin}(P) = \{d \in \mathbb{R}^n : Ad = 0\}$$

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.

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 $d \in \text{rec}(P)$, 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 $d \in \text{rec}(P)$ 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 $\text{rec}(P)$, invece che nella forma (3.5), fosse dato nella forma (3.8), ossia se si conoscesse un insieme finito di vettori V tale che $\text{rec}(P) = \text{cono}(V)$, in quanto vale il seguente risultato di cui omettiamo la facile dimostrazione:

Lemma 3.1. Esiste $d \in C = \text{cono}(V)$ tale che $cd > 0$ se e solo se esiste $v_i \in V$ tale che $cv_i > 0$.

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). Questo è in effetti possibile, in entrambe le direzioni, anche se noi ci limiteremo a mostrare nel dettaglio una sola delle trasformazioni in quanto è basata su un principio algoritmico interessante.

Teorema 3.2. Dato $C = \text{cono}(V)$, esiste una matrice A per cui $C = \{x : Ax \leq 0\}$.

Dimostrazione È chiaro che $\text{cono}(V)$ definito da (3.8) è un cono poliedrico nello 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, \dots, x_n . Per fare ciò si esamina il generico vincolo i -esimo, $i = 1, \dots, m$,

$$\sum_{j=1}^n a_{ij}x_j \leq b_i$$

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

$$\begin{aligned} i \in I^+ &\implies x_1 \leq (b_i - \sum_{j=2}^n a_{ij}x_j) / a_{i1} \\ i \in I^- &\implies x_1 \geq (b_i - \sum_{j=2}^n a_{ij}x_j) / a_{i1} \end{aligned}$$

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

$$(b_h - \sum_{j=2}^n a_{hj}x_j) / a_{h1} \leq (b_i - \sum_{j=2}^n a_{ij}x_j) / a_{i1} \quad (3.9)$$

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. \diamond

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

$$\mathcal{C} = \text{cono} \left(\left\{ \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} \right\} \right)$$

come un cono poliedrico. Seguendo la dimostrazione del Teorema 3.2, la definizione (3.8) da

$$\mathcal{C} = \left\{ \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \alpha_1 \begin{bmatrix} -1 \\ 0 \\ 0 \end{bmatrix} + \alpha_2 \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + \alpha_3 \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} + \alpha_4 \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}, [\alpha_1, \dots, \alpha_4] \geq 0 \right\}$$

In altri termini, nello spazio esteso $[x, \alpha]$ il cono è definito dal seguente insieme di equazioni

$$\{x_1 = -\alpha_1 - \alpha_3 + \alpha_4, \quad x_2 = \alpha_2 - \alpha_3 + \alpha_4, \quad x_3 = \alpha_3 - \alpha_4\}$$

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

$$\{x_1 = \alpha_1 - x_3, \quad x_2 = \alpha_2 - x_3, \quad \alpha_1 \geq 0, \quad \alpha_2 \geq 0\}$$

che è facilmente riscrivibile come

$$\mathcal{C} = \left\{ \begin{array}{ccc} x_1 & + & x_3 \leq 0 \\ & - & x_3 \leq 0 \end{array} \right.$$

(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 disequazioni 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.

È possibile dimostrare la relazione inversa, ossia che qualsiasi cono poliedrico può essere espresso come cono finitamente generato.

Teorema 3.3. Dato $C = \{x \in \mathbb{R}^n : Ax \leq 0\}$, esiste un insieme finito $V \subset \mathbb{R}^n$ per cui $C = \text{cono}(V)$. 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: *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 $X = \{x^1, \dots, x^s\} \subset \mathbb{R}^n$ e $V = \{v^1, \dots, v^t\} \subset \mathbb{R}^n$ per cui $P = \text{conv}(X) + \text{cono}(V)$. L'insieme $\text{cono}(V)$ coincide con $\text{rec}(P)$. In più, se P non contiene direzioni di linealità ($\text{lin}(P) = \{0\}$), allora una rappresentazione minimale di X contiene tutti e soli i punti estremi di P .

Il significato geometrico del Teorema 3.4 è illustrato nella Figura 3.5, dove il poliedro P è dato dalla somma del triangolo $Q = \text{conv}(\{x^1, x^2, x^3\})$ con il cono $C = \text{cono}(\{v^1, v^2\})$. Si noti che nel caso in cui $\text{rec}(P) = \{0\}$ si ha $P = \text{conv}(X)$. Viceversa, se P è un cono e $\text{rango}(A) = n$, ossia $\text{lin}(P) = \{0\}$, allora ha il solo vertice 0: quindi, $X = \{0\}$ e $P = \text{cono}(V) = \text{rec}(P)$.

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 $Q = \text{conv}(X)$ è il “lato sinistro” del poliedro, mentre $C = \text{cono}(V)$ è il semiasse x_1 positivo, che è parallelo a tutti e quattro gli spigoli illimitati di P .

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 $\text{rec}(P) = \{0\}$, oppure il fatto che $\text{conv}(X)$ è il più piccolo poliedro che contiene tutti i punti di X . Dal punto di vista della PL, la proprietà più rilevante (di cui omettiamo la facile dimostrazione) è la seguente:

Corollario 3.1. Sia $P = \{x \in \mathbb{R}^n : Ax \leq b\} = \text{conv}(X) + \text{cono}(V) \neq \emptyset$: allora il problema (3.1) ha ottimo finito se e solo se $cv^j \leq 0$ per $j = 1, \dots, t$, ed in questo caso esiste un $h \in \{1, \dots, s\}$ tale che x^h è una soluzione ottima del problema.

Se $\text{rango}(A) = n$, 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.

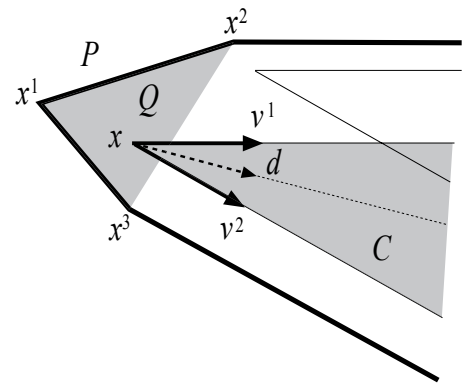


Figura 3.5: Decomposizione di un poliedro

Esempio 3.7. 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 $\text{rango}(A) < n$, ossia $\text{lin}(P) \neq \{0\}$, 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 $\text{lin}(P) \neq \{0\}$, ossia $\text{rango}(A) < n$, 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 $\text{rango}(A) < n$, 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

$$\max \{c'x' : A'x' \leq b\} . \quad (3.10)$$

ossia rimuovendo l'ultima colonna di A e la corrispondente variabile. Per verificare questo notiamo che a qualsiasi soluzione ammissibile x' di (3.10) 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 (3.10). Da questo segue che:

- se (3.10) non ha soluzione ammissibile allora non può averla nemmeno (3.1);
- se (3.10) 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 (3.10). 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 (3.10) 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 (3.10) 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 (3.10) corrisponde una soluzione $x = [x', 0]$ di (3.1) con lo stesso costo. Pertanto, $\bar{x} = [\bar{x}', 0]$ è ottima per (3.1). \diamond

Esempio 3.8. 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 $\text{rango}(A') = 2$). 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.

Esempio 3.9. Diversa dimensionalità delle rappresentazioni

Si consideri l'*ipercubo unitario* in \mathbb{R}^n

$$H = \{ x \in \mathbb{R}^n : -1 \leq x_i \leq 1 \quad i = 1, \dots, n \}$$

che è caratterizzato da $2n$ vincoli. È facile verificare che H ha 2^n punti estremi. Per questo basta considerare che H è il prodotto Cartesiano degli n intervalli $[-1, 1]^n$ per ciascuna delle variabili; in altri termini le variabili sono indipendenti (il valore scelto per la variabile x_i non ha alcun impatto su come possa essere scelto il valore della variabile x_j per $j \neq i$). Ovviamente, i punti estremi di ciascun intervallo sono -1 e 1 . È quindi facile verificare, e sarà confermato algebricamente nell'Esempio 3.13, che i vertici di H si ottengono componendo liberamente i vertici degli intervalli per ciascuna variabile; in altri termini sono tutti i 2^n vettori di $\{-1, 1\}^n$, ossia nella forma $x = [\pm 1, \pm 1, \dots, \pm 1]$ corrispondente a tutti i modi possibili per scegliere -1 oppure 1 nelle n posizioni del vettore x . Quindi, H ha $2n$ faccette (è ovvio che ciascuno dei vincoli è non ridondante) ma 2^n vertici.

Spesso ci si riferisce all'*ipercubo unitario* come il poliedro definito invece dai vincoli $0 \leq x_i \leq 1$, $i = 1, \dots, n$, i cui vertici sono tutti i punti di $\{0, 1\}^n$; è ovvio come le due definizioni siano sostanzialmente equivalenti.

Un esempio opposto è quello dell'*n-co-cubo*, ossia l'insieme

$$B_1 = \{ x \in \mathbb{R}^n : \sum_{i=1}^n |x_i| \leq 1 \}$$

(la sfera unitaria nella norma L_1). Questo insieme, nonostante sia espresso in termini della funzione non lineare $|\cdot|$, è un poliedro caratterizzato da “pochi” vertici e “molte” faccette.

Esempio 3.10. La proiezione dell'*n-co-cubo*

Con la tecnica vista nell'Esempio 1.19, B_1 si può scrivere come

$$B_1 = \{ x \in \mathbb{R}^n : \sum_{i=1}^n v_i \leq 1, \quad -v_i \leq x_i \leq v_i \quad i = 1, \dots, n \}$$

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

$$B_1 = \{ x \in \mathbb{R}^3 : v_1 + v_2 + v_3 \leq 1, \quad -v_1 \leq x_1 \leq v_1, \quad -v_2 \leq x_2 \leq v_2, \quad -v_3 \leq x_3 \leq v_3 \}$$

in quanto la generalizzazione ad n qualsiasi è poi ovvia.

Iniziamo proiettando la variabile ausiliaria v_1 , che appare in tre vincoli: $v_1 + v_2 + v_3 \leq 1$ (con coefficiente 1, quindi in I^+) $-x_1 - v_1 \leq 0$, $x_1 - v_1 \leq 0$ (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

$$v_1 \leq 1 - v_2 - v_3, \quad -x_1 \leq v_1, \quad x_1 \leq v_1$$

e poi li combiniamo ottenendone due

$$-x_1 \leq 1 - v_2 - v_3, \quad x_1 \leq 1 - v_2 - v_3$$

che sostituiamo ai tre originali nella definizione, terminando il primo passo.

Proiettiamo adesso la variabile ausiliaria v_2 , che appare nei due vincoli $-x_2 \leq v_2$ e $x_2 \leq v_2$ (in I^-) così come nei due vincoli appena creati

$$-x_1 + v_2 + v_3 \leq 1, \quad x_1 + v_2 + v_3 \leq 1,$$

entrambi in I^+ . Occorre quindi creare tutte le possibili 2×2 coppie di tali vincoli, il che ne produce i seguenti quattro

$$-x_1 - x_2 \leq 1 - v_3, \quad x_1 - x_2 \leq 1 - v_3, \quad -x_1 + x_2 \leq 1 - v_3, \quad x_1 + x_2 \leq 1 - v_3.$$

Al terzo ed ultimo passo proiettiamo v_3 , che appare come al solito in due soli vincoli in I^- , $-x_3 \leq v_3$ e $x_3 \leq v_3$, ma in tutti e quattro i vincoli in I^+ appena creati. Formando le 4×2 coppie di tali vincoli si determinano gli 8

$$\begin{aligned} -x_1 - x_2 - x_3 &\leq 1, & x_1 - x_2 - x_3 &\leq 1, & -x_1 + x_2 - x_3 &\leq 1, & x_1 + x_2 - x_3 &\leq 1 \\ -x_1 - x_2 + x_3 &\leq 1, & x_1 - x_2 + x_3 &\leq 1, & -x_1 + x_2 + x_3 &\leq 1, & x_1 + x_2 + x_3 &\leq 1 \end{aligned}$$

che finalmente forniscono la rappresentazione di B_1 in \mathbb{R}^3 . È facile vedere che questi vincoli possono essere rappresentati con la forma

$$\pm x_1 \pm x_2 \pm x_3 \leq 1$$

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

$$B_1 = \{x \in \mathbb{R}^n : \pm x_1 \pm x_2 \dots \pm x_n \leq 1\}$$

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 $\{-1, 1\}^n$ visto nell’Esempio 3.9. Non approfondiremo questa relazione, notando solo che l’ipercubo unitario ha $2n$ vincoli e 2^n vertici, mentre B_1 ha 2^n vincoli, uno per ciascun vertice dell’ipercubo unitario, ma solamente $2n$ vertici, uno per ciascun vincolo dell’ipercubo unitario. Ciò 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.

Gli esempi precedenti mostrano come la dimensionalità delle rappresentazioni “per facce” e “per punti estremi” dei poliedri possa essere (e normalmente in effetti sia) drammaticamente differente. L’Esempio 3.10 mostra anche come la proiezione di un poliedro in uno spazio a dimensione più piccola possa aumentare esponenzialmente il numero di disequazioni 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.

3.1.1.2 L’algebra dei vincoli e delle facce

Vogliamo adesso riesaminare i concetti geometrici sviluppati nei paragrafi precedenti con strumenti algebrici. Dato il poliedro $P = \{x \in \mathbb{R}^n : Ax \leq b\}$, sia $I \subseteq \{1, \dots, m\}$ 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 $\bar{I} = \{1, \dots, m\} \setminus I$ il complemento di I , l’insieme

$$P_I = \{x \in \mathbb{R}^n : A_I x = b_I, A_{\bar{I}} x \leq b_{\bar{I}}\}$$

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 $\{1, \dots, m\}$, 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 $I \subset \{1, \dots, m\}$ 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. Introduciamo adesso alcuni concetti che ci saranno 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.

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 $i = 1, \dots, m$ vale

$$A_i x(\lambda) = A_i \bar{x} + \lambda A_i \xi \leq b_i . \quad (3.11)$$

Chiaramente, qualsiasi direzione $\xi \in \text{rec}(P)$ 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 $A_i \bar{x} = b_i$, (3.11) 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 (3.11) 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} .

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*. È possibile mostrare il seguente risultato:

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 degeneri*; altrimenti ($|I(\bar{x})| > n$) il vertice si dice *degeneri*. 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 degeneri.

Il poliedro ha quattro spigoli limitati, corrispondenti al “lato sinistro”, e quattro spigoli illimitati, corrispondenti ai “lati infiniti” del parallelepipedo.

Esempio 3.13. Basi e vertici dell'ipercubo unitario

Mostriamo ora algebricamente la correttezza delle affermazioni fatte nell'Esempio 3.9. Per questo consideriamo i vincoli che definiscono l'ipercubo unitario H scritti in forma matriciale come

$$\begin{bmatrix} I \\ -I \end{bmatrix} x \leq \begin{bmatrix} u \\ u \end{bmatrix} ,$$

e le sue basi. È ovvio che ciascuna base B dovrà contenere, per ciascuna variabile x_i , $i = 1, \dots, n$, almeno una delle due righe corrispondenti ai due vincoli $x_i \leq 1$ e $-x_i \leq 1$: infatti, altrimenti l' i -esima colonna della corrispondente matrice di base A_B sarebbe tutta nulla, il che contraddice il fatto che A_B sia invertibile. Ma è anche ovvio che A_B non può contenere entrambi i vincoli, perché le due righe corrispondenti sono linearmente dipendenti. In altri termini, si può ottenere una base B scegliendo esattamente uno tra i due vincoli per ciascuna delle variabili: esistono quindi 2^n basi per H . Tali basi sono tutte ammissibili. Infatti, se nella base è presente il vincolo $x_i \leq 1$ allora nella

corrispondente soluzione di base \bar{x} si avrà $\bar{x}_i = 1$, mentre se è presente $-x_i \leq 1$ si avrà $\bar{x}_i = -1$; in ciascuno dei due casi la soluzione non viola l'altro vincolo (fuori base), e ciò avviene indipendentemente per tutte le variabili. Pertanto, come precedentemente indicato i vertici di H sono tutti i 2^n vettori di $\{-1, 1\}^n$.

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 $\text{rec}(P)$, 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:

$$\{x : A_B x \leq 0\} = \{x : -A_B x = \nu, \nu \geq 0\} = \{x = -A_B^{-1} \nu : \nu \geq 0\}. \quad (3.12)$$

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 3.6. Dato un cono poliedrico $C = (x \in \mathbb{R}^n : Ax \leq 0)$, v è un generatore di C se e solo se esiste una base B tale che v è un generatore del *cono di base* $C_B = (x \in \mathbb{R}^n : A_B x \leq 0)$ (che si ottiene quindi attraverso (3.12)) e $v \in C$.

Esempio 3.14. Generatori di un cono simpliciale

Si consideri il poliedro P in Figura 3.3 e la base $B = \{1, 4, 5\}$. Si ha quindi

$$A_B = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad b_B = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad -A_B^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}.$$

Calcolando $A_B^{-1} b_B$ si ottiene infatti il vertice $[0, 0, 1]$ del poliedro; inoltre, il cono simpliciale associato alla base,

$$-x_1 \leq 0, \quad -x_2 \leq 0, \quad x_3 \leq 0$$

corrisponde al cono finitamente generato dai vettori

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$$

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 3.15. 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.

	<i>A</i>	<i>B</i>	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*:

$$\begin{array}{rcll}
 \min & 1200x_1 & + & 750x_2 \\
 & 5x_1 & + & 7x_2 \geq 8 \\
 & 4x_1 & + & 2x_2 \geq 15 \\
 & 2x_1 & + & x_2 \geq 3 \\
 & x_1 & , & x_2 \geq 0
 \end{array}$$

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

$$\begin{array}{rcll}
 \max & 8y_1 & + & 15y_2 + 3y_3 \\
 & 5y_1 & + & 4y_2 + 2y_3 \leq 1200 \\
 & 7y_1 & + & 2y_2 + y_3 \leq 750 \\
 & y_1 & , & y_2 , y_3 \geq 0
 \end{array}$$

I due problemi sono riassunti nella seguente tabella

	x_1	x_2		max
y_1	5	7	\geq	8
y_2	4	2		15
y_3	2	1		3
	\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 disequazioni 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 3.16. Un problema di birra e prezzi

Il primo produttore mondiale di birra, BugWasser (BW) deve organizzare il trasporto della birra dalle sue n fabbriche, $F = \{1, \dots, n\}$, ai suoi m depositi $D = \{1, \dots, m\}$. 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

$$\begin{aligned} \min \quad & \sum_{i \in F} \sum_{j \in D} c_{ij} x_{ij} \\ & \sum_{j \in D} x_{ij} = a_i \quad i \in F \\ & \sum_{i \in F} x_{ij} = b_j \quad j \in D \\ & x_{ij} \geq 0 \quad (i, j) \in F \times D \end{aligned} .$$

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 $-\lambda_i + \mu_j \leq c_{ij}$ per ogni $(i, j) \in F \times D$. Il problema di EIM può quindi essere formulato come

$$\begin{aligned} \max \quad & -\sum_{i \in F} a_i \lambda_i + \sum_{j \in D} b_j \mu_j \\ & -\lambda_i + \mu_j \leq c_{ij} \quad (i, j) \in F \times D \end{aligned}$$

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 $-\sum_{j \in D} x_{ij} = -a_i$.

$$\begin{array}{c}
\lambda_1 \\
\lambda_2 \\
\mu_1 \\
\mu_2 \\
\mu_3
\end{array}
\begin{array}{c}
x_{11} \quad x_{12} \quad x_{13} \quad x_{21} \quad x_{22} \quad x_{23} \\
\begin{array}{|c|c|c|c|c|c|}
\hline
-1 & -1 & -1 & & & \\
\hline
& & & -1 & -1 & -1 \\
\hline
1 & & & 1 & & \\
\hline
& 1 & & & 1 & \\
\hline
& & 1 & & & 1 \\
\hline
\end{array}
\end{array}
= \begin{array}{c} \max \\ \begin{array}{|c|} \hline -a_1 \\ \hline -a_2 \\ \hline b_1 \\ \hline b_2 \\ \hline b_3 \\ \hline \end{array} \end{array}$$

$$\min \begin{array}{|c|c|c|c|c|c|} \hline c_{11} & c_{12} & c_{13} & c_{21} & c_{22} & c_{23} \\ \hline \end{array}$$

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

$$\sum_{(i,j) \in F \times D} c_{ij} x_{ij} \geq \sum_{(i,j) \in F \times D} (\mu_j - \lambda_i) x_{ij} .$$

Mettendo in evidenza μ_j e λ_i ed utilizzando i vincoli del problema di BW si ottiene

$$\sum_{(i,j) \in F \times D} c_{ij} x_{ij} \geq \sum_{j \in F} b_j \mu_j - \sum_{i \in D} a_i \lambda_i ,$$

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

$$(P) \quad \max \{ cx : Ax \leq b, x \geq 0 \} \quad (3.13)$$

possiamo sempre associare ad esso un problema così definito:

$$(D) \quad \min \{ yb : yA \geq c, y \geq 0 \} . \quad (3.14)$$

I problemi (3.13) e (3.14) costituiscono una *coppia di problemi duali*: spesso (3.13) viene chiamato il *primale* e (3.14) il *duale*, ma la nomenclatura è arbitraria in quanto vale il seguente teorema.

Teorema 3.7. Il duale del duale è il primale.

Dimostrazione Utilizzando (3.2.i) ed (3.2.iii) il problema (3.14) può essere trasformato in

$$-\max \{ -yb : -yA \leq -c, y \geq 0 \} ;$$

secondo la definizione (3.13)–(3.14), il duale di questo problema è

$$-\min \{ -cx : -Ax \geq -b, x \geq 0 \} ,$$

che, utilizzando di nuovo (3.2.i) ed (3.2.iii), può essere trasformato in (3.13). \diamond

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*

$$(P) \quad \max \{ cx : Ax \leq b \} \quad (D) \quad \min \{ yb : yA = c, y \geq 0 \} .$$

Infatti, (P) può essere scritto equivalentemente, utilizzando (3.3), come

$$\max \{ cx^+ - cx^- : Ax^+ - Ax^- \leq b, x^+ \geq 0, x^- \geq 0 \} ;$$

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 3.3. Si dimostri il viceversa, cioè che partendo dalla definizione di coppia asimmetrica si ottiene la definizione di coppia simmetrica.

Esercizio 3.4. Si dimostri il Teorema 3.7 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 .

$$\begin{array}{c|c|c|c|c|c|c|c|c}
\max & c & b & A_i x \leq b_i & A_i x \geq b_i & A_i x = b_i & x_j \geq 0 & x_j \leq 0 & x_j \geq 0 \\
\min & b & c & y_i \geq 0 & y_i \leq 0 & y_i \geq 0 & y A^j \geq c_j & y A^j \leq c_j & y A^j = c_j
\end{array} \quad (3.15)$$

Esempio 3.17. Applicazione delle corrispondenze $(P)-(D)$

Il duale del problema di minimo

$$\begin{array}{ll}
\min & 12x_1 + 7x_2 \\
& 5x_1 + 7x_2 = 8 \\
& 4x_1 + 2x_2 \geq 15 \\
& 2x_1 + x_2 \leq 3 \\
& x_1 \geq 0
\end{array}
\quad \text{è} \quad
\begin{array}{ll}
\max & 8y_1 + 15y_2 + 3y_3 \\
& 5y_1 + 4y_2 + 2y_3 \leq 12 \\
& 7y_1 + 2y_2 + y_3 = 7 \\
& \quad y_2 \geq 0 \\
& \quad y_3 \leq 0
\end{array}$$

3.2.2 I teoremi debole e forte della dualità

Come già evidenziato negli esempi 3.15 e 3.16, 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 3.8. (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

$$\left. \begin{array}{l} \bar{y}A = c \\ A\bar{x} \leq b, \bar{y} \geq 0 \end{array} \right\} \implies \bar{y}A\bar{x} = c\bar{x} \implies \bar{y}A\bar{x} \leq \bar{y}b \implies c\bar{x} \leq \bar{y}b.$$

◇

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 3.18. 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:

$$\begin{array}{llll}
 \max & 500x_1 & + & 200x_2 \\
 & x_1 & & \leq 4 \\
 & & x_2 & \leq 7 \\
 & 2x_1 & + & x_2 \leq 9 \\
 & -x_1 & & \leq 0 \\
 & & & -x_2 \leq 0
 \end{array}
 \qquad
 \begin{array}{llllll}
 \min & 4y_1 & + & 7y_2 & + & 9y_3 \\
 & y_1 & & & + & 2y_3 - y_4 = 500 \\
 & & & y_2 & + & y_3 - y_5 = 200 \\
 & y_1 & , & y_2 & , & y_3 , y_4 , y_5 \geq 0
 \end{array}$$

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 $\bar{y}'b = 2500$ 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.

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

$$c(\bar{x} + \lambda\xi) = c\bar{x} + \lambda c\xi > c\bar{x} .$$

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 3.2. 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 perché la regione ammissibile del problema è convessa, il che fornisce una contraddizione. \diamond

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. La condizione del Lemma 3.2 non è però immediata da verificare, nel senso che richiede di dimostrare la *non esistenza* di un certo tipo di direzioni. È usualmente più facile dimostrare l'*esistenza* di qualcosa, nel senso che è sufficiente esibirlo. Per questo è altamente utile la teoria della dualità, ed in particolare la seguente versione “forte” del Teorema 3.8 del quale non diamo immediata dimostrazione poiché essa sarà ottenuta “gratis” nel seguito dall'analisi degli algoritmi.

Teorema 3.9. (Teorema forte della dualità) Se (P) e (D) ammettono entrambi soluzioni ammissibili, allora

$$z(P) = \max\{cx : Ax \leq b\} = \min\{yb : yA = c, y \geq 0\} = z(D) .$$

Da questa analisi si può derivare la seguente utile caratterizzazione:

Teorema 3.10. 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
	ottimo finito	*		
(D)	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 resto della dimostrazione segue dal fatto che se uno dei due problemi ha ottimo finito allora deve averlo anche l'altro, come è facile verificare. \diamond

3.2.3 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 3.9 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

$$c\bar{x} = \bar{y}b \iff \bar{y}A\bar{x} = \bar{y}b \iff \bar{y}(b - A\bar{x}) = 0$$

vale quando $\bar{y}A = c$ indipendentemente dall'eventuale ammissibilità delle soluzioni, la seguente definizione risulterà utile nel seguito.

Definizione 3.1. 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*:

$$\bar{y}(b - A\bar{x}) = 0. \quad (3.16)$$

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 3.11. 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 (3.16).

Esplicitando il prodotto scalare, l'equazione (3.16) può essere riscritta

$$\bar{y}(b - A\bar{x}) = \sum_{i=1}^m \bar{y}_i(b_i - A_i\bar{x}) = 0.$$

Quindi, una condizione *sufficiente* perché valga (3.16) è

$$\bar{y}_i(b_i - A_i\bar{x}) = 0 \quad i = 1, \dots, m \quad (3.17)$$

in quanto la somma di termini nulli da sicuramente zero; a sua volta, (3.17) è equivalente a

$$\begin{aligned} \bar{y}_i &> 0 \implies A_i\bar{x} = b_i \\ A_i\bar{x} &< b_i \implies \bar{y}_i &= 0 \end{aligned} \quad i = 1, \dots, m. \quad (3.18)$$

Si noti che (3.17) e (3.18) sono *equivalenti* a (3.16) 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, è facile dimostrare che 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} .

Una soluzione ammissibile \bar{y} complementare a \bar{x} soddisfa (3.18), 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). 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:

$$\begin{array}{ll}
 \max & x_1 + 2x_2 \\
 (P) & x_1 + x_2 \leq 5 \\
 & x_1 \leq 4 \\
 & x_2 \leq 3 \\
 & -x_1 \leq 0 \\
 & -x_2 \leq 0 \\
 \min & 5y_1 + 4y_2 + 3y_3 \\
 (D) & y_1 + y_2 - y_4 = 1 \\
 & y_1 + y_3 - y_5 = 2 \\
 & y_1, y_2, y_3, y_4, y_5 \geq 0
 \end{array}$$

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 3.11 si ha quindi che per *qualsiasi* soluzione ottima \bar{y} di (D) deve valere

$$\begin{aligned}
 \bar{x}_1 < 4 &\implies \bar{y}_2 = 0 \\
 -\bar{x}_1 < 0 &\implies \bar{y}_4 = 0 \\
 -\bar{x}_2 < 0 &\implies \bar{y}_5 = 0
 \end{aligned}$$

Quindi, le uniche componenti possibilmente non nulle di qualsiasi soluzione ottima \bar{y} di (D) devono soddisfare il sistema

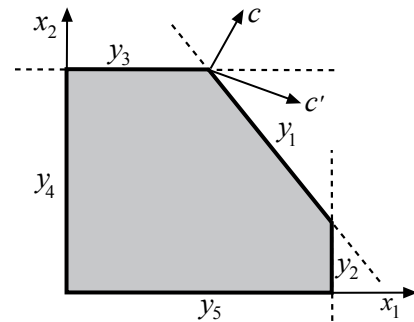
$$y_1 = 1, \quad y_1 + y_3 = 2.$$

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 3.11 essa è effettivamente ottima: infatti, anch'essa ha valore $\bar{y}b = 8$.

Se invece il vettore dei costi di (P) fosse stato $c' = [3, -1]$, ripetendo gli stessi ragionamenti si sarebbe giunti a

$$y_1 = 3, \quad y_1 + y_3 = -1$$

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 dimostra *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.

Scriviamo la coppia asimmetrica che comprende il problema in esame:

$$\begin{array}{ll}
 \max & 4x_1 + \alpha x_2 \\
 (P) & 2x_1 + x_2 \leq 7 \\
 & x_1 - x_2 \leq 3 \\
 & x_1 \leq 3 \\
 & 2x_1 - x_2 \leq 5 \\
 \min & 7y_1 - 3y_2 + 3y_3 + 5y_4 \\
 (D) & 2y_1 + y_2 + y_3 + 2y_4 = 4 \\
 & y_1 - y_2 - y_4 = \alpha \\
 & y_1, y_2, y_3, y_4 \geq 0
 \end{array}$$

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, appearing solo in funzione obiettivo, non influenza i vincoli e quindi l'ammissibilità). L'insieme degli indici dei vincoli attivi in \bar{x} è $I(\bar{x}) = \{i \in \{1, \dots, m\} : A_i \bar{x} = b_i\} = \{1, 3, 4\}$. 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

$$\begin{cases} 2y_1 + y_3 + 2y_4 = 4 \\ y_1 - y_4 = \alpha \\ y_1, y_3, y_4 \geq 0 \end{cases}$$

Per ogni fissato valore di α , tale sistema ammette infinite soluzioni della forma

$$\begin{bmatrix} y_1, y_3, y_4 \end{bmatrix} = \begin{bmatrix} 1 + \alpha/2 - y_3/4, y_3, 1 - \alpha/2 - y_3/4 \end{bmatrix} .$$

Pertanto, \bar{x} è ottima per un dato valore di α se e solo se almeno una di tali soluzioni è ammissibile, ossia

$$1 + \alpha/2 - y_3/4 \geq 0, \quad y_3 \geq 0, \quad 1 - \alpha/2 - y_3/4 \geq 0,$$

che equivale a

$$4 + 2\alpha \geq y_3, \quad y_3 \geq 0, \quad 4 - 2\alpha \geq y_3.$$

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

$$\bar{y} = \begin{bmatrix} 1 + \alpha/2 - y_3/4, 0, y_3, 1 - \alpha/2 - y_3/4 \end{bmatrix}$$

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 degenera, 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 soluzioni ottime del primale.

Il duale del problema in oggetto è:

$$\begin{array}{rclclcl} \min & (2 - \varepsilon)y_1 & & + & 4y_3 & + & (3 + \varepsilon)y_4 \\ & y_1 & - & y_2 & - & y_3 & + & 2y_4 & = & 3 \\ & -3y_1 & + & y_2 & - & y_3 & + & 3y_4 & = & 0 \\ & 3y_1 & + & y_2 & - & y_3 & - & 3y_4 & = & 0 \\ & y_1 & , & y_2 & , & y_3 & , & y_4 & \geq & 0 \end{array}$$

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

$$\begin{cases} x_1 - 3x_2 + 3x_3 = 2 - \varepsilon \\ 2x_1 + 3x_2 - 3x_3 = 3 + \varepsilon \end{cases}.$$

Posto $3(x_2 - x_3) = \alpha$, tale sistema può essere riscritto come

$$\begin{cases} x_1 - \alpha = 2 - \varepsilon \\ 2x_1 + \alpha = 3 + \varepsilon \end{cases};$$

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

$$[x_1, x_2, x_3] = [5/3, x_3 + \varepsilon/3 - 1/9, x_3] \quad \text{per} \quad -50/18 - \varepsilon/6 \leq x_3 \leq 16/18 - \varepsilon/6 ,$$

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 3.5. Siano \bar{x} e \bar{y} soluzioni ammissibili rispettivamente per (3.13) e (3.14), allora \bar{x} e \bar{y} sono soluzioni ottime se e solo se $\bar{y}(b - A\bar{x}) = 0$ e $(\bar{y}A - c)\bar{x} = 0$.

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 $\bar{y} = [0, 0, 1]$ è 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.

$$\begin{array}{rcll} \max & x_1 & & \\ & -x_1 & + & x_2 \leq 1 \\ & 2x_1 & - & x_2 \leq 1 \\ & x_1 & + & x_2 \leq \alpha \\ & x_1 & , & x_2 \geq 0 \end{array} \quad (P_\alpha)$$

Secondo la definizione di coppia simmetrica, il duale di (P_α) è

$$\begin{array}{rcll} \min & y_1 & + & y_2 + \alpha y_3 \\ (D_\alpha) & -y_1 & + & 2y_2 + y_3 \geq 1 \\ & y_1 & - & y_2 + y_3 \geq 0 \\ & y_1 & , & y_2 , y_3 \geq 0 \end{array}$$

È immediato verificare che \bar{y} è ammissibile per ogni valore di α . Possiamo quindi utilizzare il Corollario 3.5 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

$$\begin{aligned} y_4(b_3 - A_3x) = 0 &\equiv 1(\alpha - x_1 - x_2) = 0 \equiv x_1 + x_2 = \alpha \\ x_2(yA^2 - c_2) = 0 &\equiv x_2(1 - 0) = 0 \equiv x_2 = 0 \end{aligned} .$$

Le altre condizioni degli scarti complementari non impongono nulla: quelle duali $y_i(b_i - A_i x) = 0$ 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

$$\begin{cases} -\alpha + 0 \leq 1 \\ 2\alpha - 0 \leq 1 \end{cases}$$

(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 disuguaglianza, 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

$$2y_2 + y_3 = 1 \quad , \quad -y_2 + y_3 \geq 0 \quad , \quad y_2 \geq 0 \quad , \quad y_3 \geq 0 .$$

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.

3.2.4 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 $N = \{1, \dots, m\} \setminus B$, possiamo associare alla base B la *soluzione duale di base*

$$\bar{y} = [\bar{y}_B, \bar{y}_N] = [cA_B^{-1}, 0] \quad .$$

Tale soluzione soddisfa per costruzione la proprietà $\bar{y}A = c$ richiesta dalla Definizione 3.1; inoltre, segue facilmente da (3.18) 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 (3.16).

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}
<i>ammissibile</i>	$A_N\bar{x} \leq b_N$	$\bar{y}_B \geq 0$
<i>non ammissibile</i>	$\exists i \in N : A_i\bar{x} > b_i$	$\exists i \in B : \bar{y}_i < 0$
<i>degeneri</i>	$\exists i \in N : A_i\bar{x} = b_i$	$\exists i \in B : \bar{y}_i = 0$
<i>non degeneri</i>	$\forall i \in N : A_i\bar{x} \neq b_i$	$\forall i \in B : \bar{y}_i \neq 0$

Una base B è detta *primale o duale ammissibile, non ammissibile, degeneri o non degeneri* 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 3.23. Soluzioni di base complementari

Si consideri la coppia di problemi duali rappresentata nella tabella e nel grafico di Figura 3.6.

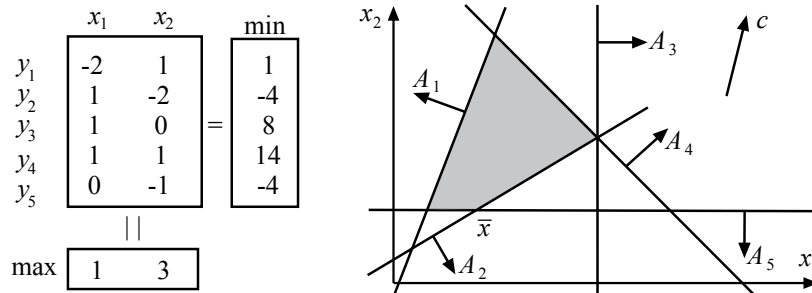


Figura 3.6: Dati per l'Esempio 3.23

Per $B = \{2, 5\}$ si ha

$$A_B = A_B^{-1} = \begin{bmatrix} 1 & -2 \\ 0 & -1 \end{bmatrix} \implies \bar{x} = A_B^{-1}b_B = \begin{bmatrix} 1 & -2 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} -4 \\ -4 \end{bmatrix} = \begin{bmatrix} 4 \\ 4 \end{bmatrix} \quad .$$

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 $\bar{y} = \begin{bmatrix} 1 \\ -5 \end{bmatrix}$ 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 $\text{rango}(A_I) = \text{rango}(A) = n$. Infatti, $\text{rango}(A_I) = n$ 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 degeneri*. 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 *degeneri*: 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 3.24. Soluzioni primali degeneri

Nel problema dell'Esempio 3.23 si consideri $\bar{x} = [8, 6]$ con $c\bar{x} = 26$: si ha

$$\begin{aligned} A_1\bar{x} &= -10 < 1 \\ A_2\bar{x} &= -4 = -4 \\ A_3\bar{x} &= 8 = 8 \\ A_4\bar{x} &= 14 = 14 \\ A_5\bar{x} &= -6 < -4 \end{aligned}$$

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 degenera, 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

$$\bar{y}' = [0, -3/2, 5/2, 0, 0] \quad , \quad \bar{y}'' = [0, -2/3, 0, 5/3, 0] \quad , \quad \bar{y}''' = [0, 0, -2, 3, 0]$$

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 degenera: 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 $\text{rango}(A) = n$). Conseguentemente, più soluzioni di base \bar{x} di (P) possono costituire con \bar{y} una coppia di soluzioni complementari.

Esempio 3.25. Soluzioni duali degeneri

Si consideri ancora il problema dell'Esempio 3.23 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 degenera per (D): infatti risulta $J = \{4\}$. Si può facilmente 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 degenera)	(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; ciò vale simmetricamente per il primale ed il duale, come è facile dimostrare.

Teorema 3.12. Siano \bar{x} una soluzione di base primale ammissibile non degenera 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.

Teorema 3.13. Siano \bar{y} una soluzione di base duale ammissibile non degenera 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.

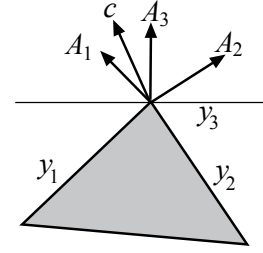
Nel caso di soluzioni di base degeneri, invece, l'ammissibilità della soluzione di base complementare fornisce soltanto una condizione sufficiente di ottimalità, come mostra il seguente esempio.

Esempio 3.26. 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:

$$\begin{aligned} c \in \text{cono}(\{A_1, A_2\}) &\implies \bar{y}_{B'} \geq 0 \\ c \in \text{cono}(\{A_1, A_3\}) &\implies \bar{y}_{B''} \geq 0 \quad ; \\ c \notin \text{cono}(\{A_2, A_3\}) &\implies \bar{y}_{B'''} \not\geq 0 \end{aligned}$$

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 3.12 e 3.13:

Teorema 3.14. 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 3.15. 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 3.5. Costruire esempi di problemi (P) , (D) rappresentabili in \mathbb{R}^2 aventi soluzione ottima degenera, individuando una matrice di base che non soddisfi le condizioni di ottimo.

Esercizio 3.6. Costruire, se esiste, una coppia di problemi (P) , (D) tale che le soluzioni ottime \bar{x} e \bar{y} siano entrambe degeneri.

3.3 Algoritmi del Simplexso

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 simplexso*. Questo algoritmo costituisce il primo approccio computazionalmente efficiente per la soluzione di problemi di PL ; originariamente proposto da G.B. Dantzig [1951] a partire da un'idea di J. Von Neuman, il metodo del simplexso 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 simplexso 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 simplexso duale*. Vedremo però che i due sono nei fatti sostanzialmente lo stesso algoritmo.

3.3.1 L'algoritmo del Simplexso Primale

L'algoritmo del simplexso 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 §???. Introduciamo i concetti fondamentali per mezzo di un esempio.

Esempio 3.27. Esecuzione “informale” del Simplexso Primale

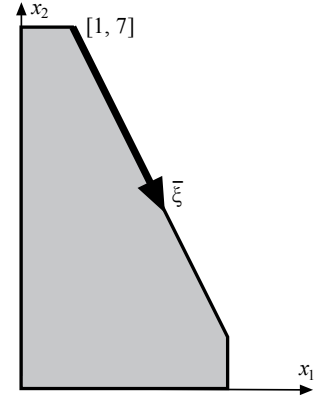
Riprendiamo il problema della Pintel, e consideriamo il vertice $\bar{x} = [1, 7]$, che già sappiamo non essere una soluzione ottima; tale vertice è una soluzione primale di base non degenere in quanto $B = I(\bar{x}) = \{2, 3\}$. Poiché \bar{x} non è una soluzione ottima per il problema, dovrà necessariamente 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:

$$\begin{array}{rcl} \xi_2 & \leq & 0 \\ 2\xi_1 + \xi_2 & \leq & 0 \\ 500\xi_1 + 200\xi_2 & > & 0 \end{array}, \quad \begin{array}{rcl} 2y_2 & = & 500 \\ y_1 + y_2 & = & 200 \\ y_1, y_2 & \geq & 0 \end{array}.$$

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

$$\begin{array}{rcl} y_1 + 2y_3 & = & 500 \\ y_3 & = & 200 \\ y_1, y_3 & \geq & 0 \end{array}$$

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 (3.12). 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 (3.12) 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

$$\xi_i = -A_B^{-1}u_{B(i)} \quad i = 1, \dots, n$$

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 però 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 3.6).

Esempio 3.28. Direzioni ξ_h non ammissibili

Si consideri il seguente coppia di problemi duali

$$(P) \quad \begin{array}{rcl} \max & x_1 & + \quad x_2 \\ & -x_1 & - \quad x_2 \leq 1 \\ & x_1 & \leq 0 \\ & x_1 & + \quad x_2 \leq 2 \end{array} \quad , \quad (D) \quad \begin{array}{rcl} \min & y_1 & + \quad y_2 \quad + \quad 2y_4 \\ & -y_1 & - \quad y_2 \quad + \quad y_3 \quad + \quad y_4 = 1 \\ & -y_1 & - \quad y_2 \quad + \quad y_4 = 1 \\ & y_1 & , \quad y_2 \quad , \quad y_3 \quad , \quad y_4 \geq 0 \end{array}$$

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

$$A_B = \begin{bmatrix} 0 & -1 \\ -1 & -1 \end{bmatrix}, \quad \bar{x} = \begin{bmatrix} 1 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \quad \bar{y}_B = [1, 1] \begin{bmatrix} 1 & -1 \\ -1 & 0 \end{bmatrix} = [0, -1]$$

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

$$C_B = \left\{ \begin{array}{rcl} -\xi_1 & - & \xi_2 \leq 0 \\ -\xi_1 & - & \xi_2 \leq 0 \end{array} \right\} \supset C(\bar{x}) = \left\{ \begin{array}{rcl} -\xi_1 & - & \xi_2 \leq 0 \\ \xi_1 & & \leq 0 \end{array} \right\}.$$

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

$$c\xi_2 = [1, 1] \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 1 = -\bar{y}_2 > 0$$

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 ciò 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

$$A_i x(\lambda) = A_i \bar{x} + \lambda A_i \xi \leq b_i \iff \lambda \leq (b_i - A_i \bar{x}) / (A_i \xi).$$

Pertanto, il massimo passo che può essere effettuato lungo ξ a partire da \bar{x} senza violare il vincolo i -esimo risulta essere

$$\lambda_i = \begin{cases} (b_i - A_i \bar{x}) / (A_i \xi) & \text{se } A_i \xi > 0 \\ +\infty & \text{altrimenti} \end{cases}. \quad (3.19)$$

Scegliendo il più piccolo di questi valori, ovvero

$$\bar{\lambda} = \min \{ \lambda_i : i \in N \}, \quad (3.20)$$

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

$$A_ix(\lambda) = A_i(\bar{x} + \lambda\xi_h) = A_i\bar{x} = b_i \quad i \in B \setminus \{h\} , \quad (3.21)$$

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. Con questa caratterizzazione si può dimostrare la seguente invariante algoritmica:

Lemma 3.3. 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.

Nonostante l'analisi precedente sia stata sviluppata per il caso non degenerare, è facile verificare che essa si estende senza alcun problema a quello degenerare, in cui possiamo avere sia $\bar{\lambda} > 0$ che $\bar{\lambda} = 0$. Nel primo caso si effettua lo spostamento analogamente al caso non degenerare. 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 degenerare*.

Un cambiamento di base non degenerare, 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 degenerare, 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 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 3.29. Cambiamenti di base degeneri

Per esemplificare lo sviluppo precedente si consideri la seguente coppia di problemi duali

$$\begin{array}{ll} \max & x_2 \\ (P) \quad & \begin{array}{rcl} -x_1 & - & x_2 \leq 1 \\ -x_1 & & \leq 1 \\ -x_1 & + & x_2 \leq 0 \\ -x_1 & + & x_2 \leq -1 \\ x_1 & & \leq 2 \end{array} \end{array} \quad \begin{array}{ll} \min & y_1 + y_2 - y_4 + 2y_5 \\ (D) \quad & \begin{array}{rcl} -y_2 & - & y_3 - y_4 + y_5 = 0 \\ -y_1 - y_2 & + & y_4 = 1 \\ y_1, y_2, y_3, y_4, y_5 & \geq & 0 \end{array} \end{array}$$

e la base $B^1 = \{1, 2\}$. Le corrispondenti matrice e soluzione primale di base sono

$$A_{B^1} = \begin{bmatrix} 0 & -1 \\ -1 & -1 \end{bmatrix}, \quad \bar{x} = A_{B^1}^{-1}b_{B^1} = \begin{bmatrix} 1 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}.$$

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 degenerare. La soluzione duale di base corrispondente a B^1 è

$$y_{B^1}^1 = cA_{B^1}^{-1} = [0, 1] \begin{bmatrix} 1 & -1 \\ -1 & 0 \end{bmatrix} = [-1, 0].$$

Poiché $y_1^1 < 0$, selezionando $h = 1$ ($B^1(h) = 1$) la direzione

$$\xi^1 = -A_{B^1}^{-1} u_{B^1(h)} = - \begin{bmatrix} 1 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix} ,$$

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, non si è ottenuta né una soluzione duale ammissibile, né una direzione ammissibile di crescita. Possiamo però effettuare un cambiamento di base degenerare, scegliendo $k_1 = 3$. Per la nuova base $B^2 = B^1 \setminus \{h\} \cup \{k\} = \{3, 2\}$ risulta

$$A_{B^2} = \begin{bmatrix} -1 & 0 \\ -1 & -1 \end{bmatrix} , \quad A_{B^2}^{-1} = \begin{bmatrix} -1 & 0 \\ 1 & -1 \end{bmatrix} , \quad y_{B^2}^2 = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} -1 & 0 \\ 1 & -1 \end{bmatrix} = \begin{bmatrix} 1 & -1 \end{bmatrix} ,$$

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 degenerare sostituendo $k = 4$ a h : otteniamo così la base $B^3 = \{3, 4\}$ per cui abbiamo

$$A_{B^3} = \begin{bmatrix} -1 & 0 \\ -1 & 1 \end{bmatrix} , \quad A_{B^3}^{-1} = \begin{bmatrix} -1 & 0 \\ -1 & 1 \end{bmatrix} , \quad y_{B^3}^3 = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} -1 & 0 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} -1 & 1 \end{bmatrix} .$$

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

$$A_1 \xi^3 = \begin{bmatrix} 0 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = -1 < 0 , \quad A_2 \xi^3 = \begin{bmatrix} -1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = -2 < 0 .$$

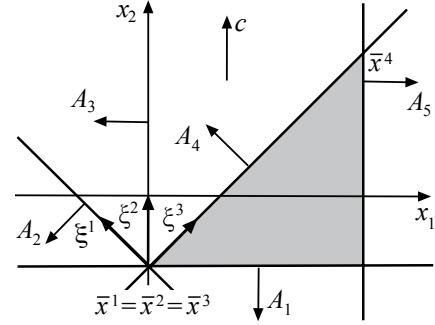
Pertanto, ξ^3 risolve (P_R) e quindi \bar{x} non è una soluzione ottima di (P) . Infatti, il massimo spostamento consentito lungo ξ^3 , calcolato mediante (3.20),

$$\bar{\lambda} = \lambda_5 = \frac{b_5 - A_5 \bar{x}}{A_5 \xi^3} = \frac{2 - 0}{1} = 2 ,$$

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 *Simplexso Primale*.

```

procedure (  $B, \bar{x}, \bar{y}, stato$  ) = Simplesso.Primale(  $A, b, c, B$  ) {
  for(  $stato = \text{" "}; ;$  ) { // invariante:  $B$  primale ammissibile
     $\bar{x} = A_B^{-1}b_B; \bar{y} = [\bar{y}_B, \bar{y}_N] = [cA_B^{-1}, 0];$ 
    if(  $\bar{y}_B \geq 0$  ) then {  $stato = \text{"ottimo"}; \text{break};$  }
     $h = \min\{i \in B : \bar{y}_i < 0\}; \xi = -A_B^{-1}u_{B(h)};$ 
    if(  $A_N\xi \leq 0$  ) then {  $stato = \text{"P.illimitato"}; \text{break};$  }
     $\bar{\lambda} = \min\{\lambda_i = (b_i - A_i\bar{x}) / (A_i\xi), A_i\xi > 0, i \in N\};$ 
     $k = \min\{i \in N : \lambda_i = \bar{\lambda}\}; B = B \cup \{k\} \setminus \{h\};$ 
  }
}

```

Procedura 3.1: Simplexso Primale

L'algoritmo riceve in input una descrizione del problema ed una 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 degenerare.

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 degenerare, 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 $k = \min\{i \in N : \lambda_i = \bar{\lambda}\}$. Si osservi 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 possibile dimostrare 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. Da questo segue la terminazione finita dell'Algoritmo del Simplexso primale, e quindi anche il Teorema forte della dualità, almeno nel caso in cui la matrice A abbia rango pieno. Infatti, se il primale ed il duale non sono vuoti allora esiste una base ammissibile di partenza per il primale (e vedremo nella §3.3.1.1 come sia possibile determinarla algebricamente). Poiché l'algoritmo termina in un numero finito di iterazioni, e non può determinare che (P) sia superiormente illimitato, sicuramente determina una soluzione ottima di base del duale, che rispettando gli scarti complementari ha lo stesso valore della funzione obiettivo di quella primale. La dimostrazione può essere generalizzata al caso in cui A non abbia rango pieno sfruttando le idee del Teorema 3.5.

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

$$c\xi > 0 \quad \text{e} \quad A\xi = \begin{bmatrix} A_B \\ A_N \end{bmatrix} \xi = \begin{bmatrix} -u_{B(h)} \\ z_N \end{bmatrix} \quad \text{con} \quad z_N = A_N \xi \leq 0 \quad .$$

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

$$c\xi + y_h = z_N y_N \quad ,$$

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 $yA = c$.

Esercizio 3.7. Determinare la complessità delle singole istruzioni dell'algoritmo *Simpleso_Primate*.

Esercizio 3.8. Determinare l'occupazione di memoria dell'algoritmo *Simpleso_Primate*.

Esempio 3.30. Esecuzione dell'algoritmo del simpleso primale

Consideriamo la coppia asimmetrica di problemi duali dell'Esempio 3.23 e applichiamo l'algoritmo del simpleso 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

$$\xi^1 = -A_{B^1}^{-1} u_{B^1(h)} = \begin{bmatrix} -1 & 2 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \quad .$$

La ricerca dell'indice uscente è effettuata analizzando i vincoli non in base:

$$A_{N^1} \xi^1 = \begin{bmatrix} -2 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} -3 \\ 2 \\ 3 \end{bmatrix} \quad .$$

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 3.6 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 dei 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 (3.19):

$$\lambda_3 = (14 - 8) / 3 = 2 \quad , \quad \lambda_4 = (8 - 4) / 2 = 2 \quad ;$$

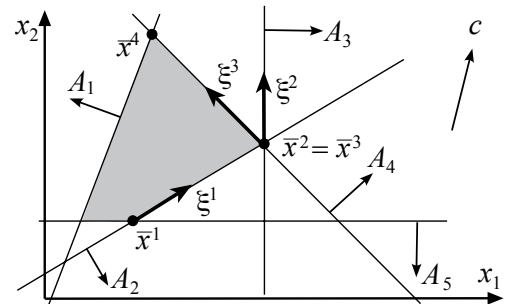
essendo $\bar{\lambda} = \lambda_3 = \lambda_4 = 2$, per la regola anticiclo di Bland si ha $k = \min\{3, 4\} = 3$. La nuova base è perciò $B^2 = B^1 \setminus \{h\} \cup \{k\} = \{2, 5\} \setminus \{5\} \cup \{3\} = \{2, 3\}$.

Effettuiamo la seconda iterazione:

$$A_{B^2} = \begin{bmatrix} 1 & -2 \\ 1 & 0 \end{bmatrix} \quad , \quad A_{B^2}^{-1} = \begin{bmatrix} 0 & 1 \\ -1/2 & 1/2 \end{bmatrix} \quad , \quad b_{B^2} = \begin{bmatrix} -4 \\ 8 \end{bmatrix} \quad ;$$

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:

$$\xi^2 = \begin{bmatrix} 0 & -1 \\ 1/2 & -1/2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1/2 \end{bmatrix} \quad , \quad A_{N^2}\xi^2 = \begin{bmatrix} -2 & 1 \\ 1 & 1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1/2 \end{bmatrix} = \begin{bmatrix} 1/2 \\ 1/2 \\ -1/2 \end{bmatrix} \quad ,$$

$$J^2 = \{1, 4\} \quad , \quad \lambda_1 = 22 \quad , \quad \lambda_4 = 0 = \bar{\lambda} \quad , \quad k = 4 \quad .$$

$\lambda_4 = 0$ è una conseguenza del fatto che $A_4\bar{x}^2 = 14 = b_4$, cioè che la soluzione di base è primale degenera, come avevamo già mostrato nell'Esempio 3.24. 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

$$A_{B^3} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \quad , \quad A_{B^3}^{-1} = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \quad , \quad b_{B^3} = \begin{bmatrix} 8 \\ 14 \end{bmatrix} \quad , \quad \bar{x}^3 = \begin{bmatrix} 8 \\ 6 \end{bmatrix} \quad .$$

Abbiamo effettuato un cambio di base degenera, in quanto la soluzione primale individuata coincide con la precedente; la nuova soluzione di base duale, come visto nell'Esempio 3.24, è diversa dalla precedente, infatti:

$$\bar{y}_{B^3} = \begin{bmatrix} 1 & 3 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} -2 & 3 \end{bmatrix} \quad , \quad \bar{y}_{N^3} = 0 \quad , \quad \bar{y}^3 = \begin{bmatrix} 0 & 0 & -2 & 3 & 0 \end{bmatrix} \quad .$$

L'indice uscente è quindi $h = 3$, e

$$\xi^3 = \begin{bmatrix} -1 & 0 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \quad , \quad A_{N^3}\xi^3 = \begin{bmatrix} -2 & 1 \\ 1 & -2 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ -3 \\ -1 \end{bmatrix} \quad ;$$

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

$$A_{B^4} = \begin{bmatrix} -2 & 1 \\ 1 & 1 \end{bmatrix} \quad , \quad A_{B^4}^{-1} = \begin{bmatrix} -1/3 & 1/3 \\ 1/3 & 2/3 \end{bmatrix} \quad , \quad b_{B^4} = \begin{bmatrix} 1 \\ 14 \end{bmatrix} \quad , \quad \bar{x}^4 = \begin{bmatrix} 13/3 \\ 29/3 \end{bmatrix}$$

$$\bar{y}_{B^4} = \begin{bmatrix} 2/3 & 7/3 \end{bmatrix} \quad , \quad \bar{y}_{N^4} = 0 \quad , \quad \bar{y}^4 = \begin{bmatrix} 2/3 & 0 & 0 & 7/3 & 0 \end{bmatrix} \geq 0$$

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 3.31. Simplexso primale (algebrico), caso illimitato

Si vuole risolvere il problema di PL dato applicando l'algoritmo del simplexso 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.

$$\begin{array}{rcll} \max & -4x_1 & + & 2x_2 \\ & x_1 & + & x_2 & \leq & 12 \\ & -x_1 & + & x_2 & \leq & 2 \\ & x_1 & - & x_2 & \leq & 2 \\ & x_1 & & & \leq & 6 \\ & & & x_2 & \leq & 4 \end{array}$$

$$\text{it.1)} \quad B = \{3, 4\} \quad , \quad A_B = \begin{bmatrix} 1 & -1 \\ 1 & 0 \end{bmatrix} \quad , \quad A_B^{-1} = \begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix} \quad , \quad \bar{x} = A_B^{-1}b_B = \begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 6 \end{bmatrix} = \begin{bmatrix} 6 \\ 4 \end{bmatrix}$$

$$\bar{y}_B = cA_B^{-1} = \begin{bmatrix} -4 & 2 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} -2 & -2 \end{bmatrix} \quad , \quad \bar{y}_N = 0 \quad , \quad \bar{y} = \begin{bmatrix} 0 & 0 & -2 & -2 & 0 \end{bmatrix}$$

La base è primale degenera in quanto esiste almeno un indice $i \in N$ tale che $A_i\bar{x} = b_i$ (in particolare, $i = 5$) ma duale non degenera in quanto $\bar{y}_i \neq 0$ per ogni $i \in B$. $h = \min\{i \in B : \bar{y}_i < 0\} = \min\{3, 4\} = 3$, $B(h) = 1$

$$\xi = -A_B^{-1}u_{B(h)} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad , \quad A_N\xi = \begin{bmatrix} 1 & 1 \\ -1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad , \quad J = \{i \in N : A_i\xi > 0\} = \{1, 2, 5\}$$

$$\lambda_i = (b_i - A_i\bar{x}) / A_i\xi \quad , \quad \lambda_1 = 2 \quad , \quad \lambda_2 = 4 \quad , \quad \lambda_5 = 0 \quad , \quad \bar{\lambda} = \min\{\lambda_i : i \in J\} = 0$$

$$k = \min\{i \in J : \lambda_i = \bar{\lambda}\} = 5 \quad [\text{cambio di base degenera}]$$

$$\text{it.2)} B = \{4, 5\}, \quad A_B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad A_B^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \bar{x} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 6 \\ 4 \end{bmatrix} = \begin{bmatrix} 6 \\ 4 \end{bmatrix}$$

$$\bar{y}_B = [-4, 2] \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = [-4, 2], \quad \bar{y} = [0, 0, 0, -4, 2]$$

[base ovviamente ancora primale degenera e duale non degenera] $h = 4$, $B(h) = 1$

$$\xi = \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \quad A_N \xi = \begin{bmatrix} 1 & 1 \\ -1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}, \quad J = \{2\}, \quad \bar{\lambda} = \lambda_2 = 4, \quad k = 2$$

$$\text{it.3)} B = \{2, 5\}, \quad A_B = \begin{bmatrix} -1 & 1 \\ 0 & 1 \end{bmatrix}, \quad A_B^{-1} = \begin{bmatrix} -1 & 1 \\ 0 & 1 \end{bmatrix}, \quad \bar{x} = \begin{bmatrix} -1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$

$$\bar{y}_B = [-4, 2] \begin{bmatrix} -1 & 1 \\ 0 & 1 \end{bmatrix} = [4, -2], \quad \bar{y} = [0, 4, 0, 0, -2]$$

[base primale non degenera e duale non degenera] $h = 5$, $B(h) = 2$

$$\xi = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, \quad A_N \xi = \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \end{bmatrix} = \begin{bmatrix} -2 \\ 0 \\ -1 \end{bmatrix}$$

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

$$cx(\lambda) = \bar{c} + \lambda \xi = [-4, 2] \begin{bmatrix} 2 - \lambda \\ 4 - \lambda \end{bmatrix} = 2\lambda$$

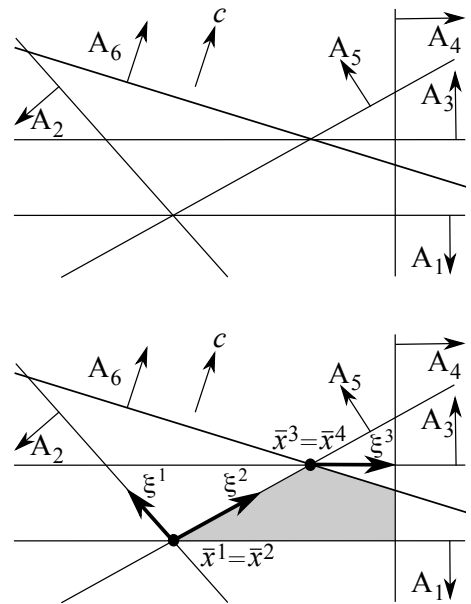
è chiaramente crescente in λ . Pertanto il problema primale è superiormente illimitato e di conseguenza il problema duale è vuoto.

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 3.32. Simpleso primale geometrico

Si vuole risolvere per mezzo dell'algoritmo del simplesso primale il problema di *PL* utilizzando come dati esclusivamente la figura qui accanto in cui è rappresentato 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 uscenti ed entranti, giustificando le risposte con argomentazioni geometriche. Allo stesso modo 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 3.7 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.



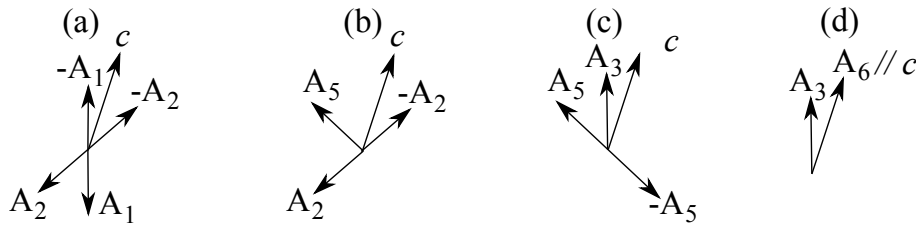


Figura 3.7: Ingrandimento dei dettagli rilevanti nell'esecuzione geometrica

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 3.7.(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 degenerare, in quanto $I(\bar{x}^1) = \{1, 2, 5\} \supsetneq B$, ed è duale non degenerare 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 degenerare.

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 3.7.(b); quindi, $h = 2$. La base è quindi ancora sia primale degenerare (ovviamente, in quanto la soluzione primale di base è la stessa dell'iterazione precedente) che duale non degenerare. 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 degenerare).

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 3.7.(c); quindi, $h = 5$. La base è di nuovo primale degenerare in quanto $I(\bar{x}^3) = \{3, 5, 6\} \supsetneq B$, e duale non degenerare. 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 degenerare.

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 degenerare", ossia è collineare ad A_3 , come mostrato in Figura 3.7.(d). La base è quindi ancora primale degenerare (ovviamente, dato che $\bar{x}^3 = \bar{x}^4$, come mostrato in figura, avendo fatto un passo degenerare) ma adesso è anche duale degenerare. 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 degenerare, 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, $c \notin \text{cono}(\{A_3, A_5\})$ (si veda ancora la Figura 3.7.(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 *Simplex_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 *Simplex_Primale*, applicandola ad un *problema ausiliario* opportunamente costruito. Possiamo innanzitutto assumere di avere a disposizione una base B ; questa esiste per l'assunzione che $\text{rango}(A) = n$, e può essere facilmente determinata.

Esercizio 3.9. 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

$$(PA) \quad \begin{array}{rcl} \max & & -u\nu \\ & A_H x & \leq b_H \\ & A_J x & -\nu \leq b_J \\ & & -\nu \leq 0 \end{array}$$

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

$$\begin{bmatrix} A_B & 0 \\ A_J & -I \end{bmatrix} \begin{bmatrix} x \\ \nu \end{bmatrix} = \begin{bmatrix} b_B \\ b_J \end{bmatrix} \quad \text{dove} \quad \mathcal{A}_B = \begin{bmatrix} A_B & 0 \\ A_J & -I \end{bmatrix}$$

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 *Simplexso-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 3.33. Individuazione di una base primale ammissibile

Si consideri il seguente problema di PL

$$(P) \quad \begin{array}{rcl} \max & x_1 & + 3x_2 \\ & x_1 & + 2x_2 \leq 5 \\ & x_1 & - x_2 \leq 2 \\ & x_1 & + 3x_2 \leq 7 \\ & & -x_2 \leq 0 \end{array}$$

e la base $B = \{1, 4\}$ non primale ammissibile, in quanto

$$A_B = A_B^{-1} = \begin{bmatrix} 1 & 2 \\ 0 & -1 \end{bmatrix}, \quad \bar{x} = A_B^{-1} b_B = \begin{bmatrix} 1 & 2 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 5 \\ 0 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$$

$$A_N \bar{x} = \begin{bmatrix} 1 & -1 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} 5 \\ 0 \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix} \not\leq \begin{bmatrix} 2 \\ 7 \end{bmatrix} = b_N.$$

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

$$(PA) \quad \begin{array}{rcl} \max & & -\nu \\ & x_1 & + 2x_2 \leq 5 \\ & x_1 & - x_2 - \nu \leq 2 \\ & x_1 & + 3x_2 \leq 7 \\ & & -x_2 \leq 0 \\ & & -\nu \leq 0 \end{array} \quad (DA) \quad \begin{array}{rcl} \min & 5y_1 & + 2y_2 + 7y_3 \\ & y_1 & + y_2 + y_3 = 0 \\ & 2y_1 & - y_2 + 3y_3 - y_4 = 0 \\ & & y_2 + y_5 = 1 \\ & y_1 & , y_2 , y_3 , y_4 , y_5 \geq 0 \end{array}$$

La base $\mathcal{B}^1 = B \cup J = \{1, 2, 4\}$ è primale ammissibile per costruzione, possiamo quindi applicare l'algoritmo del simplexso primale a partire da essa.

it. 1) $\mathcal{B}^1 = \{1, 2, 4\}$, $\mathcal{N}^1 = \{3, 5\}$:

$$\mathcal{A}_{\mathcal{B}^1} = \begin{bmatrix} 1 & 2 & 0 \\ 1 & -1 & -1 \\ 0 & -1 & 0 \end{bmatrix}, \quad \mathcal{A}_{\mathcal{B}^1}^{-1} = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 0 & -1 \\ 1 & -1 & 3 \end{bmatrix}, \quad \begin{bmatrix} \bar{x}^1 \\ \bar{\nu}^1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 0 & -1 \\ 1 & -1 & 3 \end{bmatrix} \begin{bmatrix} 5 \\ 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \\ 3 \end{bmatrix}$$

$$\mathcal{A}_{\mathcal{N}^1} \begin{bmatrix} \bar{x}^1 \\ \bar{\nu}^1 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} 5 \\ 0 \\ 3 \end{bmatrix} = \begin{bmatrix} 5 \\ -3 \end{bmatrix} \leq \begin{bmatrix} 7 \\ 0 \end{bmatrix} = b_{\mathcal{N}^1}$$

$$\bar{y}_{\mathcal{B}^1} = [0, 0, -1] \begin{bmatrix} 1 & 0 & 2 \\ 0 & 0 & -1 \\ 1 & -1 & 3 \end{bmatrix} = [-1, 1, -3] \quad , \quad \bar{y}^1 = [-1, 1, 0, -3, 0] \not\geq 0.$$

Pertanto, per la regola anticiclo di Bland $h = 1$ con $\mathcal{B}^1(h) = 1$. Per individuare l'indice entrante calcoliamo

$$\xi^1 = \begin{bmatrix} -1 \\ 0 \\ -1 \end{bmatrix}, \quad \mathcal{A}_{\mathcal{N}^1} \xi^1 = \begin{bmatrix} 1 & 3 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \\ -1 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \quad \bar{\lambda} = \lambda_5 = 3, \quad k = 5.$$

it. 2) $\mathcal{B}^2 = \{5, 2, 4\}$, $\mathcal{N}^2 = \{1, 3\}$:

$$\mathcal{A}_{\mathcal{B}^2} = \begin{bmatrix} 0 & 0 & -1 \\ 1 & -1 & -1 \\ 0 & -1 & 0 \end{bmatrix}, \quad \mathcal{A}_{\mathcal{B}^2}^{-1} = \begin{bmatrix} -1 & 1 & -1 \\ 0 & 0 & -1 \\ -1 & 0 & 0 \end{bmatrix}, \quad \begin{bmatrix} \bar{x}^2 \\ \bar{\nu}^2 \end{bmatrix} = \begin{bmatrix} -1 & 1 & -1 \\ 0 & 0 & -1 \\ -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix}$$

$$\mathcal{A}_{\mathcal{N}^2} \begin{bmatrix} \bar{x}^2 \\ \bar{\nu}^2 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 0 \\ 1 & 3 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix} \leq \begin{bmatrix} 5 \\ 7 \end{bmatrix} = b_{\mathcal{N}^2}$$

$$\bar{y}_{\mathcal{B}^2} = [0, 0, -1] \begin{bmatrix} -1 & 1 & -1 \\ 0 & 0 & -1 \\ -1 & 0 & 0 \end{bmatrix} = [1, 0, 0], \quad \bar{y}^2 = [0, 0, 0, 0, 1] \geq 0.$$

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:

$$A_B = A_B^{-1} = \begin{bmatrix} 1 & -1 \\ 0 & -1 \end{bmatrix}, \quad \bar{x} = A_B^{-1} b_B = \begin{bmatrix} 1 & -1 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \end{bmatrix} = \bar{x}^2$$

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 3.10. Risolvere il problema (P) dell'esempio precedente utilizzando l'algoritmo del semplice primale a partire dalla base $B = \{2, 4\}$.

3.3.2 L'algoritmo del Simpleso Duale

L'algoritmo del semplice 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 *simpleso duale*, presentato nel seguito.

Il simpleso duale altro non è che il simpleso 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 Simpleso 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

$$(D') \quad \max \{ c'y : A'y \leq b' \} \quad \text{dove} \quad c' = -b^T, \quad A' = \begin{bmatrix} A^T \\ -A^T \\ -I \end{bmatrix}, \quad b' = \begin{bmatrix} c \\ -c \\ 0 \end{bmatrix}. \quad (3.22)$$

Poiché per ogni \bar{y} ammissibile deve risultare $\bar{y}A^i = c_i$ per ogni $i = 1, \dots, n$, 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 $N \subset \{1, \dots, m\}$ con $|N| = m - n$ l'insieme degli indici che caratterizzano quest'ultimo blocco di vincoli e $B = \{1, \dots, m\} \setminus N$ 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

$$A'_{B'} = \begin{bmatrix} \pm A_B^T & \pm A_N^T \\ 0 & -I_N \end{bmatrix} \quad b'_{B'} = \begin{bmatrix} \pm c \\ 0 \end{bmatrix}$$

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 $[\bar{y}_B, \bar{y}_N] = [cA_B^{-1}, 0]$.

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 Simplexso 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 degenerare 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 simplexso primale.

Forniamo adesso una descrizione formale dell’algoritmo.

```

procedure (  $B, \bar{x}, \bar{y}, stato$  ) = Simplexso_Duale(  $A, b, c, B$  ) {
  for(  $stato = \text{“”}$  ; ) { // invariante:  $B$  duale ammissibile
     $\bar{x} = A_B^{-1} b_B$ ;  $\bar{y} = [\bar{y}_B, \bar{y}_N] = [cA_B^{-1}, 0]$ ;
    if(  $A_N \bar{x} \leq b_N$  ) then {  $stato = \text{“ottimo”}$ ; break; }
     $k = \min\{i \in N : A_i \bar{x} > b_i\}$ ;  $\eta_B = A_k A_B^{-1}$ ;
    if(  $\eta_B \leq 0$  ) then {  $stato = \text{“P.vuoto”}$ ; break; }
     $\bar{\theta} = \min\{\theta_i = \bar{y}_i / \eta_i : \eta_i > 0, i \in B\}$ ;
     $h = \min\{i \in B : \theta_i = \bar{\theta}\}$ ;  $B = B \cup \{k\} \setminus \{h\}$ ;
  }
}

```

Procedura 3.2: Simplexso Duale

L’algoritmo del simplexso 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:

$$d_i = \begin{cases} -\eta_i & \text{se } i \in B \\ 1 & \text{se } i = k \\ 0 & \text{altrimenti} \end{cases} \quad (3.23)$$

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 degenerare: infatti ciò può accadere solamente se esiste un indice $i \in B$ tale che $\bar{y}_i = 0$, ossia se B è duale degenerare. 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

$$(\eta_B A_B x = A_k x) \geq \eta_B b_B \quad \text{che è incompatibile con} \quad A_k x \leq b_k$$

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 3.34. Applicazione del simplesso duale per via algebrica

Si consideri la coppia asimmetrica di problemi duali

$$\begin{array}{ll}
 \max & x_1 \\
 (P) & \begin{array}{l} x_1 + 2x_2 \leq 6 \\ x_1 - 2x_2 \leq 6 \\ 2x_1 + x_2 \leq 4 \\ 2x_1 - x_2 \leq 4 \\ -x_1 \leq 0 \end{array} \\
 \min & 6y_1 + 6y_2 + 4y_3 + 4y_4 \\
 (D) & \begin{array}{l} y_1 + y_2 + 2y_3 + 2y_4 - y_5 = 1 \\ 2y_1 - 2y_2 + y_3 - y_4 = 0 \\ y_1, y_2, y_3, y_4, y_5 \geq 0 \end{array}
 \end{array}$$

e la base $B^1 = \{1, 2\}$. Essendo

$$A_{B^1} = \begin{bmatrix} 1 & 2 \\ 1 & -2 \end{bmatrix} , \quad A_{B^1}^{-1} = \begin{bmatrix} 1/2 & 1/2 \\ 1/4 & -1/4 \end{bmatrix} , \quad b_{B^1} = \begin{bmatrix} 6 \\ 6 \end{bmatrix}$$

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

$$A_{N^1} \bar{x}^1 = \begin{bmatrix} 2 & 1 \\ 2 & -1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 6 \\ 0 \end{bmatrix} = \begin{bmatrix} 12 \\ 12 \\ -6 \end{bmatrix} \not\leq \begin{bmatrix} 4 \\ 4 \\ 0 \end{bmatrix} = b_{N^1} .$$

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

$$\bar{y}_{B^1} = cA_{B^1}^{-1} = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 1/2 & 1/2 \\ 1/4 & -1/4 \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 \end{bmatrix} \quad , \quad \bar{y}^1 = \begin{bmatrix} 1/2 & 1/2 & 0 & 0 & 0 \end{bmatrix} \geq 0 \quad .$$

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

$$\eta_{B^1} = A_3 A_{B^1}^{-1} = \begin{bmatrix} 2 & 1 \end{bmatrix} \begin{bmatrix} 1/2 & 1/2 \\ 1/4 & -1/4 \end{bmatrix} = \begin{bmatrix} 5/4 & 3/4 \end{bmatrix}$$

$$\bar{\theta} = \min \left\{ \frac{\bar{y}_1^1}{\eta_1^1}, \frac{\bar{y}_2^1}{\eta_2^1} \right\} = \min \left\{ \frac{1/2}{5/4}, \frac{1/2}{3/4} \right\} = \min \left\{ \frac{2}{5}, \frac{2}{3} \right\} = \frac{2}{5} \quad ;$$

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

$$A_{B^2} = \begin{bmatrix} 2 & 1 \\ 1 & -2 \end{bmatrix} \quad , \quad A_{B^2}^{-1} = \begin{bmatrix} 2/5 & 1/5 \\ 1/5 & -2/5 \end{bmatrix} \quad , \quad b_{B^2} = \begin{bmatrix} 4 \\ 6 \end{bmatrix} \quad , \quad \bar{x}^2 = \begin{bmatrix} 2/5 & 1/5 \\ 1/5 & -2/5 \end{bmatrix} \begin{bmatrix} 4 \\ 6 \end{bmatrix} = \begin{bmatrix} 14/5 \\ -8/5 \end{bmatrix}$$

$$\bar{y}_{B^2} = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 2/5 & 1/5 \\ 1/5 & -2/5 \end{bmatrix} = \begin{bmatrix} 2/5 & 1/5 \end{bmatrix} \quad , \quad \bar{y}^2 = \begin{bmatrix} 0 & 1/5 & 2/5 & 0 & 0 \end{bmatrix} \quad .$$

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

$$\eta_{B^2} = \begin{bmatrix} 2 & -1 \end{bmatrix} \begin{bmatrix} 2/5 & 1/5 \\ 1/5 & -2/5 \end{bmatrix} = \begin{bmatrix} 3/5 & 4/5 \end{bmatrix}$$

$$\bar{\theta} = \min \left\{ \frac{\bar{y}_3^2}{\eta_3^2}, \frac{\bar{y}_2^2}{\eta_2^2} \right\} = \min \left\{ \frac{2/5}{3/5}, \frac{1/5}{4/5} \right\} = \min \left\{ \frac{2}{3}, \frac{1}{4} \right\} = \frac{1}{4} \quad ,$$

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

$$A_{B^3} = \begin{bmatrix} 2 & 1 \\ 2 & -1 \end{bmatrix} \quad , \quad A_{B^3}^{-1} = \begin{bmatrix} 1/4 & 1/4 \\ 1/2 & -1/2 \end{bmatrix} \quad , \quad b_{B^3} = \begin{bmatrix} 4 \\ 4 \end{bmatrix} \quad , \quad \bar{x}^3 = \begin{bmatrix} 1/4 & 1/4 \\ 1/2 & -1/2 \end{bmatrix} \begin{bmatrix} 4 \\ 4 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$$

$$\bar{y}_{B^3} = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} 1/4 & 1/4 \\ 1/2 & -1/2 \end{bmatrix} = \begin{bmatrix} 1/4 & 1/4 \end{bmatrix} \quad , \quad \bar{y}^3 = \begin{bmatrix} 0 & 0 & 1/4 & 1/4 & 0 \end{bmatrix} \quad .$$

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 3.35. Simplexso 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.

$$\begin{array}{rclcl} \max & x_1 & + & 2x_2 & \\ & & & x_2 & \leq 4 \\ & x_1 & - & 2x_2 & \leq 1 \\ & 2x_1 & + & x_2 & \leq 4 \\ & x_1 & & & \leq 2 \\ & -x_1 & + & x_2 & \leq -2 \end{array}$$

$$\text{it. 1) } B = \{1, 4\}: A_B = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad , \quad A_B^{-1} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad \bar{x} = A_B^{-1} b_B = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 4 \\ 2 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$

$$\bar{y}_B = cA_B^{-1} = [1, 2] \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = [2, 1], \quad \bar{y}_N = 0, \quad \bar{y} = [2, 0, 0, 1, 0]$$

$$A_N \bar{x} = \begin{bmatrix} 1 & -2 \\ 2 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 4 \end{bmatrix} = \begin{bmatrix} -6 \\ 8 \\ 2 \end{bmatrix} \not\leq b_N = \begin{bmatrix} 1 \\ 4 \\ -2 \end{bmatrix}$$

$k = \min \{i \in N : A_i \bar{x} > b_i\} = \min \{3, 5\} = 3$ [regola anticiclo di Bland]

$$\eta_B = A_k A_B^{-1} = [2, 1] \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = [1, 2]$$

$$\bar{\theta} = \min \{\bar{y}_i / \eta_i : i \in B, \eta_i > 0\} = \min \{\bar{y}_1 / \eta_1, \bar{y}_4 / \eta_4\} = \min \{2/1, 1/2\} = 1/2$$

$$h = \min \{i \in B : \eta_i > 0, \bar{\theta} = \bar{y}_i / \eta_i\} = \min \{4\} = 4$$

$$B = B \setminus \{u\} \cup \{k\} = \{1, 4\} \setminus \{4\} \cup \{3\} = \{1, 3\}$$

it. 2) $B = \{1, 3\}$: $A_B = \begin{bmatrix} 0 & 1 \\ 2 & 1 \end{bmatrix}$, $A_B^{-1} = \begin{bmatrix} -1/2 & 1/2 \\ 1 & 0 \end{bmatrix}$, $\bar{x} = \begin{bmatrix} -1/2 & 1/2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 4 \\ 4 \end{bmatrix} = \begin{bmatrix} 0 \\ 4 \end{bmatrix}$

$$\bar{y}_B = [1, 2] \begin{bmatrix} -1/2 & 1/2 \\ 1 & 0 \end{bmatrix} = [3/2, 1/2], \quad \bar{y}_N = 0, \quad \bar{y} = [3/2, 0, 1/2, 0, 0]$$

$$A_N \bar{x} = \begin{bmatrix} 1 & -2 \\ 1 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 4 \end{bmatrix} = \begin{bmatrix} -8 \\ 0 \\ 4 \end{bmatrix} \not\leq b_N = \begin{bmatrix} 1 \\ 2 \\ -2 \end{bmatrix}, \quad k = 5 \text{ (unico vincolo violato)}$$

$$\eta_B = [-1, 1] \begin{bmatrix} -1/2 & 1/2 \\ 1 & 0 \end{bmatrix} = [3/2, -1/2], \quad h = 1 \text{ in quanto } \eta_1 \text{ è l'unico } > 0$$

$$\bar{\theta} = \bar{y}_1 / \eta_1 = (3/2) / (3/2) = 1 \text{ (ma } \bar{\theta} \text{ è nei fatti ininfluente)}$$

it. 3) $B = \{3, 5\}$: $A_B = \begin{bmatrix} 2 & 1 \\ -1 & 1 \end{bmatrix}$, $A_B^{-1} = \begin{bmatrix} 1/3 & -1/3 \\ 1/3 & 2/3 \end{bmatrix}$, $\bar{x} = \begin{bmatrix} 1/3 & -1/3 \\ 1/3 & 2/3 \end{bmatrix} \begin{bmatrix} 4 \\ -2 \end{bmatrix} = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$

$$\bar{y}_B = [1, 2] \begin{bmatrix} 1/3 & -1/3 \\ 1/3 & 2/3 \end{bmatrix} = [1, 1], \quad \bar{y}_N = 0, \quad \bar{y} = [0, 0, 1, 0, 1]$$

$$A_N \bar{x} = \begin{bmatrix} 0 & 1 \\ 1 & -2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ 2 \end{bmatrix} \not\leq b_N = \begin{bmatrix} 4 \\ 1 \\ 2 \end{bmatrix}, \quad k = 2 \text{ (unico vincolo violato)}$$

$$\eta_B = [1, -2] \begin{bmatrix} 1/3 & -1/3 \\ 1/3 & 2/3 \end{bmatrix} = [-1/3, -5/3] \leq 0$$

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

$$d = [-\eta_B, 1, 0] = [0, 1, 1/3, 0, 5/3]$$

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

$$y(\theta) = \bar{y} + \theta d = [0, 0, 1, 0, 1] + \theta [0, 1, 1/3, 0, 5/3] = [0, \theta, 1 + \theta/3, 0, 1 + 5\theta/3]$$

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

$$\eta_B A_B x = A_k A_B^{-1} A_B x = A_k x = x_1 - 2x_2 \geq \eta_B b_B = [-1/3, -5/3] \begin{bmatrix} 4 \\ -2 \end{bmatrix} = 2$$

(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 simpleso primale è possibile uno svolgimento geometrico, o grafico, del simpleso duale per problemi di piccole dimensioni. Questo è leggermente meno ovvia che nel caso del simpleso 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 / η_i e \bar{y}_j / η_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 3.8, 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 $\text{cono}(\{A_i, A_k\})$ e $\text{cono}(\{A_j, A_k\})$ (si veda ancoa la Figura 3.8): poiché il sempliceo duale visita solamente duali ammissibili, deve necessariamente uscire di base l'indice $p \in \{i, j\}$ tale che $c \notin \text{cono}(\{A_p, A_k\})$.

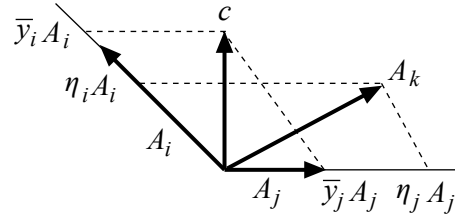


Figura 3.8: Il “caso difficile” del calcolo di $\bar{\theta}$

Esempio 3.36. Esecuzione grafica del sempliceo duale

Si vuole risolvere graficamente il problema di PL in figura qui accanto, utilizzando l'algoritmo del sempliceo 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 sempliceo 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 3.9 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.

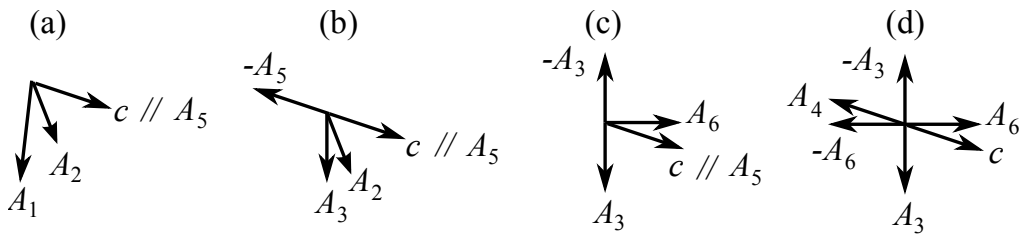
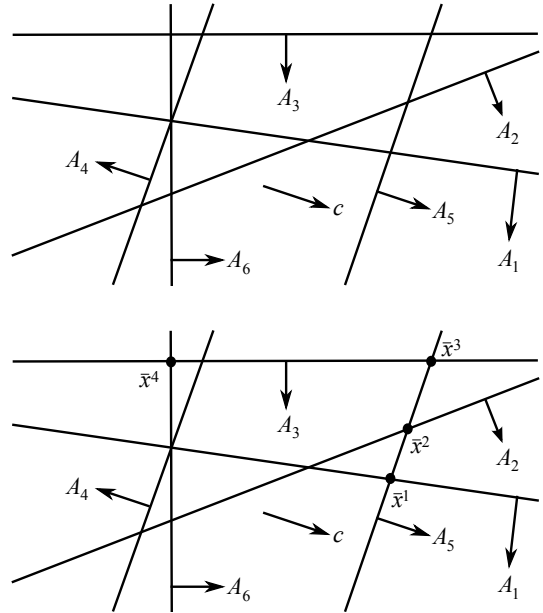


Figura 3.9: Ingrandimento dei dettagli rilevanti nell'esecuzione geometrica

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 $A_2 \in \text{cono}(\{A_1, A_5\})$, come mostrato in Figura 3.9.(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, poiché $c \in \text{cono}(\{A_2, A_5\})$ ma $c \notin \text{cono}(\{A_1, A_2\})$, 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 degenerare poiché $B = I(\bar{x}^2)$. Poiché $A_3 \in \text{cono}(\{A_2, -A_5\})$, come mostrato in Figura 3.9.(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 degenerare, ed è anche primale non degenerare poiché $B = I(\bar{x}^3)$. Poiché $A_6 \in \text{cono}(\{-A_3, A_5\})$, come mostrato in Figura 3.9.(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 degenerare, poiché ancora $B = I(\bar{x}^4)$. Poiché $A_4 \in \text{cono}(\{-A_3, -A_6\})$, come mostrato in Figura 3.9.(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. (3.23)), 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 3.11. Si risolva geometricamente il problema dell'Esempio 3.34 mediante l'algoritmo semplice 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 Simplexso 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)

$$(DA) \quad \min \{yb + wM : yA + w = c, \quad y \geq 0, \quad w \geq 0\} \quad (PA) \quad \max \{cx : Ax \leq b, \quad x \leq M\}$$

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” $B = \{m+1, \dots, m+n\}$ 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 simplexso 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 simplexso 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 3.37. Individuazione di una base duale ammissibile

Si consideri la coppia di problemi duali:

$$\begin{array}{ll}
 \max & 4x_1 + 2x_2 \\
 (P) & \begin{array}{l} -x_1 + 4x_2 \leq 2 \\ x_1 - 2x_2 \leq -3 \\ -x_1 + x_2 \leq -1 \end{array} \\
 \min & 2y_1 - 3y_2 - y_3 \\
 (D) & \begin{array}{l} -y_1 + y_2 - y_3 = 4 \\ 4y_1 - 2y_2 + y_3 = 2 \\ y_1, y_2, y_3 \geq 0 \end{array}
 \end{array}$$

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 $M = [20, 20]$:

$$\begin{array}{ll}
 \min & 2y_1 - 3y_2 - y_3 + 20w_1 + 20w_2 \\
 (DA) & \begin{array}{l} -y_1 + y_2 - y_3 + w_1 = 4 \\ 4y_1 - 2y_2 + y_3 + w_2 = 2 \\ y_1, y_2, y_3, w_1, w_2 \geq 0 \end{array} \\
 \max & 4x_1 + 2x_2 \\
 (PA) & \begin{array}{l} -x_1 + 4x_2 \leq 2 \\ x_1 - 2x_2 \leq -3 \\ -x_1 + x_2 \leq -1 \\ x_1 \leq 20 \\ x_2 \leq 20 \end{array}
 \end{array}$$

La base di partenza “artificiale” $B^1 = \{4, 5\}$ è quella relativa alle variabili ausiliarie w_1 e w_2 .

it. 1): $B^1 = \{4, 5\}$, $A_{B^1} = A_{B^1}^{-1} = I$, $\bar{x}^1 = A_{B^1}^{-1}b_{B^1} = b_{B^1} = [20, 20]$, $\bar{y}^1 = [0, 0, 0]$, $\bar{w}^1 = cA_{B^1}^{-1} = [4, 2]$

$$A_{N^1}\bar{x}^1 = \begin{bmatrix} -1 & 4 \\ 1 & -2 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 20 \\ 20 \end{bmatrix} = \begin{bmatrix} 60 \\ -20 \\ 0 \end{bmatrix} \not\leq \begin{bmatrix} 2 \\ -3 \\ -1 \end{bmatrix}$$

$$k = \min \{i \in N : A_i\bar{x} > b_i\} = \min \{1, 3\} = 1$$

$$\eta_{B^1} = A_k A_{B^1}^{-1} = A_1 = \begin{bmatrix} -1 & 4 \end{bmatrix}, \eta_4 = -1 < 0 \text{ e } \eta_5 = 4 > 0 \text{ quindi l'indice uscente è } h = B^1(2) = 5$$

$$B^2 = B^1 \setminus \{5\} \cup \{1\} = \{4, 1\}.$$

$$\text{it. 2): } B^2 = \{4, 1\}, A_{B^2} = \begin{bmatrix} 1 & 0 \\ -1 & 4 \end{bmatrix}, A_{B^2}^{-1} = \begin{bmatrix} 1 & 0 \\ 1/4 & 1/4 \end{bmatrix}, \bar{x}^2 = \begin{bmatrix} 20 \\ 11/2 \end{bmatrix}$$

$$[\bar{y}^2, \bar{w}^2]_{B^2} = cA_{B^2}^{-1} = [4, 2] \begin{bmatrix} 1 & 0 \\ 1/4 & 1/4 \end{bmatrix} = [9/2, 1/2], [\bar{y}^2, \bar{w}^2] = [1/2, 0, 0, 9/2, 0]$$

$$A_{N^2}\bar{x}^2 = \begin{bmatrix} 9 \\ -29/2 \\ 11/2 \end{bmatrix} \not\leq \begin{bmatrix} -3 \\ -1 \\ 20 \end{bmatrix} \implies k = 2 \implies \eta_{B^2} = [1/2, -1/2] \implies h = 4$$

$$\text{it. 3): } B^3 = \{2, 1\}, A_{B^3} = \begin{bmatrix} 1 & -2 \\ -1 & 4 \end{bmatrix}, A_{B^3}^{-1} = \begin{bmatrix} 2 & 1 \\ 1/2 & 1/2 \end{bmatrix}, \bar{x}^3 = \begin{bmatrix} -4 \\ -1/2 \end{bmatrix}$$

$$[\bar{y}^3, \bar{w}^3]_{B^3} = [9, 5], [\bar{y}^3, \bar{w}^3] = [5, 9, 0, 0, 0]$$

$$A_{N^3}\bar{x}^3 = \begin{bmatrix} 7/2 \\ -4 \\ -1/2 \end{bmatrix} \not\leq \begin{bmatrix} -1 \\ 20 \\ 20 \end{bmatrix} \implies k = 3 \implies \eta_{B^3} = [-3/2, -1/2]$$

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 3.12. 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 3.13. 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. (3.22)).

Lo stesso principio può essere applicato in molti altri casi, rendendo molto più efficienti le implemen-

tazioni 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 variante 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

$$(c + \lambda\nu)A_B^{-1}u_{B(i)} \geq 0 \quad i \in B ,$$

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

$$\begin{aligned} \lambda &\geq \frac{-cA_B^{-1}u_{B(i)}}{\nu A_B^{-1}u_{B(i)}} = -\frac{\bar{y}_i}{\nu A_B^{-1}u_{B(i)}} && \text{se } \nu A_B^{-1}u_{B(i)} > 0 , \\ \lambda &\leq \frac{-cA_B^{-1}u_{B(i)}}{\nu A_B^{-1}u_{B(i)}} = -\frac{\bar{y}_i}{\nu A_B^{-1}u_{B(i)}} && \text{se } \nu A_B^{-1}u_{B(i)} < 0 ; \end{aligned}$$

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 3.38. Variazione del vettore c

Si consideri il problema dell'Esempio 3.34; al termine dell'applicazione del semplice 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

$$\text{a) } \nu A_B^{-1}u_{B(1)} = 3/4 > 0 \implies \lambda \geq -\frac{\bar{y}_3}{3/4} = -\frac{1/4}{3/4} = -1/3$$

$$\text{b) } \nu A_B^{-1}u_{B(2)} = -1/4 < 0 \implies \lambda \leq \frac{\bar{y}_4}{-(-1/4)} = \frac{1/4}{1/4} = 1$$

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 3.14. 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 3.15. 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 3.39. 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 base 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 .

Dalla soluzione ottima, nota, per $\varepsilon = 1$ otteniamo immediatamente che $z(1) = 4$. Appliciamo adesso l'analisi di sensitività alla base $B = \{1, 2\}$ per determi-

nare 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

$$\bar{y}_B(\varepsilon) = \begin{bmatrix} \varepsilon & 1 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 - \varepsilon & \varepsilon \end{bmatrix} \geq 0 .$$

$$\begin{array}{rcll} \max & \varepsilon x_1 & + & x_2 \\ & & & x_2 \leq 2 \\ & x_1 & + & x_2 \leq 4 \\ & -x_1 & + & 2x_2 \leq 2 \\ & -x_1 & + & x_2 \leq 1 \\ & -x_1 & & \leq 1 \end{array}$$

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

$$\xi = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \quad A_N \xi = \begin{bmatrix} -1 & 2 \\ -1 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = \begin{bmatrix} -3 \\ -2 \\ -1 \end{bmatrix} < 0$$

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

$$\xi = \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \quad A_N \xi = \begin{bmatrix} -1 & 2 \\ -1 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

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

$$\bar{y}_B = \begin{bmatrix} \varepsilon & 1 \end{bmatrix} \begin{bmatrix} -2 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} \varepsilon + 1 & -2\varepsilon - 1 \end{bmatrix},$$

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 3.40. Variazione del vettore b

Si consideri la seguente coppia di problemi di PL

$$\begin{array}{ll}
 \max & 3x_1 + x_2 \\
 (P) & \begin{array}{l} -2x_1 + x_2 \leq 1 \\ x_1 - 2x_2 \leq -4 \\ x_1 + x_2 \leq 14 \\ x_1 \leq 8 \\ - x_2 \leq -4 \end{array} \\
 \min & y_1 - 4y_2 + 14y_3 + 8y_4 - 4y_5 \\
 (D) & \begin{array}{l} -2y_1 + y_2 + y_3 + y_4 = 3 \\ y_1 - 2y_2 + y_3 - y_5 = 1 \\ y_1, y_2, y_3, y_4, y_5 \geq 0 \end{array}
 \end{array}$$

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:

$$A_4 \bar{x}^1 = [1, 0] \begin{bmatrix} 8 \\ 6 \end{bmatrix} = 8 > 7 ;$$

poiché ciò non accade, la base B^1 non è primale ammissibile. Si applica allora il simplesso duale partendo da B^1 :

$$A_{B^1} = \begin{bmatrix} 1 & -2 \\ 1 & 1 \end{bmatrix}, \quad A_{B^1}^{-1} = \begin{bmatrix} 1/3 & 2/3 \\ -1/3 & 1/3 \end{bmatrix}$$

e l'indice entrante è ovviamente $k = 4$; si ha quindi

$$\eta_{B^1} = A_k A_{B^1}^{-1} = [1, 0] \begin{bmatrix} 1/3 & 2/3 \\ -1/3 & 1/3 \end{bmatrix} = [1/3, 2/3] .$$

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

$$\bar{\theta} = \min \left\{ \frac{\bar{y}_2^1}{\eta_2^1}, \frac{\bar{y}_3^1}{\eta_3^1} \right\} = \min \left\{ \frac{2/3}{1/3}, \frac{7/3}{2/3} \right\} = \min \left\{ 2, \frac{7}{2} \right\} = 2 ;$$

quindi $h = 2$ e si ottiene la nuova base $B^2 = \{4, 3\}$. Alla successiva iterazione si ha quindi

$$\begin{aligned}
 A_{B^2} &= \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \quad A_{B^2}^{-1} = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix}, \quad \bar{x}^2 = \begin{bmatrix} 7 \\ 7 \end{bmatrix}, \quad \bar{y}_{B^2}^2 = [2, 1], \quad \bar{y}^2 = [0, 0, 1, 2, 0] \\
 A_{N^2} \bar{x}^2 &= \begin{bmatrix} -2 & 1 \\ 1 & -2 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 7 \\ 7 \end{bmatrix} = \begin{bmatrix} -7 \\ -7 \\ -7 \end{bmatrix} < \begin{bmatrix} 1 \\ -4 \\ -4 \end{bmatrix} = b_{N^2} .
 \end{aligned}$$

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 3.16. 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 3.17. Determinare, per la coppia di problemi dell'Esempio 3.40, 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 3.18. 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

$$A' = \begin{bmatrix} A & A^{n+1} \\ 0 & -1 \end{bmatrix} \quad \text{e la sottomatrice} \quad A'_{B'} = \begin{bmatrix} A_B & A_B^{n+1} \\ 0 & -1 \end{bmatrix}$$

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 3.19. 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 ottenendo un problema equivalente la cui matrice dei vincoli è

$$A'' = \begin{bmatrix} A & A^{n+1} & -A^{n+1} \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{e la cui sottomatrice} \quad A''_{B''} = \begin{bmatrix} A_B & A_B^{n+1} & -A_B^{n+1} \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

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 Simpleso Primale.

Esercizio 3.20. 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 3.41. Aggiunta di una variabile

Si consideri la coppia di problemi di PL dell'Esempio 3.40, 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

$\begin{array}{rcll} \max & 3x_1 & + & x_2 & - & 2x_3 & & & \\ & -2x_1 & + & x_2 & + & x_3 & \leq & 1 & \\ & x_1 & - & 2x_2 & - & 3x_3 & \leq & -4 & \\ (P') & x_1 & + & x_2 & & & \leq & 14 & \\ & x_1 & & & + & x_3 & \leq & 8 & \\ & & - & x_2 & + & 2x_3 & \leq & -4 & \\ & & & & - & x_3 & \leq & 0 & \end{array}$	$\begin{array}{rcll} \min & y_1 & - & 4y_2 & + & 14y_3 & + & 8y_4 & - & 4y_5 & & & \\ & -2y_1 & + & y_2 & + & y_3 & + & y_4 & & & = & 3 & \\ (D') & y_1 & - & 2y_2 & + & y_3 & & & - & y_5 & = & 1 & \\ & y_1 & - & 3y_2 & & & + & y_4 & + & 2y_5 & - & y_6 & = & -2 & \\ & y_1 & , & y_2 & , & y_3 & , & y_4 & , & y_5 & , & y_6 & \geq & 0 & \end{array}$
---	---

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:

$$A_B = \begin{bmatrix} 1 & -2 \\ 1 & 1 \end{bmatrix}, \quad A_B^{-1} = \begin{bmatrix} 1/3 & 2/3 \\ -1/3 & 1/3 \end{bmatrix}.$$

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

$$A'_{B'} = \begin{bmatrix} A_B & A_B^{n+1} \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 & -2 & -3 \\ 1 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \quad A'^{-1}_{B'} = \begin{bmatrix} 1/3 & 2/3 & -1 \\ -1/3 & 1/3 & 1 \\ 0 & 0 & -1 \end{bmatrix}$$

e quindi è facile verificare che la corrispondente soluzione primale di base

$$\bar{x}' = A'^{-1}_{B'} b_{B'} = \begin{bmatrix} 1/3 & 2/3 & -1 \\ -1/3 & 1/3 & 1 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} -4 \\ 14 \\ 0 \end{bmatrix} = \begin{bmatrix} 8 \\ 6 \\ 0 \end{bmatrix}$$

è esattamente $\bar{x}' = [\bar{x}, 0]$. La soluzione duale di base è data da

$$\bar{y}'_{B'} = c' A'^{-1}_{B'} = [3, 1, -2] \begin{bmatrix} 1/3 & 2/3 & -1 \\ -1/3 & 1/3 & 1 \\ 0 & 0 & -1 \end{bmatrix} = [2/3, 7/3, 0], \quad \bar{y}' = [0, 2/3, 7/3, 0, 0, 0].$$

Si noti che si è ottenuto $\bar{y}'_6 = 0$; infatti, per verifica, applicando la formula fornita si ottiene:

$$\bar{y}'_6 = \bar{y}_B A_B^6 - c_6 = [2/3, 7/3] \begin{bmatrix} -3 \\ 0 \end{bmatrix} + 2 = -2 + 2 = 0.$$

Si è ottenuta una soluzione duale ammissibile e degenera; 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

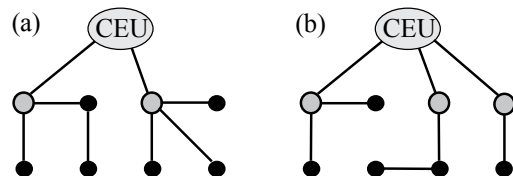
L'*Ottimizzazione Combinatoria* (*OC*) studia i problemi di ottimizzazione in cui l'insieme ammissibile è definito in termini di strutture combinatorie, tra le quali svolgono sicuramente un ruolo di rilievo i grafi. La caratteristica fondamentale di tali problemi è quella di avere insiemi ammissibili *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.



Esempio 4.2. 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

$$\begin{aligned} \min \quad & \sum_{(i,j) \in A} c_{ij} x_{ij} \\ & \sum_{(j,i) \in BS(i)} x_{ji} - \sum_{(i,j) \in FS(i)} x_{ij} = b_i \quad i \in N \end{aligned} \quad (4.1)$$

$$\sum_{(i,j) \in A} l_{ij} x_{ij} \leq L \quad (4.2)$$

$$x_{ij} \in \{0, 1\} \quad (i, j) \in A$$

dove $b_i = -1$ se $i = r$, $b_i = 1$ se $i = t$, e $b_i = 0$ altrimenti. Anche in questo caso il singolo vincolo (4.2) risulta “complicante”; l’introduzione del vincolo rende il problema \mathcal{NP} -arduo.

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 irrisolvibili. 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 prestazioni 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.

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*.

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

$$(PLI) \quad \max\{cx : Ax \leq b, x \in \mathbb{Z}^n\}.$$

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

$$(PLM) \quad \max\{c'x' + c''x'' : A'x' + A''x'' \leq b, x' \in \mathbb{Z}^n\}.$$

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*. 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

$$\bar{\mathcal{F}} = \{x \in \mathbb{R}^n : Ax \leq b\},$$

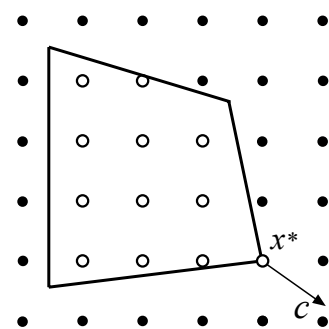


Figura 4.1: *PL* e *PLI*

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 3.2: 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 3.2 in effetti continua a valere: banalmente, poiché x^* è ammissibile, $d = x^* - \bar{x}$ è una direzione ammissibile secondo la definizione meno stringente (con $\bar{\lambda} = 1$). 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: 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.

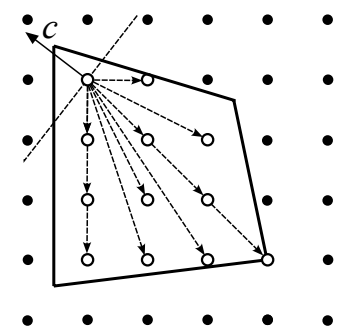


Figura 4.2: Direzioni di crescita

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 $f : \mathbb{R}^n \rightarrow \mathbb{R}$ 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* $\text{epi}(f) = \{(v, x) \in \mathbb{R}^{n+1} : v \geq f(x)\}$. 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 $f : \mathbb{R} \rightarrow \mathbb{R}$ 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 di risolvere all'ottimo il problema. Questi concetti saranno ripresi nel Capitolo ???. 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.

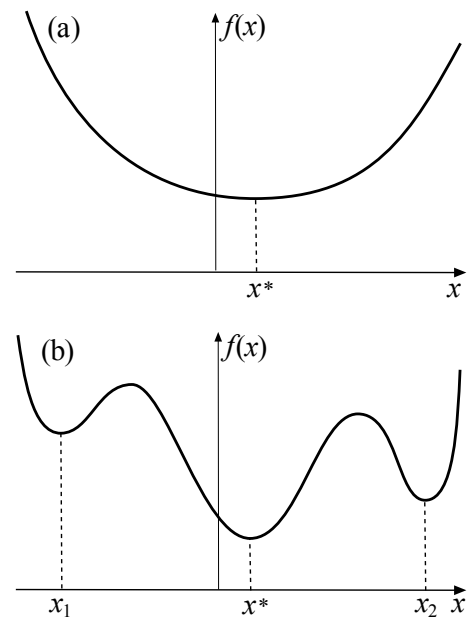


Figura 4.3: Funzioni convesse e non

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 proprietà 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 4.3. 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 4.3, 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

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.

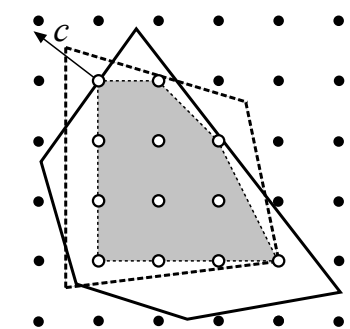


Figura 4.4: Formulazioni diverse

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’involuppo convesso di \mathcal{F}

$$\tilde{\mathcal{F}} = \text{Conv}(\mathcal{F})$$

è un poliedro, cioè può essere rappresentato da un sistema finito di vincoli lineari $\tilde{A}x \leq \tilde{b}$. In Figura 4.4 l’involuppo 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 $\text{rec}(\tilde{\mathcal{F}}) = (0)$, e quindi per il Teorema 3.4 pari all’involuppo 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

$$(\widetilde{RC}) \quad \max\{cx : \tilde{A}x \leq \tilde{b}\}$$

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 $\mathcal{P} = \{x \in \mathbb{R}^n : Ax \leq b\}$, non vuoto e tale che $\text{rango}(A) = n$, 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.

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 2.11) 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 purché 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’involuppo 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'involuppo convesso di un problema esprimibile attraverso una formulazione *PLI* “compatta” abbia un numero esponenziale di vincoli.

Esempio 4.4. 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 $[0, \dots, \pm 1, \dots, 0]$ 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* significativi.

Esempio 4.5. Diverse formulazioni di (MST)

Per il problema (MST), introdotto nell'Esempio 1.6, e discusso in dettaglio nel §2.3, una formulazione *PLI* “compatta” 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 designiamo 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 la tecnica vista nel §1.2.8 per implementare la condizione logica “se c'è flusso su almeno uno degli archi (i, j) e (j, i) allora si deve selezionare il lato $\{i, j\}$ ”, ottenendo la formulazione

$$\begin{aligned}
 \text{(MST)} \quad \min \quad & \sum_{\{i, j\} \in E} c_{ij} x_{ij} \\
 & \sum_{(j, i) \in BS(i)} f_{ji} - \sum_{(i, j) \in FS(i)} f_{ij} = \begin{cases} -(n-1) & i = 1 \\ 1 & i \neq 1 \end{cases} \quad i \in V \\
 & f_{ij} + f_{ji} \leq (n-1) x_{ij} \quad \{i, j\} \in E \\
 & f_{ij} \geq 0 \quad (i, j) \in A \\
 & x_{ij} \in \{0, 1\} \quad \{i, j\} \in E
 \end{aligned}$$

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'involuppo 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

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

Tale formulazione ha solo le m variabili binarie “naturali”, ma ha un numero esponenziale di vincoli. È possibile mostrare che questa formulazione è “esatta”, ossia esprime l'involuppo 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 disequaglianza $dx \leq \delta$ 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 (interi) del problema. Si noti che tutti i vincoli $\tilde{A}_i x \leq \tilde{b}_i$ che definiscono $\tilde{\mathcal{F}}$ sono per definizione disequaglianze 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 disequaglianze 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 $\tilde{\mathcal{F}}$ definisca 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 (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 disequaglianza 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 disequaglianza che separa \bar{x} da $\tilde{\mathcal{F}}$. Una tale disequaglianza 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 (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 *simplexso duale meccanizzato*:

```

procedure ( $\bar{x}, \bar{y}, stato$ ) = Simplexso_Duale_Meccanizzato( $A, b, c, oracolo$ ) {
for(  $stato = \text{""}$  ; ; ) { // invariante:  $A$  invertibile,  $cA^{-1} \geq 0$ 
     $\bar{x} = A^{-1}b$ ;  $\bar{y} = cA^{-1}$ ;
    [ $d, \delta, stato$ ] = oracolo( $\bar{x}$ );
    if(  $stato == \text{"ottimo"}$  )then break;
     $\eta = dA^{-1}$ ;
    if(  $\eta \leq 0$  ) then {  $stato = \text{"P.vuoto"}$ ; break; }
     $\bar{\theta} = \min \{ \theta_i = \bar{y}_i / \eta_i : \eta_i > 0, i = 1, \dots, n \}$ ;
     $h = \min \{ i = 1, \dots, n : \theta_i = \bar{\theta} \}$ ;  $A_h = d$ ;  $b_h = \delta$ ;
  }
}
```

Procedura 4.1: Simplexso Duale Meccanizzato

Il simplexso duale meccanizzato è semplicemente l'algoritmo del simplexso 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—oltretutto 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 $stato = \text{"ottimo"}$ 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 $stato = \text{""}$ e la rappresentazione (d, δ) di un taglio valido. In questo caso l'algoritmo procede come il simplexso 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 semplice duale "normale", ma ci si può aspettare che sia efficiente nella pratica.

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 4.6. Un separatore per l' n -co-cubo

Costruiamo adesso un separatore per la regione ammissibile definita dall' n -co-cubo, come nell'Esempio 4.4. 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

$$\pm x_1 \pm x_2 \dots \pm x_n \leq 1$$

tale che la corrispondente disuguaglianza 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

$$(SEPncc) \quad \max\{\pm \bar{x}_1 \pm \bar{x}_2 \dots \pm \bar{x}_n\}.$$

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 ottimizzare 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 4.7. Un separatore per (MST)

Anche la formulazione "per tagli" di (MST) (cf. l'Esempio 4.5) 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 violato, ossia tale che risulti

$$\sum_{i \in S, j \notin S} \bar{x}_{ij} < 1.$$

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'involuppo convesso $\tilde{\mathcal{F}}$ delle corrispondenti soluzioni intere è anch'esso \mathcal{NP} -arduo.

Ciò significa che le rappresentazioni $\tilde{\mathcal{F}}$ degli involuppi 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 4.8. 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

$$\sum_{i \in I} a_i x_i \leq b, \quad x_i \in [0, 1] \quad i \in I$$

oltre naturalmente a quelli di integralità. È facile vedere che tale formulazione *non* ha la proprietà di integralità considerando la semplicissima istanza

$$\max\{3x_1 + 2x_2 : 2x_1 + 2x_2 \leq 3, \quad [x_1, x_2] \in \{0, 1\}^2\}.$$

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 §?? 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 sì che la regione ammissibile del rilassamento continuo abbia la proprietà di integralità (si ottiene il semplice 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 $\tilde{\mathcal{F}}$, 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*:

$$(CS_{\bar{x}}) \quad \min \left\{ \sum_{i \in I} (1 - \bar{x}_i) z_i : \sum_{i \in I} a_i z_i > b, z_i \in \{0, 1\} \quad i \in I \right\} .$$

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 “ $\dots > b$ ” può quindi essere riscritto “ $\dots \geq b + 1$ ”, 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 faccette 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.

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 2.4): 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 3.8): al termine, gli algoritmi del Simplexso 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. 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.

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 4.9. Auto-riducibilità della Programmazione Lineare 0–1

Come abbiamo visto, la *Programmazione Lineare 0–1*

$$(PL01) \max\{cx : Ax \leq b, x \in \{0, 1\}^n\}$$

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_1^0 . Se invece $z_1^0 < z^*$, 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 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 4.9, si veda il Capitolo 5) 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 5) 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 enumerativi

Tra gli approcci per la determinazione della soluzione esatta di problemi di *OC* \mathcal{NP} -ardui, 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, sfruttando valutazioni inferiori (euristiche) e superiori (rilassamenti) del valore ottimo della funzione obiettivo (cf. §1.1.2) insieme ad opportune regole di dominanza 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 *visitare 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.

5.1 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 finché 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

$$(P) \quad \max\{c(x) : x \in X\},$$

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. Se ciò non accade si deve applicare il meccanismo del “divide et impera”, utilizzando la seguente osservazione:

Lemma 5.1. Sia X_1, X_2, \dots, X_k una suddivisione di X ($X_i \subset X, i = 1, \dots, k, \bigcup_{i=1}^k X_i = X$) e sia

$$(P_i) \quad \max\{c(x) : x \in X_i\}; \quad \text{allora} \\ \max\{\underline{z}(P_i) : i = 1, \dots, k\} \leq \underline{z}(P) \leq \max\{\bar{z}(P_i) : i = 1, \dots, k\}.$$

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 $z(P) \leq \bar{z}$ occorre dimostrare che $z(P_i) \leq \bar{z}$ per ogni $i = 1, \dots, k$. 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, \dots, 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 $x_i, i = 1, \dots, n$: 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 5.1. 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 5.1. 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.

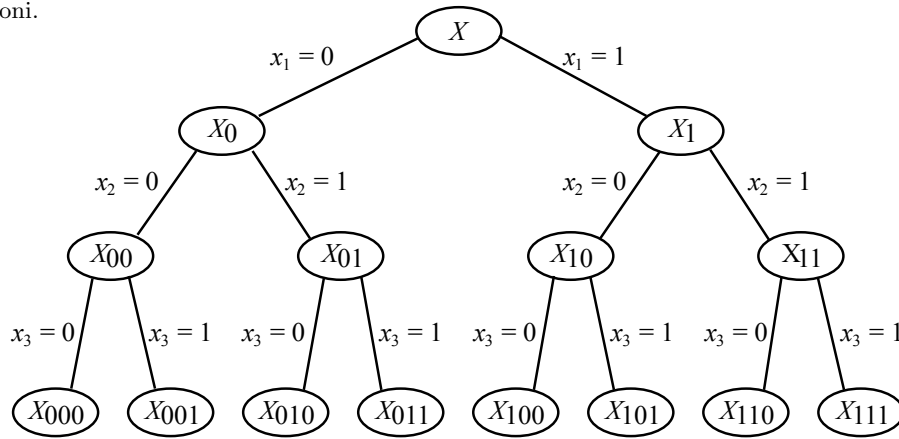


Figura 5.1: Un albero delle decisioni per $\{0, 1\}^3$

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 5.2. 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 5.2(a) è mostrato in Figura 5.2(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).

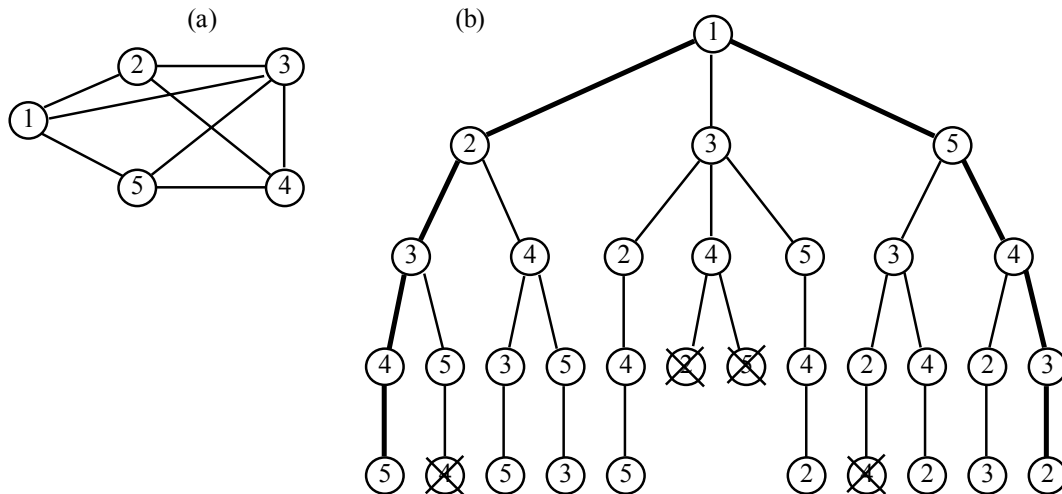


Figura 5.2: Un albero delle decisioni per (TSP)

Questo albero delle decisioni ha alcune caratteristiche rilevanti che è opportuno sottolineare, e che lo differenziano da quello in Figura 5.1:

- 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 sottoinsieme X' corrispondente ad un nodo nell'albero delle decisioni, non sempre 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 $X' \subseteq X$. 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  $z = B\&B(P, \varepsilon)$  {
   $Q = \{(P)\}$ ;  $z = -\infty$ ;
  do {  $(P') = Next(Q)$ ;  $Q = Q \setminus \{(P')\}$ ;
     $(\bar{z}, \underline{z}) = rilassamento(P')$ ;
    if ( $\underline{z} > z$ ) then  $z = \underline{z}$ ;
    if ( $\bar{z} \leq z + \varepsilon$ ) then continue;
     $\underline{z} = euristica(P')$ ;
    if ( $\underline{z} > z$ ) then  $z = \underline{z}$ ;
    if ( $\bar{z} \leq z + \varepsilon$ ) then continue;
     $Q = Q \cup branch(P')$ ;
  } while ( $Q \neq \emptyset$ );
}

```

Procedura 5.1: 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 cioè $\bar{X}' \supseteq X'$ 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.

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 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 (oltretutto, 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 (cf. Lemma 4.1), 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 $\bar{z} > z + \varepsilon$, 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, \dots, 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, \dots, 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 5.1. 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 5.2. 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 5.2. 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 ogniquale volta 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 5.1. \diamond

Corollario 5.1. 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 5.2. \diamond

Si noti che la valutazione superiore \bar{z} indicata dal Corollario 5.1 può essere effettivamente calcolata ad ogni iterazione dell'algoritmo, in quanto tutti i valori di $\bar{z}(P')$ richiesti sono stati effettivamente calcolati. Di conseguenza, l'algoritmo $B\bar{E}B$ 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 perché in effetti $X = \emptyset$, ma l'algoritmo non è stato in grado di dimostrare che questo sia vero.

5.2 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. 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;
- la *regola di branching* utilizzata;
- le eventuali operazioni di *pretrattamento* (*preprocessing*) dei dati utilizzate.

Nel seguito discuteremo ciascuna di queste componenti individualmente.

5.2.1 Rilassamento ed euristica

Come già rimarcato, il rilassamento e l'euristica devono essere quanto più possibile efficaci ed efficienti; 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 §5.4, 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.

5.2.2 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. 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 5.1, 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 5.2 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.

5.2.3 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 regole deve essere compiuta con oculatezza. In generale, affinché l'algoritmo termini occorre solamente che la regola sia *completa*, ossia tale che $X' = X'_1 \cup \dots \cup X'_k$, 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 l’esempio del cammino minimo vincolato introdotto nel §4.2. Un ovvio rilassamento per questo problema è quello che elimina il vincolo sulla massima lunghezza dei cammini, producendo 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. 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 §5.3.3 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 affinché 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.

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 5.2, 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 perché 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 5.1 ha altezza n , anche se ha 2^n foglie. Si consideri, come esempio estremo di una regola che costruisce alberi “sbilanciati”, quella che costantemente seleziona 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 quasi altrettanto difficile.

L'equisuddivisione dei sottoproblemi è particolarmente importante qualora si usi una strategia depth-first, che visita un sottoalbero finché 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 semplicità 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 5.3. 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 negli esempi.

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.

5.2.4 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 $x_i = 1$;

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, 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.

5.3 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 pro e contro di alcune alternative diverse. Mostriamo poi il funzionamento dell'algoritmo su un esempio.

5.3.1 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 (cf. §A.4.1.1):
- branching sull'unica variabile frazionaria nella soluzione del rilassamento continuo (se ve ne sono);
- visita depth-first, visitando per primo il figlio in cui la variabile è fissata ad 1;
- 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.

La scelta del rilassamento continuo e dell'euristica CUD è dettata dal fatto che sono entrambe efficienti e “collaborano”; una volta ordinati gli oggetti dello zaino per costo unitario non crescente all'inizio dell'algoritmo, entrambe possono essere risolte in $O(n)$, e riutilizzando per l'euristica parte del lavoro svolto nel rilassamento. Infatti, il rilassamento continuo del problema dello zaino, ossia il problema in cui le variabili x_i possono assumere qualsiasi valore in $[0, 1]$, si può risolvere facilmente mediante il seguente algoritmo (greedy): si ordinano gli oggetti per costo unitario decrescente (CUD), cioè iniziando da quelli che hanno il più alto valore di c_i / a_i , si inizializza l'insieme I degli oggetti selezionati (ossia degli indici delle variabili i con $\bar{x}_i = 1$) all'insieme vuoto e si iniziano ad inserire oggetti (indici di variabili) in I finché c'è spazio. Quando si raggiunge il primo oggetto h (nell'ordine dato) per cui la capacità residua dello zaino non è più sufficiente, cioè $b - \sum_{i \in I} a_i < a_h$, si pone

$$\bar{x}_h = (b - \sum_{i \in I} a_i) / a_h$$

e $\bar{x}_i = 0$ per $i \notin I \cup \{h\}$; invece, l'euristica scarta l'oggetto h -esimo e continua nello stesso ordine ad inserirne interi, se entrano, finché lo zaino non è completamente pieno oppure si raggiunge l'ultimo oggetto. È facile verificare che \bar{x} è ammissibile per il rilassamento continuo; si noti che la soluzione è “quasi intera”, ossia c'è (potenzialmente) il solo elemento \bar{x}_h frazionario, tranne nel caso cui l'oggetto “critico” h ottiene $\bar{x}_h = 0$, il che significa che l'oggetto inserito nello zaino all'iterazione immediatamente precedente ha “completamente riempito lo zaino”. In tale caso \bar{x} è intera e quindi ottima per il problema dello zaino.

Esercizio 5.4. Si dimostri che la procedura suggerita determina la soluzione ottima del rilassamento continuo del problema dello zaino (suggerimento: si scriva il problema duale e si determini una soluzione duale ammissibile che rispetta le condizioni degli scarti complementari con quella ottenuta).

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 5.5. 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 5.3. B&B per lo zaino

Risolviamo la seguente istanza del problema dello zaino

$$\begin{array}{rcccccl} \max & 11x_1 & + & 8x_2 & + & 7x_3 & + & 6x_4 & & \\ & 5x_1 & + & 4x_2 & + & 4x_3 & + & 4x_4 & \leq & 12 \\ & x_1 & , & x_2 & , & x_3 & , & x_4 & \in & \{0, 1\} \end{array}$$

mediante l’algoritmo B&B proposto. Poiché il problema è “difficile”, nella pratica è uso inserire limiti allo sforzo computazionale totale (tempo, iterazioni, ...), accontentandosi pertanto di una soluzione approssimata in cambio della garanzia di ottenerla in tempi prestabiliti. In questo caso il limite viene implementato non visitando alcun nodo di livello superiore al secondo nell’albero delle decisioni, ossia effettuando branching solamente su al più due variabili. 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. Al nodo radice dell’albero delle decisioni abbiamo quindi

$$\bar{x} = [1, 1, 0.75, 0] \quad , \quad \bar{z} = 24.25 \quad , \quad \underline{x} = [1, 1, 0, 0] \quad , \quad \underline{z} = z = 19 \quad .$$

Il gap assoluto al nodo radice è quindi di 5.25. 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

$$\begin{array}{rcccccl} 7 + \max & 11x_1 & + & 8x_2 & + & 6x_4 & & \\ & 5x_1 & + & 4x_2 & + & 4x_4 & \leq & 8 \\ & x_1 & , & x_2 & , & x_4 & \in & \{0, 1\} \end{array} \quad .$$

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); pertanto 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. La valutazione superiore non permette però di eliminare il nodo, e d'altra parte è stato raggiunto il massimo livello consentito, pertanto anche se viene effettuato il branching sulla variabile x_4 l'esplorazione di questo sottoalbero viene comunque interrotta.

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

$$\bar{x} = [1, 1, 0, 0.75] \quad , \quad \bar{z} = 23.5 \quad , \quad \underline{x} = [1, 1, 0, 0] \quad , \quad \underline{z} = 19 < z = 21 \quad .$$

Questo non permette di terminare l'esplorazione, per cui si deve eseguire il branching sulla variabile frazionaria x_4 . Nel nodo in cui $x_3 = 0$ e $x_4 = 1$ si ottiene

$$\bar{x} = [1, 0.75, 0, 1] \quad , \quad \bar{z} = 23 \quad , \quad \underline{x} = [1, 0, 0, 1] \quad , \quad \underline{z} = 17 < z = 21 \quad .$$

Ancora una volta la valutazione superiore è scesa, ma non abbastanza per potare il nodo. Si dovrebbe quindi procedere al branching sulla variabile x_2 , ma poiché si è raggiunta la massima profondità consentita si esegue invece un "backtrack" per esaminare il nodo $x_3 = 0 = x_4 = 0$. In questo nodo il rilassamento continuo e l'euristica forniscono la soluzione intera $\bar{x} = \underline{x} = [1, 1, 0, 0]$ con $\bar{z} = \underline{z} = 19 < 21$, e pertanto il nodo può essere potato dalla valutazione superiore.

L'algoritmo viene a questo punto fatto terminare prematuramente avendo esaurito le risorse computazionali disponibili. A fronte della valutazione inferiore $\underline{z} = 21$, il Corollario 5.1 permette di determinare una valutazione superiore corrispondente al massimo tra quelle dei nodi che hanno generato figli che non sono stati ancora esplorati; questi sono $x_2 = 0, x_3 = 1$, con $\bar{z} = 22.5$, e $x_3 = 0, x_4 = 1$, con $\bar{z} = 23$. La miglior valutazione superiore disponibile in questo momento è quindi $z(P) \leq 23$, a cui corrisponde un gap assoluto di $\bar{z} - \underline{z} = 23 - 21 = 2$, e quindi un gap relativo di $(\bar{z} - \underline{z}) / \underline{z} = 2/21$ (circa il 9.5%). Proseguendo l'algoritmo si determinerebbe che la soluzione "incumbent" con $\underline{z} = 21$ è in effetti ottima, ma l'algoritmo, pur avendola determinata, non è al momento in grado di dimostrarne l'ottimalità. Questo accade abbastanza spesso in pratica: l'algoritmo determina la soluzione ottima abbastanza rapidamente, ma deve poi spendere una grande quantità di risorse per certificarne l'ottimalità.

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

$$x_i \leq \lfloor x_i^* \rfloor \quad \text{oppure} \quad x_i \geq \lceil x_i^* \rceil . \quad (5.1)$$

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. Sfruttando le opportunamente le capacità di riottimizzazione degli algoritmi del semplice (si veda il §5.2.1) 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; sono state proposte euristiche più sofisticate che 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 §5.4. 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.

5.3.2 Il problema del commesso viaggiatore

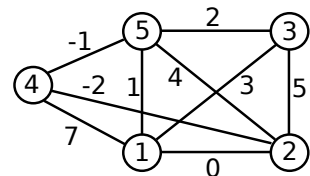
Discutiamo adesso un algoritmo di enumerazione implicita per il problema del commesso viaggiatore (TSP). Per quanto riguarda l'euristica potremmo ad esempio utilizzare il semplice algoritmo "Nearest Neighbour" (cf. A.4.1.3), ma scegliamo invece di non applicarne nessuna attendendo che sia il rilassamento, o il branching, a produrre soluzioni ammissibili. L'osservazione utile a determinare un rilassamento per il problema che possa essere calcolato in modo efficace è che qualsiasi ciclo Hamiltoniano è un sottografo connesso del grafo di partenza; di conseguenza, il suo costo non può essere inferiore a quello dell'albero di copertura di costo minimo. Quindi, (MST) fornisce un rilassamento per (TSP). Si può però migliorare il risultato notando che i cicli Hamiltoniani hanno esattamente n lati, mentre gli alberi di copertura ne hanno $n - 1$. Si vorrebbe quindi determinare il sottografo connesso di costo minimo del grafo dato con esattamente n lati. Ciò è in effetti facile: basta determinare l'(MST), e poi aggiungere il lato di costo minimo tra quelli non utilizzati. Per ovvi motivi questo viene detto *1-albero di copertura di costo minimo* (MS1-T), e può essere facilmente calcolato, ad esempio, con una minima modifica all'algoritmo di Kruskal (cf. §2.3.1). 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 $L = \{l_1, \dots, l_k\}$ ($k > 2$) 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 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 è 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. 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.

Un'utile regola di preprocessamento consiste nell'osservare che qualsiasi ciclo Hamiltoniano deve avere due diversi lati incidenti in ciascun nodo; quindi, se nel grafo esiste almeno un nodo con meno di due lati incidenti non è possibile costruire nessun ciclo Hamiltoniano (l'istanza è vuota). Mentre ci si aspetta che ciò non capiti per l'istanza originale, è ragionevole controllare la condizione durante l'algoritmo visto che la regola di branching elimina lati, e quindi potrebbe portare al fatto che la condizione non sia più verificata.

Esempio 5.4. B&B per il (TSP)

Risolviamo l'istanza del problema (TSP) illustrata in figura qui accanto con l'algoritmo descritto, avendo il limite massimo di esplorazione pari a sei nodi (compresa la radice). Per visitare depth-first l'albero delle decisioni si implementa Q come una pila. Decidiamo inoltre di inserire in Q i figli di ogni nodo in ordine decrescente del vertice j del lato $\{i, j\}$ fissato a zero, dove i è il vertice con grado 3; si noti che, essendo Q una pila, i nodi ne escono in ordine inverso a quello con cui vi entrano. Indichiamo con \underline{z} la valutazione inferiore ottenuta ad ogni nodo e con z la migliore delle valutazioni superiori determinate. La coda Q viene inizializzata inserendovi il solo nodo radice dell'albero delle decisioni, corrispondente a non aver fissato alcuna variabile; inoltre, si pone $z = +\infty$.



Nodo radice Il corrispondente MS1T, con $\underline{z} = 0$, è mostrato in Figura 5.3(a). Poiché non è un ciclo Hamiltoniano, non si è determinata alcuna soluzione ammissibile: pertanto $\underline{z} = 0 < \infty = z$ ed occorre procedere col branching. Ciò corrisponde a selezionare il vertice 5 (che ha tre lati incidenti) e creare tre figli, in ciascuno dei quali si fissa a zero la variabile corrispondente a uno di tali lati; per la strategia di visita stabilita, si inseriscono in Q nell'ordine $\{4, 5\}$, $\{3, 5\}$ e $\{1, 5\}$ (essendo Q una pila, verranno estratti nell'ordine inverso).

$x_{15} = 0$ Il corrispondente MS1T, con $\underline{z} = 2$, è mostrato in Figura 5.3(b). Poiché è un ciclo Hamiltoniano, si pone $z = 2$. Inoltre il nodo viene chiuso per ottimalità.

$x_{35} = 0$ Il corrispondente MS1T, con $\underline{z} = 1$, è mostrato in Figura 5.3(c). Poiché $\underline{z} = 1 < 2 = z$, occorre procedere col branching. Ciò corrisponde a selezionare il vertice 1 (che ha tre lati incidenti), e creare tre figli fissando a zero, rispettivamente, la variabile relativa ai lati $\{1, 5\}$, $\{1, 3\}$, e $\{1, 2\}$.

$x_{35} = 0, x_{12} = 0$ Il corrispondente MS1T, con $\underline{z} = 5$, è mostrato in Figura 5.3(d). Poiché $\underline{z} = 5 > 2 = z$, il nodo viene chiuso dalla valutazione inferiore.

$x_{35} = 0, x_{13} = 0$ Poiché il nodo 3 ha un solo lato incidente rimasto non può esistere nessun ciclo Hamiltoniano nel grafo ed il nodo viene chiuso per inammissibilità.

$x_{35} = 0, x_{15} = 0$ Il corrispondente MS1T, con $\underline{z} = 4$, è mostrato in Figura 5.3(e). Poiché $\underline{z} = 4 > 2 = z$, il nodo viene chiuso dalla valutazione inferiore.

Poiché Q non è vuota, l'algoritmo viene interrotto anticipatamente. L'analisi dell'algoritmo B&B assicura che

$$\min \{ z, \min \{ \underline{z}(P') : P' \in Q' \} \}$$

è una valutazione inferiore globale corretta, dove Q' è l'insieme dei predecessori immediati dei nodi in Q . In questo caso, Q contiene il solo nodo $[x_{45} = 0]$, e pertanto Q' contiene il nodo radice che ha $\underline{z} = 0$; quindi, la valutazione inferiore globale è 0. In altri termini, le iterazioni svolte non sono riuscite a migliorare la valutazione inferiore globale già determinata al nodo radice (un ben noto problema della strategia di visita depth-first); per contro, è facile verificare che la soluzione ottenuta è effettivamente ottima, ma l'algoritmo non è stato in grado di dimostrarlo.

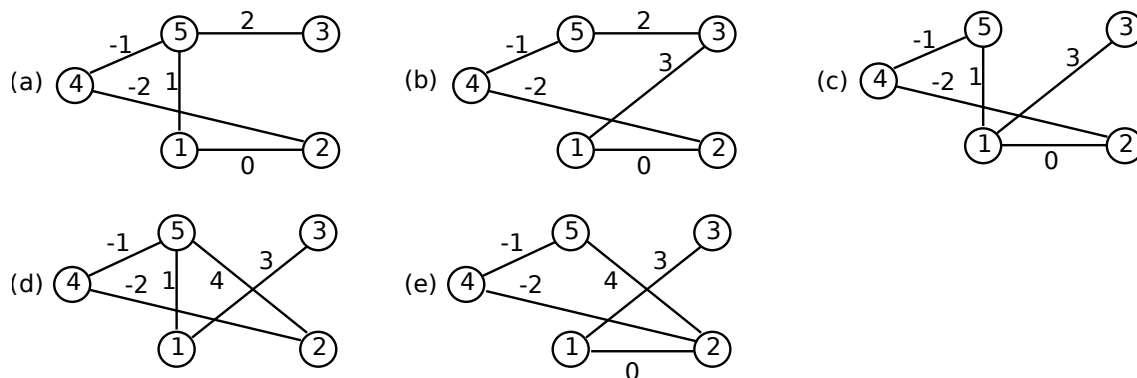


Figura 5.3: Algoritmo B&B per il (TSP)

5.3.3 Il problema del cammino minimo vincolato

Per questo problema, introdotto nell'Esempio 4.2, analizzeremo un algoritmo enumerativo che usa il rilassamento "naturale" del problema ottenuto eliminando il vincolo "complicante" sulla massima lunghezza dei cammini. 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 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 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 5.5. B&B per (CSP)

Applichiamo l’algoritmo $B\&B$ all’istanza di raffigurata qui accanto. Al nodo radice si ha la situazione rappresentata in Figura 5.4(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.

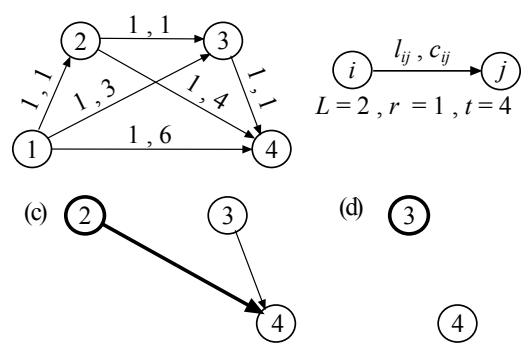


Figura 5.4: Algoritmo $B\&B$ per il (CSP)

$x_{12} = 0$: il corrispondente cammino minimo è mostrato in Figura 5.4(b); il cammino ha costo 4 ed è ammissibile, per cui si pone $z = 4$ ed il nodo viene potato per ottimalità.

$x_{12} = 1, x_{23} = 0$: si sposta la radice r al nodo 2 ponendo $L = 1$, ottenendo il cammino ottimo illustrato in Figura 5.4(c) che ha costo 5 $> z = 4$: il nodo viene quindi potato dalla valutazione inferiore.

$x_{12} = x_{23} = 1, x_{34} = 0$: si sposta la radice r al nodo 3 e si pone $L = 0$, come illustrato in Figura 5.4(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 5.4(b).

Esercizio 5.6. 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.

5.4 Tecniche poliedrali

Nei solutori *general-purpose* per la *PLI* basati sulla *PL* (si veda, ad esempio, la discussione in fondo al §5.3.1) 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), che dipende dalla differenza tra il poliedro descritto dalla formulazione *PLI* utilizzata e l'involuppo convesso delle soluzioni intere, come discusso nel §4.2.2. Visto che, in generale, l'espressione dell'involuppo 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'involuppo 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 potrebbe 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.

Nel seguito di questo paragrafo discuteremo prima una classe molto generale di disuguaglianze valide, i *tagli di Chvátal*, che permettono in principio di derivare tutti i vincoli che caratterizzano l'involuppo 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*.

5.4.1 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

$$a_y x \leq b_y \quad \text{con} \quad a_y = yA \text{ e } b_y = yb \quad (5.2)$$

è valida per tutti i punti di \mathcal{P} : infatti,

$$Ax \leq b \text{ e } y \geq 0 \implies (yA)x \leq (yb) .$$

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.

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

$$a_y x \leq \lfloor b_y \rfloor \quad (5.3)$$

è una disuguaglianza valida per tutti i punti di $\text{conv}(\mathcal{P} \cap \mathbb{Z}^n)$, 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* (5.3) è “più forte” della disuguaglianza originaria (5.2) 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 (5.2) e (5.3) 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 $\mathcal{P}^{(t)} = \text{conv}(\mathcal{P} \cap \mathbb{Z}^n)$.

In altri termini, qualsiasi disuguaglianza necessaria a rappresentare l'involuppo convesso delle soluzioni intere del problema può essere ottenuta ripetendo un numero finito di volte l'operazione (5.3). Si noti però che non è sufficiente utilizzare le disuguaglianze originarie di \mathcal{P} , ossia $\mathcal{P}^{(1)}$ può contenere strettamente l'involuppo 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 $\mathcal{P}^{(t)} = \text{conv}(\mathcal{P} \cap \mathbb{Z}^n)$, detto il *rango di Chvátal* del problema PLI , può essere considerato una sorta di misura di “quanto sia difficile” rappresentare l'involuppo 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 (5.3).

Per quanto interessanti, i risultati appena descritti sono principalmente teorici: non indicano cioè 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 .

5.4.2 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 $i \in \{1, \dots, n\}$ 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, $h b_B = u_i A_B^{-1} b_B = u_i \bar{x} = \bar{x}_i$. Definiamo quindi

$$y_B = h - [h] \quad . \quad (5.4)$$

Poiché $[h] \leq h$ per definizione, $y_B \geq 0$. Pertanto, ponendo $r = [h] \in \mathbb{Z}^n$ si ha che

$$\begin{aligned} a_y &= y_B A_B = (h - r) A_B = u_i A_B^{-1} A_B - r A_B = u_i - r A_B \\ b_y &= y_B b_B = (h - r) b_B = u_i A_B^{-1} b_B - r b_B = \bar{x}_i - r b_B \end{aligned} \quad (5.5)$$

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 (5.3) produca un taglio “più forte” di (5.5), 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 §5.2.3. 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 $r b_B \in \mathbb{Z}$, ma $\bar{x}_i \notin \mathbb{Z}$. Di conseguenza, il *taglio di Gomory* (un particolare taglio di Chvátal)

$$g = a_y = u_i - rA_B, \quad \delta = \lfloor b_y \rfloor = \lfloor \bar{x}_i - rb_B \rfloor \quad (5.6)$$

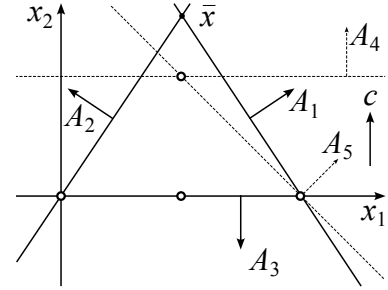
è violato dalla soluzione ottima \bar{x} del rilassamento continuo, ossia $g\bar{x} > \delta$. Poiché $g \in \mathbb{Z}^n$ e $\delta \in \mathbb{Z}$, aggiungendo il nuovo vincolo alla formulazione del problema si mantiene la proprietà che la matrice dei coefficienti ed il vettore dei lati destri sono 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 5.6. Applicazione dei tagli di Gomory

Si consideri il semplice problema di *PLI*

$$\begin{array}{llll} \max & & x_2 & \\ & 3x_1 + 2x_2 & \leq & 6 \\ & -3x_1 + 2x_2 & \leq & 0 \\ & & -x_2 & \leq 0 \\ & x_1, & x_2 & \in \mathbb{Z} \end{array}$$

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



$$A_B = \begin{bmatrix} 3 & 2 \\ -3 & 2 \end{bmatrix}, \quad A_B^{-1} = \begin{bmatrix} 1/6 & -1/6 \\ 1/4 & 1/4 \end{bmatrix}, \quad \bar{x} = A_B^{-1}b_B = \begin{bmatrix} 1 \\ 3/2 \end{bmatrix}, \quad \bar{y}_B = cA_B^{-1} = [1/4, 1/4]$$

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

$$g = u_i - rA_B = [0, 1], \quad \delta = \lfloor \bar{x}_i - rb_B \rfloor = \lfloor 3/2 - 0 \rfloor = 1.$$

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,

$$A_B = \begin{bmatrix} 3 & 2 \\ 0 & 1 \end{bmatrix}, \quad A_B^{-1} = \begin{bmatrix} 1/3 & -2/3 \\ 0 & 1 \end{bmatrix}, \quad \bar{x} = A_B^{-1}b_B = \begin{bmatrix} 4/3 \\ 1 \end{bmatrix}, \quad \bar{y}_B = cA_B^{-1} = [0, 1]$$

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

$$h = u_i A_B^{-1} = [1/3, -2/3], \quad r = \lfloor h \rfloor = [0, -1], \quad g = u_i - rA_B = [1, 1], \quad \delta = \lfloor \bar{x}_i - rb_B \rfloor = \lfloor 4/3 - (-1) \rfloor = 2.$$

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 $\bar{x} = [1, 1] = x^*$, 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 5.7. 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'involuppo 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. 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. L'efficienza dell'approccio dipende dall'opportuna combinazione di molte scelte algoritmiche che non possono essere discusse in dettaglio in queste dispense. 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 5.7. 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. §5.2.1) 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 (5.1) 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 $x_i \geq \lceil x_i^* \rceil$.

$$\begin{array}{llll} \max & x_1 & & \\ & -8x_1 & & \leq -7 \\ & x_1 & - & 4x_2 \leq 0 \\ & x_1 & + & 4x_2 \leq 4 \\ & x_1 & , & x_2 \in \mathbb{Z} \end{array}$$

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 (5.1) 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 $x_i \geq \lceil x_i^* \rceil$.

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

$$A_B = \begin{bmatrix} -8 & 0 \\ 1 & -4 \end{bmatrix}, \quad A_B^{-1} = \begin{bmatrix} -1/8 & 0 \\ -1/32 & -1/4 \end{bmatrix}, \quad \bar{x} = A_B^{-1}b_B = \begin{bmatrix} -1/8 & 0 \\ -1/32 & -1/4 \end{bmatrix} \begin{bmatrix} -7 \\ 0 \end{bmatrix} = \begin{bmatrix} 7/8 \\ 7/32 \end{bmatrix}$$

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 :

$$\bar{y}_B = cA_B^{-1} = [1, 0] \begin{bmatrix} -1/8 & 0 \\ -1/32 & -1/4 \end{bmatrix} = [-1/8, 0], \quad h = \min\{i : \bar{y}_i < 0\} = 1, \quad \xi = -A_B^{-1}u_{B(h)} = \begin{bmatrix} 1/8 \\ 1/32 \end{bmatrix}$$

$$A_3\xi = 1/4 > 0, \quad \lambda_3 = (b_3 - A_3\bar{x}) / A_3\xi = (4 - 7/4) / (1/4) = 9, \quad \bar{\lambda} = \min\{\lambda_i : i \in N, A_i\xi > 0\} = \lambda_3 = 9$$

$$k = \min\{i \in N : \lambda_i = \bar{\lambda}\} = \min\{3\} = 3, \quad B = B \setminus \{h\} \cup \{k\} = \{1, 2\} \setminus \{1\} \cup \{3\} = \{2, 3\}$$

Alla seconda iterazione si ha quindi $B = \{2, 3\}$, per la quale

$$A_B = \begin{bmatrix} 1 & -4 \\ 1 & 4 \end{bmatrix}, \quad A_B^{-1} = \begin{bmatrix} 1/2 & 1/2 \\ -1/8 & 1/8 \end{bmatrix}, \quad \bar{x} = A_B^{-1}b_B = \begin{bmatrix} 2 \\ 1/2 \end{bmatrix}, \quad \bar{y}_B = cA_B^{-1} = [1/2, 1/2] \geq 0.$$

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 ($\lceil 1/2 \rceil = 1$): 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

$$h = u_2 A_B^{-1} = [0, 1] \begin{bmatrix} 1/2 & 1/2 \\ -1/8 & 1/8 \end{bmatrix} = [-1, 1], \quad r = \lfloor h \rfloor = [-1/8, 1/8]$$

$$g = u_2 - rA_B = [0, 1] - [-1, 0] \begin{bmatrix} 1 & -4 \\ 1 & 4 \end{bmatrix} = [1, -3], \quad \delta = \lfloor \bar{x}_i - rb_B \rfloor = \left\lfloor 1/2 - [-1, 0] \begin{bmatrix} 0 \\ 4 \end{bmatrix} \right\rfloor = 0.$$

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

$$\eta_B = A_k A_B^{-1} = [1, -3] \begin{bmatrix} 1/2 & 1/2 \\ -1/8 & 1/8 \end{bmatrix} = [7/8, 1/8]$$

$$\theta = \min\{\bar{y}_i / \eta_i : i \in B, \eta_i > 0\} = \min\{(1/2) / (7/8), (1/2) / (1/8)\} = \min\{4/7, 4\} = 4/7$$

$$h = \min\{i \in B; \bar{y}_i / \eta_i = \theta\} = \min\{2\} = 2, \quad B = B \setminus \{h\} \cup \{k\} = \{2, 3\} \setminus \{2\} \cup \{4\} = \{3, 4\}.$$

Alla seconda iterazione si ha quindi $B = \{3, 4\}$, per la quale

$$A_B = \begin{bmatrix} 1 & 4 \\ 1 & -3 \end{bmatrix}, \quad A_B^{-1} = \begin{bmatrix} 3/7 & 4/7 \\ 1/7 & -1/7 \end{bmatrix}, \quad \bar{x} = A_B^{-1}b_B = \begin{bmatrix} 12/7 \\ 4/7 \end{bmatrix}, \quad \bar{y}_B = cA_B^{-1} = [3/7, 4/7]$$

$$A_N \bar{x} = \begin{bmatrix} -8 & 0 \\ 1 & -4 \end{bmatrix} \begin{bmatrix} 12/7 \\ 4/7 \end{bmatrix} = \begin{bmatrix} -96/7 \\ -4/7 \end{bmatrix} \leq \begin{bmatrix} -7 \\ 0 \end{bmatrix} = b_N,$$

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

$$\eta_B = A_k A_B^{-1} = [1, 0] \begin{bmatrix} 3/7 & 4/7 \\ 1/7 & -1/7 \end{bmatrix} = [3/7, 4/7]$$

$$\theta = \min\{(3/7) / (3/7), (4/7) / (4/7)\} = \min\{1, 1\} = 1, \quad h = \min\{3, 4\} = 3, \quad B = \{4, 5\}.$$

Alla seconda iterazione si ha quindi

$$A_B = \begin{bmatrix} 1 & -3 \\ 1 & 0 \end{bmatrix}, \quad A_B^{-1} = \begin{bmatrix} 0 & 1 \\ -1/3 & 1/3 \end{bmatrix}, \quad \bar{x} = A_B^{-1}b_B = \begin{bmatrix} 1 \\ 1/3 \end{bmatrix}, \quad \bar{y}_B = cA_B^{-1} = [0, 1] \geq 0$$

$$A_N \bar{x} = \begin{bmatrix} -8 & 0 \\ 1 & -4 \\ 1 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 1/3 \end{bmatrix} = \begin{bmatrix} -8 \\ -1/3 \\ 7/3 \end{bmatrix} \leq \begin{bmatrix} -7 \\ 0 \\ 4 \end{bmatrix} = b_N,$$

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 valutazioni 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 $x = \lceil \bar{x} \rceil = [1, 0]$ 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

$$h = u_2 A_B^{-1} = [0, 1] \begin{bmatrix} 0 & 1 \\ -1/3 & 1/3 \end{bmatrix} = [-1/3, 1/3], \quad r = [h] = [-1, 0]$$

$$g = u_2 - rA_B = [0, 1] - [-1, 0] \begin{bmatrix} 1 & -3 \\ 1 & 0 \end{bmatrix} = [1, -2], \quad \delta = [\bar{x}_i - rb_B] = [1/3 - [-1, 0] \begin{bmatrix} 0 \\ 1 \end{bmatrix}] = 0.$$

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 5.8. Si rappresenti graficamente il problema in esame e si verifichi geometricamente l'esattezza delle relazioni ottenute algebricamente.

Esercizio 5.9. 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.

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

L. Wolsey “Integer Programming” *Wiley*, 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, ed agli aspetti legati alla modellazione di situazioni (ir)reali, di interesse per studenti di ambiti più applicativi quali le ingegneria.

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'applicazione 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 e 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. Presentiamo inoltre una breve introduzione ad alcuni dei principali paradigmi algoritmici che si utilizzano per la soluzione, eventualmente approssimata, di 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'istruzione 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 “sì”, 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 *Polinomiale nel calcolo Nondeterministico*). 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. 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 §5.1) 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 "sì" 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).

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 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.

A.4 Algoritmi

Al termine di questa appendice, presentiamo una brevissima panoramica di alcune classi generali di algoritmi per la soluzione, eventualmente approssimata, di problemi di ottimizzazione. Questa panoramica non ha certamente la pretesa di essere esaustiva, ma solo di offrire un “primo assaggio” di alcune fondamentali tecniche algoritmiche che sono usate in queste note.

A.4.1 Algoritmi greedy

Gli algoritmi *greedy* (voraci) determinano la 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, non garantiscono l'ottimalità, ed a volte neppure l'ammissibilità, della soluzione trovata. Vediamo adesso tre esempi di algoritmi greedy.

A.4.1.1 Algoritmo “Costi Unitari Decrescenti”

Questo algoritmo per il problema dello zaino costruisce una soluzione inserendo per primi nello zaino gli oggetti “più promettenti”, cioè quelli che hanno maggior valore per unità di peso. L'algoritmo inizializza l'insieme I degli oggetti selezionati come l'insieme vuoto, e poi scorre la lista degli oggetti in ordine di costo unitario non crescente, cioè iniziando da quelli che hanno il più alto valore di c_i / a_i . L'oggetto a_h di volta in volta selezionato viene accettato se la capacità residua dello zaino è sufficiente, cioè se $b - \sum_{i \in I} a_i \geq a_h$; in questo caso l'oggetto a_h viene aggiunto ad I , 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.

Consideriamo ad esempio la seguente istanza del problema dello zaino:

$$\begin{array}{rcccccc} \text{Max} & 7x_1 & +2x_2 & +4x_3 & +5x_4 & +4x_5 & +x_6 \\ & 5x_1 & +3x_2 & +2x_3 & +3x_4 & +x_5 & +x_6 \leq 8 \\ & x_1, & x_2, & x_3, & x_4, & x_5, & x_6 \in \{0, 1\} \end{array}$$

In questo caso, l'algoritmo greedy farebbe 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. Pertanto, l'algoritmo descritto non garantisce l'ottimalità; tuttavia ha il vantaggio di essere di semplice implementazione e computazionalmente molto efficiente, essendo la sua complessità lineare se gli oggetti sono forniti in input già ordinati per costo unitario decrescente.

A.4.1.2 L'Algoritmo di Kruskal

Questo algoritmo per il problema dell'albero di copertura di costo minimo di un grafo connesso costruisce una soluzione inserendo i lati archi uno per volta in ordine di costo non decrescente (i meno costosi prima), e verificando ogni volta che il sottografo corrispondente sia privo di cicli. L'algoritmo inizializza l'insieme T dei lati appartenenti all'albero come l'insieme vuoto, e poi scorre la lista dei lati archi in ordine di costo non decrescente. Il lato $\{i, j\}$ di volta in volta selezionato viene accettato se il sottografo avente tutti i lati in $T \cup \{\{i, j\}\}$ è privo di cicli; in questo caso $\{i, j\}$ viene aggiunto a T , altrimenti viene scartato e si passa al successivo nell'ordinamento. L'algoritmo termina quando T contiene esattamente $n - 1$ lati. A differenza del caso precedente, si può dimostrare che la soluzione generata dall'algoritmo è ottima: l'algoritmo è esatto, ed può essere implementato in modo da avere complessità polinomiale $O(m \log n)$.

A.4.1.3 Algoritmo “Nearest Neighbour”

Questo algoritmo per il problema del commesso viaggiatore costruisce una soluzione scegliendo ad ogni passo come prossima tappa la località più vicina a quella in cui il commesso si trova attualmente. L'algoritmo inizializza l'insieme P dei lati appartenenti al cammino come l'insieme vuoto, e definisce

come vertice “corrente” il vertice iniziale (1). Ad ogni iterazione, poi, esamina il vertice corrente i e tutti i lati che lo uniscono a vertici che non sono ancora toccati dal cammino parziale P : tra di essi seleziona il lato $\{i, j\}$ di lunghezza minima, lo aggiunge a P e definisce j come nuovo vertice corrente. L'algoritmo termina quando tutti i vertici sono toccati da P , inserendo il lato di ritorno dall'ultimo vertice al vertice 1.

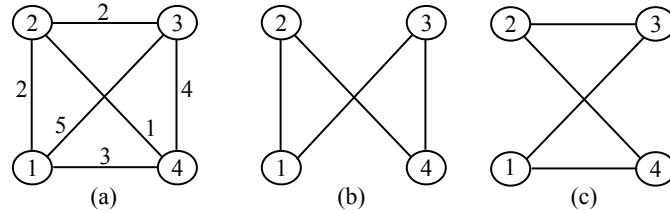


Figura A.1: Un'istanza del problema del commesso viaggiatore

Come nel caso del problema dello zaino, l'algoritmo greedy non è esatto (del resto, entrambi i problemi sono \mathcal{NP} -ardui): questo può essere facilmente verificato mediante l'istanza rappresentata in Figura A.1(a). L'algoritmo “Nearest Neighbour”, partendo dal nodo 1, produce il ciclo rappresentato in Figura A.1(b), con lunghezza 12, che è peggiore del ciclo ottimo rappresentato in Figura A.1(c), che ha costo 11. Del resto, ancora una volta il costo computazionale della procedura è molto basso, essendo lineare nel numero degli archi del grafo ($O(n^2)$).

A.4.1.4 Uno schema generale

L'analisi dei tre esempi precedenti suggerisce uno schema generale di algoritmo greedy, adatto a tutti quei casi in cui:

- le soluzioni ammissibili del problema possono essere rappresentate come una famiglia $F \subset 2^E$ di sottoinsiemi di un dato insieme “base” $E = \{e_1, e_2, \dots, e_n\}$;
- dati due sottoinsiemi ammissibili $A \in F$ e $B \in F$, se $A \subset B$ allora B è una soluzione preferibile ad A .

Lo schema generale è descritto nella seguente procedura:

```

procedure  $S = Greedy$   {
   $S = \emptyset$ ;  $R = \emptyset$ ;  $Q = E$ ;
  repeat
     $e = Best(Q)$ ;  $Q = Q \setminus \{e\}$ ;
    if ( $S \cup \{e\} \in F$ )
      then  $S = S \cup \{e\}$ ;
      else  $R = R \cup \{e\}$ ;
  until ( $Q = \emptyset$ )
}
```

Procedura 1.1: Algoritmo *Greedy*

Nella procedura, S è l'insieme degli elementi di E che sono stati inseriti nella soluzione (parziale) corrente, R è l'insieme degli elementi per i quali si è già deciso che non faranno parte della soluzione finale, e Q è l'insieme degli elementi ancora da esaminare. 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 complessità della procedura *Greedy* dipende fondamentalmente dal costo di controllare se $S \cup \{e\} \in F$, che denoteremo con $k(n)$, e dalla complessità della funzione $Best()$. Se supponiamo che gli elementi di E vengano ordinati all'inizio, e che quindi ad ogni passo $Best$ fornisca semplicemente il primo fra i rimanenti, la complessità della procedura *Greedy* è $O(n(\log n + k(n)))$.

Esercizio A.1. Sia dato un grafo $G = (N, A)$; si proponga, fornendone una descrizione formale, un algoritmo greedy per la colorazione dei suoi nodi con il minimo numero di colori, con il vincolo che due nodi adiacenti non abbiano mai lo stesso colore.

A.4.2 Algoritmi di ricerca locale

Gli algoritmi di ricerca locale determinano una soluzione ammissibile di partenza x^0 come “soluzione corrente”; ad ogni passo, poi, esaminano le soluzioni “vicine” a quella corrente, e se ne trovano una “migliore” (tipicamente, con miglior valore della funzione obiettivo) la selezionano come nuova soluzione corrente. Viene così generata una sequenza di soluzioni ammissibili $\{x^0, x^1, \dots, x^k, \dots\}$; l'algoritmo si ferma quando la soluzione corrente viene riconosciuta come un *ottimo locale*, ossia quando nessuna delle soluzioni “vicine” è migliore di quella corrente.

Elemento caratterizzante di un algoritmo di questo tipo è la definizione di “vicinanza” tra le soluzioni; se F è l'insieme ammissibile del problema in esame, viene definita una *funzione intorno* $I : F \rightarrow 2^F$ che possiede le due seguenti proprietà:

1. $x \in F \Rightarrow x \in I(x)$;
2. $x, y \in F \Rightarrow$ esiste un insieme finito $\{z^0, z^1, \dots, z^p\} \subseteq F$ tale che $z^0 = x, z^k \in I(z^{k-1})$ per $k = 1, 2, \dots, p, z^p = y$.

L'insieme $I(x)$ è detto *intorno* di x ; la proprietà 1. richiede che ogni soluzione appartenga all'intorno di se stessa, mentre la proprietà 2. richiede che sia possibile raggiungere in un numero finito di passi qualsiasi soluzione $y \in F$ a partire da qualsiasi altra soluzione $x \in F$ muovendosi sempre nell'intorno della soluzione corrente. Questo è necessario se si vuole che l'algoritmo abbia la possibilità di determinare la soluzione ottima per qualunque scelta del punto iniziale x^0 .

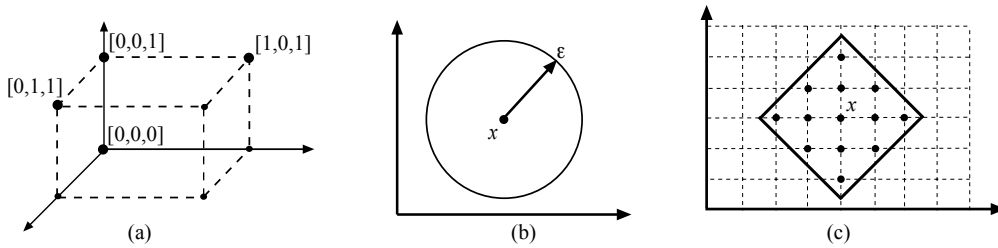


Figura A.2: Alcuni esempi di funzioni intorno

Alcuni esempi di funzioni intorno sono mostrati in Figura A.2, in particolare

- Figura A.2(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 A.2(b): $F \subseteq \mathbb{R}^n$, $I_\varepsilon(x) = \{y \in F : \|x - y\| \leq \varepsilon\}$ (Intorno Euclideo);
- Figura A.2(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 che trasformano una soluzione ammissibile del problema in un'altra soluzione ammissibile. Un esempio di intorno con queste caratteristiche è quello definito dalle operazioni di “2-scambio” per il problema del commesso viaggiatore.

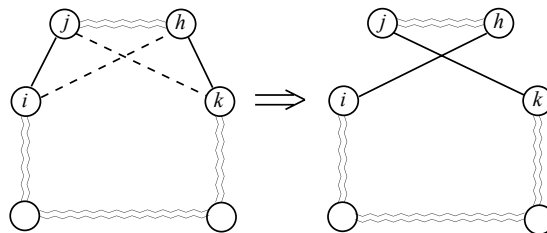


Figura A.3: Un “2-scambio” per il TSP

Nel TSP, una soluzione ammissibile x è un ciclo Hamiltoniano: 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 $\{j, k\}$. Questa operazione, detta di “2-scambio”, è esemplificata in Figura A.3. Analogamente si possono definire intorni basati sul “ k -scambio” che coinvolgono k archi del ciclo.

Opportunamente definita la funzione intorno, un algoritmo di ricerca locale può essere schematizzato come segue:

```

procedure  $x = RicercaLocale(F, c, x)$ 
  while ( $\sigma(x) \neq x$ ) do  $x = \sigma(x)$ ;
    
```

Procedura 1.2: Algoritmo di Ricerca Locale

L'algoritmo necessita in input di una soluzione ammissibile da cui partire: spesso, tale soluzione è costruita usando un algoritmo greedy. σ è una trasformazione che, data una soluzione x^k della sequenza generata, fornisce la soluzione successiva $x^{k+1} = \sigma(x^k)$. La tipica implementazione di σ , basata su una funzione intorno I , è

$$\sigma_I(x) = \operatorname{argmin}\{c(y) : y \in I(x)\} ,$$

ossia fornisce uno degli elementi di $I(x)$ aventi costo minimo. In questo modo, ad ogni passo dell'algoritmo viene risolto un problema di ottimo ristretto all'intorno considerato; alternativamente si può definire una trasformazione che per ogni x fornisca un qualsiasi elemento $y \in I(x)$, con $c(y) < c(x)$, se un tale elemento esiste, oppure x stesso se un tale elemento non esiste. Si ha $x = \sigma(x)$ quando nell'intorno non esiste nessuna soluzione migliore di x , ossia x è un *ottimo locale* per la funzione intorno I .

La complessità di un algoritmo di ricerca locale dipende fondamentalmente dalla complessità della trasformazione σ , che a sua volta dipende tipicamente dalla dimensione e dalla struttura della funzione intorno I . La dimensione dell'intorno $I(x)$ è anche collegata alla qualità degli ottimi locali che si determinano: intorni “più grandi” tipicamente forniscono ottimi locali di migliore qualità. Esiste quindi un “trade-off” tra complessità della trasformazione σ e qualità delle soluzioni determinate che deve essere accuratamente valutato.

Nel caso del TSP, ad esempio, la cardinalità di $I(x)$ per l'intorno basato sui “2-scambi” è $O(n^2)$: è quindi possibile enumerare in un tempo relativamente breve tutte le soluzioni dell'intorno e scegliere la migliore. Non è difficile vedere che la complessità di calcolare σ_I quando I è definita mediante operazioni di “ k -scambio” cresce grosso modo come $k!$: 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. Per questo si usano schemi di ricerca locale in cui la dimensione dell'intorno (il numero k di scambi) varia dinamicamente, e nei quali, quando k è “grande”, ci si accontenta di determinare una qualunque soluzione nell'intorno che migliori il valore della funzione obiettivo, senza necessariamente richiedere che sia la migliore.

Esercizio A.2. Si consideri il seguente problema di “clustering”: ripartire m vettori reali ad n componenti dati y_1, y_2, \dots, y_m in $k < m$ sottoinsiemi in modo che sia minima la somma delle distanze fra vettori appartenenti allo stesso sottoinsieme. Descrivere un algoritmo “greedy” per trovare una soluzione ammissibile, ed un algoritmo di ricerca locale per la determinazione di un ottimo locale.

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: $N = \{1, 2, \dots, n\}$. L'insieme A , detto *insieme degli archi*, è in generale indicato con $A = \{a_1, a_2, \dots, a_m\}$ ($m = |A|$). 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.

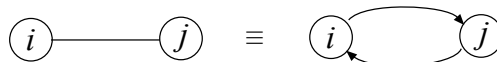


Figura B.1: Equivalenza tra un lato ed una coppia di archi

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

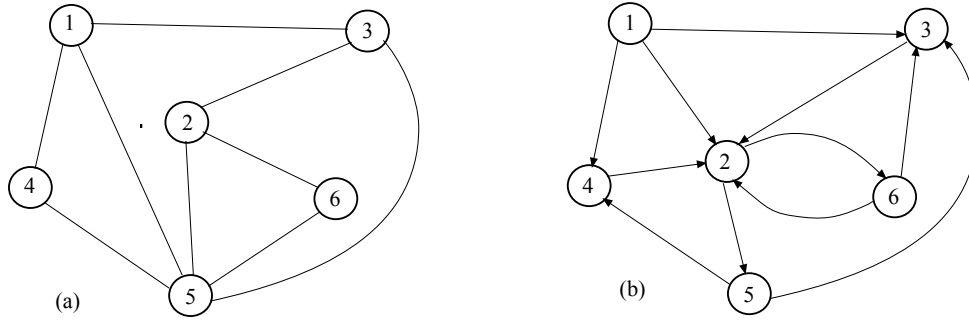


Figura B.2: Esempi di grafi

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

$$FN(i) = \{j \in N : \exists (i, j) \in A\} \quad , \quad BN(i) = \{j \in N : \exists (j, i) \in A\} \quad ,$$

mentre con $FS(i)$ e $BS(i)$ si indicano rispettivamente l'insieme degli archi uscenti da i , o stella uscente di i , e l'insieme degli archi entranti in i , o stella entrante di i :

$$FS(i) = \{(x, y) \in A : x = i\} \quad , \quad BS(i) = \{(x, y) \in A : y = i\} \quad .$$

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

$$C = \{i_0, a_1, i_1, a_2, i_2, \dots, i_{q-1}, a_q, i_q\}$$

in cui per ogni $h = 1, \dots, q$, a_h è incidente in i_{h-1} e i_h . Se, per $h = 1, \dots, q$, 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.

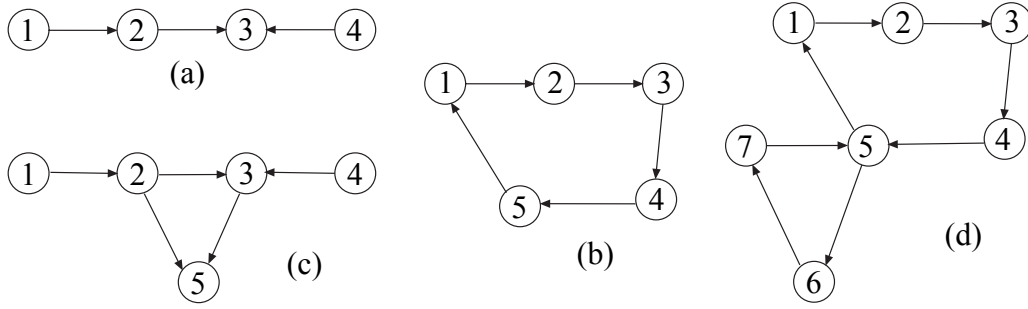


Figura B.3: (a) un cammino semplice; (b) un ciclo orientato semplice; (c) un cammino non semplice con ripetizione di archi; (d) un ciclo orientato non semplice senza ripetizione di archi

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 *ciclo 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 Figura 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 $N'' = \{4, 5, 6, 7\}$.

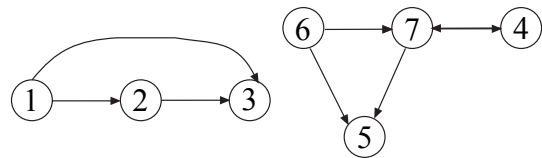


Figura B.4: Due componenti connesse

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.

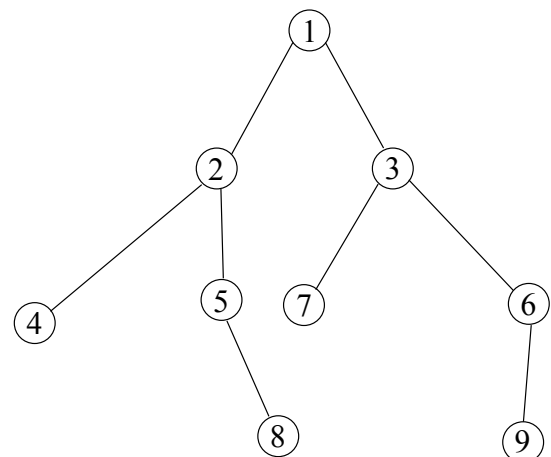


Figura B.5: Un albero

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.

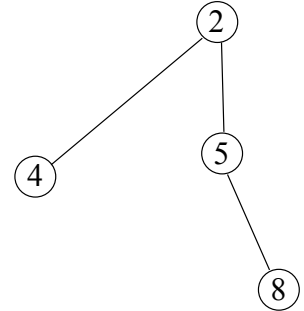


Figura B.6: Un sottoalbero

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

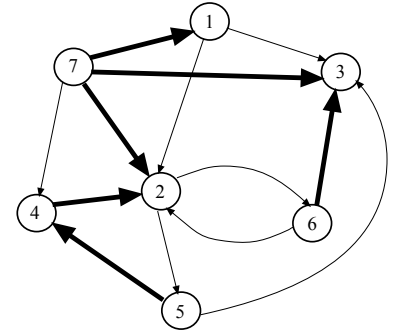


Figura B.7: Un albero di copertura

$$A(N_i, N_j) = \{ (k, l) \in A : k \in N_i, l \in N_j \text{ oppure } l \in N_i, k \in N_j \} ;$$

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 $\{FS(1), FS(2), \dots, FS(n)\}$ 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 $\{BS(1), BS(2), \dots, BS(n)\}$.

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

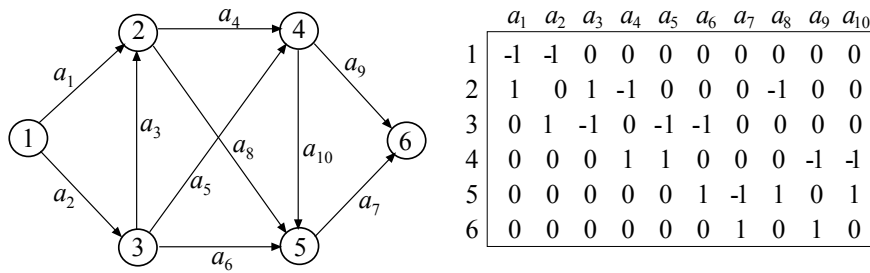


Figura B.8: Un grafo e la sua matrice di incidenza

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.

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

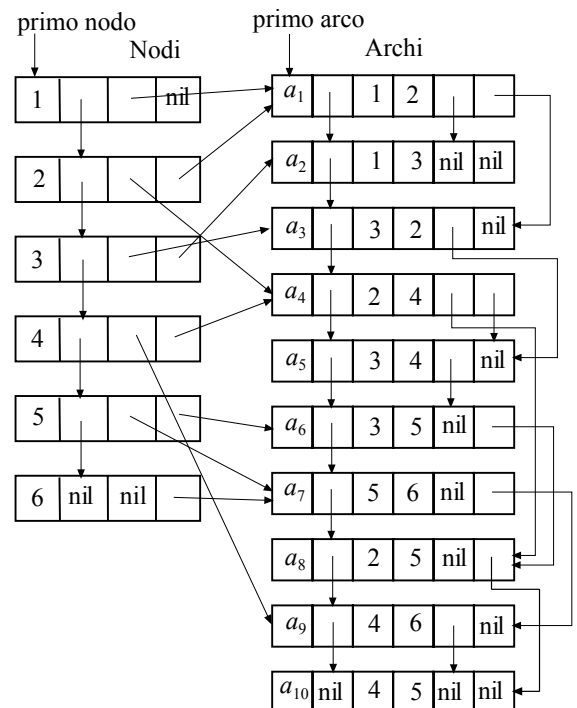


Figura B.9: Liste di adiacenza

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, $i = 1, \dots, n$, 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, $k = 1, \dots, m$, 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

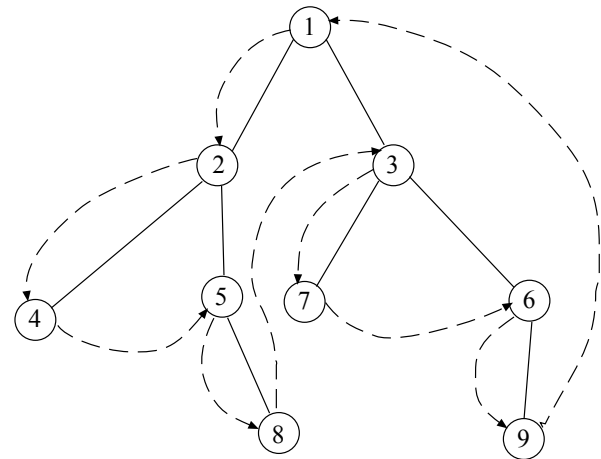


Figura B.10: La funzione $va(\cdot)$ della visita anticipata

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 (  $u \neq r$  ) 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 . Mostriamo 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  $p = \text{Visita}(G, r)$  {
  foreach(  $i \in N$  ) do  $p[i] = 0$ ;
   $Q = \{r\}$ ;  $p[r] = r$ 
  do {  $i = \text{Select}(Q)$ ;  $Q = Q \setminus \{i\}$ ;
    foreach(  $(i, j) \in FS[i]$  ) do
      if(  $p[j] == 0$  ) then {  $p[j] = i$ ;  $Q = Q \cup \{j\}$ ; }
    } while(  $Q \neq \emptyset$  );
}

```

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 $1, \dots, n$ (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:

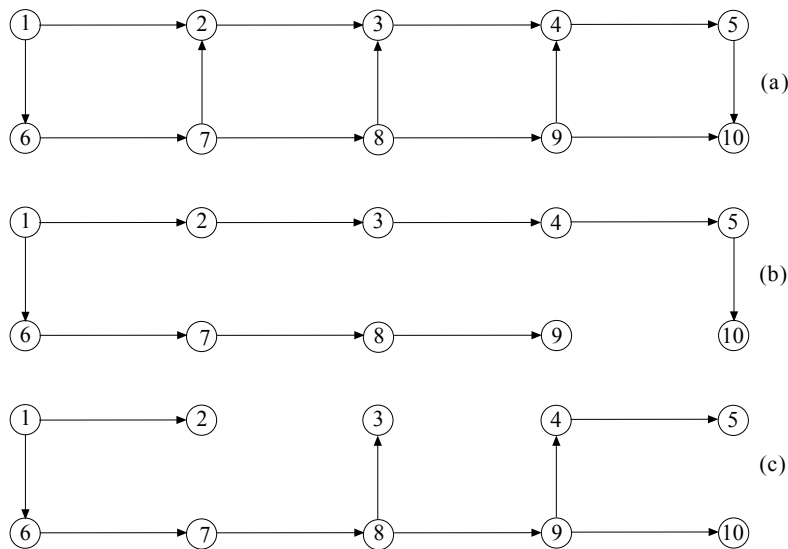
- l'ordine con cui vengono esaminati gli archi della $FS()$ del nodo i estratto da Q , e
- l'ordine con cui vengono estratti i nodi da Q , ossia in che modo l'insieme viene implementato.

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 “lungi”.

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.

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

$p[r] = r; Q = \{r\};$ con **for each**($r \in R$) **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

for each($(i, j) \in FS(i)$) **do** ... in **for each**($(j, i) \in BS(i)$) **do** ...

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).