Algoritmo 3/2-approssimato per TSP metrico

L'algoritmo che segue è dovuto a Christofides (1976). Esso produce una soluzione il cui costo è al più 3/2 volte maggiore del costo della soluzione ottima. Si tratta, cioè, di un algoritmo 3/2-approssimato. L'algoritmo 2-approssimato deve il suo grado di approssimazione al fatto che a un certo punto si opera sul grafo H ottenuto raddoppiando tutti gli archi dell'albero ricoprente di lunghezza minima per G. D'altronde, il raddoppio degli archi assicura che il grafo risultante sia euleriano. L'algoritmo che proponiamo adesso è in grado di costruire un grafo euleriano aggiungendo a T^* il minimo numero di archi necessari, e questa è l'unica sostanziale differenza con l'algoritmo precedente. Ecco i passi dell'algortimo:

1) Trova un albero ricoprente (spanning tree) T^* di lunghezza minima sul grafo G, e sia V' il sottoinsieme dei vertici di T^* con grado dispari;
2) Sia A il sottografo di G indotto da V' (ossia il grafo che ha V' come insieme di nodi e come archi tutti e soli quegli archi di G che hanno entrambi gli estremi in V'); trova un matching (perfetto) M^* di costo minimo su A;
3) Sia H il (multi)grafo che si ottiene aggiungendo a T^* gli archi di M^*, e trova un ciclo euleriano E su H;
4) Determina il ciclo hamiltoniano C che si ottiene visitando i nodi del grafo nell'ordine della loro prima apparizione nel ciclo E.

Il sottoinsieme V' del passo 1) dell'algoritmo è composto da un numero pari di nodi dato che in un qualsiasi grafo il numero di nodi a grado dispari è pari. Per questo stesso motivo, il matching M^* del passo 2) è un matching perfetto. Il grafo H del passo 3) è, generalmente parlando, un multigrafo perché può succedere che alcune coppie siano collegate da più di un arco, precisamente saranno quelle coppie di nodi i e j in cui l'arco (i, j) appartiene sia a T^* sia a M^*. Al passo 3) si deve determinare un ciclo euleriano sul grafo H. Condizione necessaria e sufficiente perché un grafo ammetta un ciclo euleriano è che tutti i suoi vertici abbiano grado pari. Questa condizione è verificata perché H è stato ottenuto dall'albero T^* aggiungendo un arco (di M^*) a ogni nodo di grado dispari. Questa operazione ha aumentato di una unità il grado di tali nodi, trasformandolo da una quantità dispari a una quantità pari. Siccome ai nodi di T^* a grado pari non è stato aggiunto alcun arco, il grafo H è euleriano

Esempio: Si consideri un grafo completo non orientato con $n = 7$ vertici, in cui la lunghezza dell'arco (i, j) è $l_{i,j} = \frac{i+j}{2}$, per $(i, j) \in E$. E' facile verificare che il problema che si definisce su un grafo con queste lunghezze degli archi è un TSP metrico: si segua un ragionamento analogo a quello fatto nell'esempio relativo all'algoritmo 2-approssimato. Applichiamo l'algoritmo 3/2-approssimato appena descritto.

1) Un albero ricoprente di lunghezza minima T^* è la stella con centro il nodo 1, ossia il grafo con 7 nodi e i 6 archi $(1, 2)$, $(1, 3)$, $(1, 4)$, $(1, 5)$, $(1, 6)$, e $(1, 7)$. Il nodo 1 è l'unico nodo a grado pari del grado, dato che ha 6 archi incidenti, tutti gli altri nodi sono a grado dispari avendo 1 unico arco incidente (infatti sono tutti foglie di T^*), dunque $V' = \{2, 3, 4, 5, 6, 7\}$ (si noti che $|V'|$ è pari, come ci aspettavamo)

2) Il grafo A è il grafo completo sui 6 nodi di V'. Un matching perfetto di costo minimo su A è $M^* = \{(2, 5), (3, 6), (4, 7)\}$.

3) Il grafo $H = T^* \cup M^*$ è il grafo con tutti e 7 i nodi e gli archi $\{(1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (1, 7), (2, 5), (3, 6), (4, 7)\}$. Un ciclo euleriano E sul grafo H è, per esempio, quello che vsita, nell'ordine, i nodi: 1, 2, 5, 1, 6, 3, 1, 4, 7, per poi ritornare su 1.

4) Una scorciatoia derivata da E e che non ritorna su nodi già visitati, è quella che visita nell'ordine i nodi 1, 2, 5, 6, 3, 4, 7 (e poi ritorna in 1). Infatti da 1 andiamo in 2, poi in 5, poi per evitare di tornare in 1, prendiamo la scorciatoia che da 5 va direttamente in 6, poi andiamo in 3, da dove, per evitare di tornare in 1, prendiamo la scorciatoia che da 3 va direttamente in 4, da cui andiamo in 7, e infine torniamo in 1. Questo è il ciclo hamiltoniano cercato; il suo costo è $\frac{1+2}{2} + \frac{2+5}{2} + \frac{5+6}{2} + \frac{6+3}{2} + \frac{3+4}{2} + \frac{4+7}{2} + \frac{7+1}{2} = 14$

Questa soluzione 3/2-approssimata determinata dall'algoritmo, ci permette di dire che $c(C^*) \leq 14 \leq 3/2\ c(C^*)$. Da questa relazione possiamo definire Lower e Upper Bound per $c(C^*)$ e dedurre che $2/3\ 14 \leq c(C^*) \leq 14$ e quindi che $9,\overline{3} \leq c(C^*) \leq 14$.

Per dimostrare il fattore di approssimazione dell'algoritmo di Christofides è necessario anteporre il seguente risultato, dove M^* è un matching perfetto di costo minimo per il grafo A indotto dal sottoinsieme V' di nodi a grado dispari in T^*.

Lemma 0.1.

$$c(M^*) \leq c(C^*)/2.$$

Proof. Si consideri un ciclo hamiltoniano C^* di lunghezza minima su G. Sia $\overline{C}$ il ciclo che si ottiene da C^* limitandosi a considerare i nodi di V', ossia applicando una scorciatoia tutte le volte che si deve evitare il passaggio attraverso nodi non appartenenti a V'. Per la disuguaglianza triangolare, si ha che $c(\overline{C}) \leq c(C^*)$. Siccome $|V'|$ è pari e $\overline{C}$ contiene tutti e soli i nodi di V', $\overline{C}$ è un ciclo pari. Come ogni ciclo pari, esso può essere visto come l'unione di due matching perfetti su V', ognuno ottenuto scegliendo in modo alternato gli archi di $\overline{C}$. Siano essi M' e M'', e siano $c(M')$ e $c(M'')$ i loro costi. Evidentemente, $c(M') + c(M'') = c(\overline{C})$. Una semplice relazione aritmetica permette di affermare che $\min\{c(M'), c(M'')\} \leq 1/2c(\overline{C})$. Siccome M^* è un matching perfetto di costo minimo per i nodi in V', mentre nulla si può affermare sulla ottimalità di M' e M'', possiamo senz'altro scrivere che $c(M^*) \leq \min\{c(M'), c(M'')\}$. Riunendo le disequazioni ottenute abbiamo che $c(M^*) \leq \min\{c(M'), c(M'')\} \leq 1/2c(\overline{C}) \leq 1/2 \leq c(C^*)$. □

A questo punto, è possibile dimostrare che l'algoritmo di Christofides ha grado di approssimazione pari a 3/2.

Theorem 0.2. *L'algoritmo di Christofides è un algoritmo 3/2-approssimato per il TSP metrico.*

Proof. Seguendo lo stesso ragionamento della dimostrazione dell'algoritmo 2-approssimato, e sulla base del lemma precedente, possiamo scrivere

$$c(C) \leq c(E) = c(T^*) + c(M^*) \leq c(C^*) + 1/2c(C^*) = \frac{3}{2}c(C^*),$$

da cui la tesi. □

Come importante commmento finale di questo paragrafo, osserviamo che gli algoritmi presentati possono essere applicati a una qualsiasi istanza di TSP infatti ogni passo dell'algoritmo non richiede la verifica di alcun prerequisito.

Nel caso in cui si riesca a dimostrare che l'istanza data è metrica, allora si sa *a priori* che la soluzione determinata dall'algoritmo è approssimata.

Nel caso in cui si dimostri che l'istanza data non è metrica, allora si può solo affermare che la soluzione determinata è ammissibile, ma non si può dire che è approssimata. Infatti, se l'istanza non soddisfa la disuguaglianza triangolare, non si può affermare che $c(C) \leq c(E)$, come fatto nelle due dimostrazioni. Gli algoritmi visti in questo caso si riducono a essere delle euristiche (vedi paragrafo successivo) applicabili a una qualsiasi istanza di TSP.

Gli algoritmi appena visti possono essere applicati a partire da un albero ricoprente qualsiasi, ma anche in questo caso si ottiene solo una soluzione ammissibile che non si può dimostrare essere ottima. Infatti, se si considera una albero ricoprente qualsiasi $\overline{T}$ anziché quello ottimo, non si può affermare che $c(\overline{T}) \leq c(C^*)$. Anche in questo caso gli algoritmi visti si riducono a essere delle euristiche (vedi paragrafo successivo) applicabili a una qualsiasi istanza di TSP.

Algoritmi euristici

Gli algoritmi euristici sono algoritmi che permettono di trovare una soluzione ammissibile al problema, senza garantire né ottimalità né approssimazione (per esempio, gli algoritmi del paragrafo precedente, se applicati a delle istanze di TSP qualsiasi). Noi vedremo il metodo *Greedy*, la *Ricerca Locale* e la *Ricerca Tabù*. Il primo permette di trovare una soluzione ammissibile. Gli altri due partono da una soluzione ammissibile e, se possibile, la migliorano iterativamente: hanno quindi un approccio migliorativo ma non possono funzionare senza che venga data loro una soluzione ammissibile iniziale. Tutti e tre questi metodi euristici vengono descritti in modo molto generale e indipendente dal problema (*meta-euristiche*): per utilizzarli nella pratica occorre definire nel dettaglio le strutture di cui essi hanno bisogno in relazione al problema che si vuole risolvere.

Metodo Greedy

I metodi Greedy fanno parte dei cosiddetti metodi euristici (di cui fanno parte anche la ricerca locale, gli algoritmi di tipo probabilistico, quelli ispirati a fenomeni naturali -come gli algoritmi genetici, ecc.). Il termine *Greedy* vuol dire goloso, e infatti questi algoritmi cercano di condurre l'ottimizzazione scegliendo di volta in volta gli elementi, le variabili che sembrano favorire il miglioramento del valore della funzione obiettivo. Tale comportamento è miope, di solito, perché può succedere che scelte molto promettenti all'inizio costringano successivamente a scelte molto povere. Il vero risultato certo di un algoritmo Greedy è che la soluzione determinata sarà ammissibile per il problema in esame. Proprio per questo, gli algoritmi Greedy vengono spesso utilizzati per determinare un Bound di tipo primale, o per determinare una soluzione ammissibile iniziale per altri algoritmi più sofisticati (ad esempio, gli algoritmi di ricerca locale, che vedremo più avanti) oppure ancora per determinare delle soluzioni sperabilmente buone per problemi di elevata complessità.

Il metodo Greedy viene definito una meta-euristica, perché fornisce uno schema generale di algoritmo dal quale occorre "derivare" l'algoritmo risolutivo vero e proprio specializzando lo schema generale in relazione al problema che si deve risolvere.

Le caratteristiche fondamentali del metodo Greedy sono:

1) Il metodo è *incrementale*: la soluzione viene costruita per gradi a partire da una soluzione "vuota", perché l'algoritmo esamina un elemento alla volta scegliendolo tra quelli non ancora valutati sulla base del criterio descritto al punto 3); dopo essere stato scelto, l'elemento viene valutato e si decide che valore attribuirgli nella soluzione che si sta costruendo (si immagini che l'elemento sia una variabile di una formulazione);
2) Il metodo non torna mai sui propri passi: si dice che è *no-back-tracking*; infatti la decisione che l'elemento faccia parte della soluzione, e con che valore, è definitiva, ovvero non viene mai rimessa in discussione durante l'intera durata dell'algoritmo (quindi non si elimina mai un elemento precedentemente inserito nella soluzione e non si inserisce mai un elemento precedentemente scartato) .
3) La selezione (*Greedy selection*) di un elemento avviene con un meccanismo "goloso" e cioè tra gli elementi non ancora valutati, ne viene scelto di volta in volta uno che rende ottimo il criterio di selezione fissato (*criterio Greedy*). Ogni elemento viene analizzato una sola volta non appena il criterio Greedy lo rende l'elemento più "appetibile" tra quelli non ancora analizzati; in tale momento, si decide in modo definitivo (vedi punti 1 e 2) il suo valore nella soluzione.

In altre parole, l'algoritmo conosce inizialmente un insieme A di elementi che prima dell'inizio dell'algoritmo sono tutti da valutare, e che dopo la fine dell'algoritmo dovranno essere tutti valutati. Istante per istante, quindi, durante l'esecuzione, l'algoritmo mantiene una bi-partizione $< X, Y >$ di A, dove X è l'insieme di quegli elementi di A che sono stati già stati valutati e Y l'insieme di quegli elementi di A ancora in attesa di valutazione. Dunque, all'inizio $Y = A$ e $X = \emptyset$, mentre alla fine $X = A$ e $Y = \emptyset$.

Quindi l'algoritmo si articola nei seguenti passi:

- Definire un criterio Greedy di SCELTA;
- Definire un meccanismo di VALUTAZIONE;
- Inizializzare $Y := A$ e $X := \emptyset$;
- Per $i = 1, \ldots, |A|$ eseguire i seguenti passi:
  scegliere un elemento $a_j \in Y$ utilizzando il criterio Greedy di SCELTA;
  valutare x_j applicando un meccanismo di VALUTAZIONE;
  aggiornare $Y := Y \setminus \{a_j\}$, $X := X \cup \{a_j\}$
- Restituire la soluzione $x = (x_1, x_2, \ldots, x_{|A|})$ determinata

A seconda del criterio Greedy che si sceglie, (naturalmente, tra quelli che hanno senso per il problema in esame), si possono ottenere algoritmi Greedy diversi. Vediamo quindi 2 diversi algoritmi Greedy (in corrispondenza alla definizione di due diversi criteri Greedy) per un problema di Knapsack Binario; e successivamente 2 diversi algoritmi Greedy (in corrispondenza alla definizione di due diversi criteri Greedy) per un problema di Set Covering.

Esempio: Un algoritmo Greedy per un problema di Knapsack Binario. Sia dato il seguente problema:

$$\begin{array}{ll} \max & 15x_1 + 13x_2 + 14x_3 + 11x_4 + 12x_5 \\ s.t. & 5x_1 + x_2 + 2x_3 + 0.5x_4 + 0.6x_5 \leq 6.4 \\ & x \geq 0 \\ & x \leq 1 \text{ e intera} \end{array}$$

Si definiscano i seguenti criterio Greedy di scelta e meccanismo di valutazione:

- SCELTA: "scegliere l'elemento $e_j \in Y$ tale che $c_j = \max\{c_j, e_k \in Y\}$, ossia tale che il suo costo sia il massimo tra tutti gli elementi rimasti (che, per definizione, formano l'insieme Y)
- VALUTAZIONE: se il vincolo è verificato ponendo $x_j = 1$, allora il valore di x_j viene fissato a 1, altrimenti si fissa $x_j = 0$, ossia se il peso dell'oggetto a_j è minore della capacità residuale dello zaino, allora $x_j = 1$, altrimenti $x_j = 0$.

Si ricordi che abbiamo a che fare con variabili binarie; se il problema fosse definito con generiche variabili intere, non appena selezionato un elemento occorrerebbe opportunamente calcolare quale è la giusta quantità da inserire nella soluzione (approfondire questo punto da soli...).

Passo 1) Inizialmente $Y = \{e_1, e_2, e_3, e_4, e_5\}$, e X è vuoto.

- SCELTA: risulta $c_1 = \max\{c_j, a_k \in Y\}$, quindi scegliamo l'oggetto e_1.
- VALUTAZIONE: esso potrà essere inserito nello zaino se è rimasto spazio a sufficienza. In questo caso sì, poiché dal vincolo risulta che il volume dell'oggetto e_1 è $a_1 = 5$ che è ≤ 6.4 (essendo all'inizio del procedimento lo spazio residuo è tutto quello dato).

Quindi, possiamo porre $x_1 = 1$, che è il valore della variabile x_1 avrà nella soluzione che l'algoritmo sta costruendo. Possiamo inserire la condizione $x_1 = 1$ nella formulazione e "applicarla" al vincolo e alla funzione obiettivo senza tuttavia eliminarla per poter poi ricostruire la corretta soluzione finale. Si ottiene:

$$\begin{array}{ll} \max & 13x_2 + 14x_3 + 11x_4 + 12x_5 + 15 \\ s.t. & x_2 + 2x_3 + 0.5x_4 + 0.6x_5 \leq 1.4 \\ & x_1 = 1 \\ & x \geq 0 \\ & x \leq 1 \text{ e intera} \end{array}$$

Ad ogni passo dell'algoritmo conserveremo l'informazione relativa alla soluzione parziale trovata; nel nostro caso, alla fine del Passo 1) si ha: $Y = \{e_2, e_3, e_4, e_5\}$, $X = \{e_1\}$, e la soluzione (parziale)

corrente è $x = (1, -, -, -, -)$, dove "-" indica che l'elemento corrispondente non è ancora stato valutato (infatti i "-" si trovano solo in corrispondenza degli elementi di Y).

Passo 2)

- SCELTA: risulta: $c_3 = \max\{c_j, e_k \in Y\}$, quindi scegliamo l'oggetto 3. Tuttavia,
- VALUTAZIONE: non è possibile inserire l'oggetto e_3 nello zaino, non essendo sufficiente lo spazio residuo (infatti: $a_3 = 2 > 1.4$!). Quindi $x_3 = 0$.

Inserendo e applicando tale condizione nella formulazione si ha:

$$\begin{array}{ll} \max & 13x_2 + 11x_4 + 12x_5 + 15 \\ s.t. & x_2 + 0.5x_4 + 0.6x_5 \leq 1.4 \\ & x_1 = 1 \\ & x_3 = 0 \\ & x \geq 0 \\ & x \leq 1 \text{ e intera} \end{array}$$

Alla fine di questo passo risulta $Y = \{e_2, e_4, e_5\}$, $X = \{e_1, e_3\}$ e la soluzione (parziale) corrente è $x = (1, -, 0, -, -)$.

Passo 3)

- SCELTA: risulta $c_2 = \max\{c_j, e_k \in Y\}$.
- VALUTAZIONE: Possiamo inserire l'oggetto e_2 nello zaino perché il suo volume è inferiore allo spazio residuo nello zaino ($a_2 = 1 \leq 1.4$). Perciò, possiamo assegnare a x_2 il valore 1,

ottenendo:

$$\begin{array}{ll} \max & 11x_4 + 12x_5 + 28 \\ s.t. & 0.5x_4 + 0.6x_5 \leq 0.4 \\ & x_1 = 1 \\ & x_3 = 0 \\ & x_2 = 1 \\ & x \geq 0 \\ & x \leq 1 \text{ e intera} \end{array}$$

A questo punto risulta $Y = \{e_4, e_5\}$, $X = \{e_1, e_2, e_3\}$, e la soluzione è $x = (1, 1, 0, -, -)$.

Passo 4)

- SCELTA: risulta $c_5 = \max\{c_j, e_k \in Y\}$. Tuttavia,
- VALUTAZIONE: non è possibile inserire l'oggetto 5 nello zaino, poiché lo spazio residuo nello zaino non è sufficiente a contenerlo (infatti: $0.6 > 0.4$). Quindi $x_5 = 0$

e risulta:

$$\begin{array}{ll} \max & 11x_4 + 28 \\ s.t. & 0.5x_4 \leq 0.4 \\ & x_1 = 1 \\ & x_3 = 0 \\ & x_2 = 1 \\ & x_5 = 0 \\ & x \geq 0 \\ & x \leq 1 \text{ e intera} \end{array}$$

Adesso si ha $Y = \{a_4\}$, $X = \{a_1, a_2, a_3, a_5\}$, $x = (1, 1, 0, -, 0)$.

Passo 5)

- SCELTA: chiaramente, siccome Y contiene un solo elemento, questo sarà l'elemento scelto per la valutazione
- VALUTAZIONE: Dato che il suo volume non permette di verificare il vincolo, infatti $0.5 > 0.4$, non inseriremo l'oggetto e_4 nello zaino, quindi $x_4 = 0$,

vincolo che aggiunto alla precedente formulazione fornisce:

$$\begin{array}{ll} \max & 28 \\ s.t. & 0 \leq 0.4 \\ & x_1 = 1 \\ & x_3 = 0 \\ & x_2 = 1 \\ & x_5 = 0 \\ & x_4 = 0 \\ & x \geq 0 \\ & x \leq 1 \text{ e intera} \end{array}$$

A questo punto Y è vuoto (infatti tutti gli elementi sono stati valutati, $X = \{e_1, e_2, e_3, e_4, e_5\}$, e la soluzione determinata è $x = (1, 1, 0, 0, 0)$, che ha valore 28.

Come ulteriore commento all'algoritmo Greedy appena visto, diciamo che il criterio Greedy è di tipo *statico*, perché l'ordine con cui vengono esaminati gli elementi è noto a priori, sulla base dei corrispondenti pesi nella funzione obiettivo. La scelta del successivo elemento dipende cioè solo ed esclusivamente dai dati del problema, e non da quali elementi sono stati scelti precedentemente, né dall'ordine con cui sono stati scelti.

Esempio: Un altro algoritmo Greedy per un problema di Knapsack Binario. Sia consideri la stessa istanza di prima:

$$\begin{array}{ll} \max & 15x_1 + 13x_2 + 14x_3 + 11x_4 + 12x_5 \\ s.t. & 5x_1 + x_2 + 2x_3 + 0.5x_4 + 0.6x_5 \leq 6.4 \\ & x \geq 0 \\ & x \leq 1 \text{ e intera} \end{array}$$

Si definiscano i seguenti criterio Greedy di scelta e meccanismo di valutazione:

- SCELTA: "scegliere un elemento $e_j \in Y$ che massimizzi il rapporto utilità/peso $\frac{c_j}{a_j}$ tra tutti gli elementi rimasti, ossia scegliere e_j tale che $\frac{c_j}{a_j} = \max\{\frac{c_k}{a_k}, e_k \in Y\}$
- VALUTAZIONE: se il peso dell'oggetto e_j è minore della capacità residuale dello zaino, allora $x_j = 1$, altrimenti $x_j = 0$.

Anche il criterio appena definito è statico, quindi possiamo calcolare una volta per tutte all'inizio i valori dei rapporti utilità/peso, e determinare di conseguenza l'ordine con cui verranno valutati gli elementi nel corso di questo secondo algoritmo Greedy per il problema di Knapsack Binario. I rapporti utilità/peso risultano: $c_1/a_1 = 15/5 = 3$; $c_2/a_2 = 13/1 = 13$; $c_3/a_3 = 14/2 = 7$; $c_4/a_4 = 11/0.5 = 22$; e $c_5/a_5 = 12/0.6 = 20$. Quindi l'ordine con cui verrano valutati gli elementi è: $e_4 \to e_5 \to e_2 \to e_3 \to e_1$.

Passo 1) Inizialmente $Y = \{e_1, e_2, e_3, e_4, e_5\}$, e $X = \emptyset$

- SCELTA: scelgo da Y l'oggetto e_4 perchè $\frac{c_4}{a_4} = \max\{\frac{c_k}{a_k}, e_k \in Y\}$.
- VALUTAZIONE: poiché il vincolo è verificato ($a_4 = 0.5 \leq 6.4$), $x_4 = 1$,

e ottengo:

$$\begin{array}{ll} \max & 15x_1 + 13x_2 + 14x_3 + 12x_5 + 11 \\ s.t. & 5x_1 + x_2 + 2x_3 + 0.6x_5 \leq 5.9 \\ & x_4 = 1 \\ & x \geq 0 \\ & x \leq 1 \text{ e intera} \end{array}$$

Dopo aver imposto $x_4 = 1$, risulta $Y = \{e_1, e_2, e_3, e_5\}$, $X = \{e_4\}$, e la soluzione attualmente è $x = (-, -, -, 1, -)$.

Passo 2)

- SCELTA: scelgo l'oggetto e_5 perché, tra quelli non ancora valutati (ossia tra quelli di Y) è l'unico che massimizza il criterio fissato, infatti $\frac{c_5}{a_5} = \max\{\frac{c_k}{a_k}, e_k \in Y\}$
- VALUTAZIONE: poiché il vincolo è verificato ($a_5 = 0.6 \leq 5.9$), $x_5 = 1$.

Inserendo e applicando il vincolo nella formulazione si ottiene:

$$\begin{array}{ll} \max & 15x_1 + 13x_2 + 14x_3 + 23 \\ s.t. & 5x_1 + x_2 + 2x_3 \leq 5.3 \\ & x_4 = 1 \\ & x_5 = 1 \\ & x \geq 0 \\ & x \leq 1 \text{ e intera} \end{array}$$

A questo punto risulta $Y = \{e_1, e_2, e_3\}$, $X = \{e_4, e_5\}$, e la soluzione (parziale) corrente è $x = (-, -, -, 1, 1)$.

Passo 3)

- SCELTA: scelgo l'oggetto e_2 perché, tra quelli non ancora valutati è l'unico che massimizza il criterio fissato, infatti $\frac{c_2}{a_2} = \max\{\frac{c_k}{a_k}, e_k \in Y\}$
- VALUTAZIONE: poiché il vincolo è verificato ($a_2 = 1 \leq 5.3$), $x_2 = 1$,

ottenendo:

$$\begin{array}{ll} \max & 15x_1 + 14x_3 + 36 \\ s.t. & 5x_1 + 2x_3 \leq 4.3 \\ & x_4 = 1 \\ & x_5 = 1 \\ & x_2 = 1 \\ & x \geq 0 \\ & x \leq 1 \text{ e intera} \end{array}$$

Risulta $Y = \{e_1, e_3\}$, $X = \{e_2, e_4, e_5\}$, e $x = (-, 1, -, 1, 1)$.

Passo 4)

- SCELTA: scelgo l'oggetto e_3 perché $\frac{c_3}{a_3} = \max\{\frac{c_k}{a_k}, e_k \in Y\}$
- VALUTAZIONE: poiché il vincolo è verificato ($a_3 = 2 \leq 4.3$), $x_3 = 1$

ottenendo:

$$\begin{array}{ll} \max & 15x_1 + 50 \\ s.t. & 5x_1 \leq 2.3 \\ & x_4 = 1 \\ & x_5 = 1 \\ & x_2 = 1 \\ & x_3 = 1 \\ & x \geq 0 \\ & x \leq 1 \text{ e intera} \end{array}$$

Risulta $Y = \{e_1\}$, $X = \{e_2, e_3, e_4, e_5\}$, e $x = (-, 1, 1, 1, 1)$.

Passo 5)

- SCELTA: scelgo l'oggetto e_1 perché $\frac{c_1}{a_1} = \max\{\frac{c_k}{a_k}, e_k \in Y\}$
- VALUTAZIONE: poiché il vincolo non è verificato ($a_1 = 5 > 2.3$), $x_1 = 0$.

Si ottiene:

$$\begin{array}{ll} \max & 50 \\ s.t. & 0 \leq 2.3 \\ & x_4 = 1 \\ & x_5 = 1 \\ & x_2 = 1 \\ & x_3 = 1 \\ & x_1 = 0 \\ & x \geq 0 \\ & x \leq 1 \text{ e intera} \end{array}$$

Risulta $Y = \emptyset$, $X = \{e_1, e_2, e_3, e_4, e_5\}$, e $x = (0, 1, 1, 1, 1)$ con valore $z = 50$.

Nota: Con questo secondo criterio è stato trovato una soluzione ammissibile migliore della precedente perché abbiamo ottenuto una maggiore utilità (il valore della funzione obiettivo). Si noti, tra l'altro, che in questa soluzione abbiamo anche uno zaino meno pesante che nella prima soluzione ($6.4 - 0.4 = 6.0$ Kg della prima a confronto di $6.4 - 2.3 = 4.1$ Kg della seconda). Ma questo fatto certo non interviene nella valutazione della soluzione, che si basa solo sul valore assunto dalla funzione obiettivo.

Vediamo ora l'applicazione dell'algoritmo Greedy a un problema di Set Covering. Anche in questo caso definiremo due algoritmi, con due diversi criteri di selezione, e li applicheremo a uno stesso esempio numerico, che è il seguente:

$$\begin{array}{ll} \min & 3x_1 + 5x_2 + 6x_3 + 2x_4 + x_5 + 7x_6 + x_7 + 8x_8 \\ s.t. & \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{pmatrix} \geq \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \\ & x \geq 0 \\ & x \leq 1 \text{ e intera} \end{array}$$

Ogni colonna della matrice dei coefficienti rappresenta un elemento, mentre ogni riga della matrice rappresenta un sottoinsieme di $A = \{a_1, a_2, a_3, \ldots, a_8\}$. Per esempio il sottoinsieme $A_1 \subseteq A$ descritto nella prima riga è $A_1 = \{a_1, a_3, a_8\}$, il sottoinsieme $A_2 \subseteq A$ descritto nella seconda riga è $A_2 = \{a_2, a_6\}$. La famiglia $\mathcal{F}$ dei sottoinsiemi di tali sottoinsiemi è quindi $\mathcal{F} = \{A_1, A_2, ..., A_8\}$. L'obiettivo è quello di selezionare un sottoinsieme $S = \{a_{j_1}, a_{j_2}, \ldots\}$ in modo tale che venga preso almeno un elemento da ogni sottoinsieme $A_i \in \mathcal{F}$ e che il costo $c(S) = \sum_{a_i \in S} c_i$ di tale sottoinsieme sia minimo.

Definiamo una prima euristica Greedy basata sui seguenti criterio Greedy di SCELTA e meccanismo di VALUTAZIONE:

- SCELTA: tra le variabili non ancora valutate ne scelgo una di costo minimo, ossia scelgo $x_j \in Y$ tale che $c_j = \min\{c_k, a_k \in Y\}$
- VALUTAZIONE: se fissare $x_j = 1$ permette di verificare almeno un vincolo non ancora soddisfatto, allora $x_j = 1$, altrimenti $x_j = 0$

Il criterio di scelta appena descritto permette di conoscere a priori l'ordine con cui verranno valutati gli elementi, infatti tale criterio è *statico*, e l'ordine con cui vengono valutati gli elementi è: $a_5 \rightarrow a_7 \rightarrow a_8 \rightarrow a_4 \rightarrow a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow a_6$. Si noti che se più di un elemento minimizza il criterio, possiamo scegliere arbitrariamente tra di essi (come ad esempio avviene tra a_5 e a_7 prima, e tra a_4 e a_8 poi: in questi casi avremmo potuto scegliere indifferentemente l'ordine $a_5 \rightarrow a_7$ oppure $a_7 \rightarrow a_5$, e/o $a_4 \rightarrow a_8$ piuttosto che $a_8 \rightarrow a_4$).

Il meccanismo di VALUTAZIONE è stato così definito perché attribuire valore 1 a una variabile che non aumenta il numero dei sottoinsiemi coperti (ossia il numero dei vincoli verificati), aumenterebbe il costo della funzione obiettivo, contrariamente al nostro obiettivo di mantenerla più piccola possibile.

In particolare, seguendo l'ordine indicato, scegliamo a_5 da Y, e siccome la scelta di questo elemento permette di verificare il quinto vincolo, fissiamo $x_5 = 1$, ottenendo $Y = \{a_1, a_2, a_3, a_4, a_6, a_7, a_8\}$, $X = \{a_5\}$, e $x = (-, -, -, -, 1, -, -, -)$. Ora tocca ad $a_7 \in Y$; siccome x_7 è presente solo in vincoli già soddisfatti (solo nel quinto) fissiamo $x_7 = 0$, e otteniamo $Y = \{a_1, a_2, a_3, a_4, a_6, a_8\}$, $X = \{a_5, a_7\}$, e $x = (-, -, -, -, 1, -, 0, -)$. Proseguendo in questo modo, il primo algoritmo Greedy fornisce la soluzione $x = (1, 1, 0, 1, 1, 0, 0, 1)$, che, per come è stata costruita, è ammissibile per il problema (infatti i valori delle componenti di x permettono di verificare tutti i vincoli), e ha valore 13.

Definiamo ora una seconda euristica Greedy basata sul seguente criterio Greedy di SCELTA e meccanismo di VALUTAZIONE:

- SCELTA: tra le variabili non ancora valutate scelgo x_j tale che $\frac{c_j}{d_j} = \min\{\frac{c_k}{d_k}, a_k \in Y\}$, dove d_k è il numero di vincoli non ancora soddisfatti in cui la variabile x_h è presente (ossia d_h è il numero di vincoli non ancora soddisfatti che verrebbero verificati ponendo $x_h = 1$);
- VALUTAZIONE: se $d_j > 0$, allora $x_j = 1$, altrimenti $x_j = 0$.

In questo caso, al contrario di tutti i 3 precedenti algoritmi Greedy, abbiamo che la valutazione del criterio deve essere ripetuta a ogni passo, in quanto il valore associato a ogni elemento cambia dopo che un nuovo elemento è stato inserito nella soluzione, dato che il rapporto è funzione del numero d_h dei vincoli non ancora verificati e che verrebbero verificati fissando $x_j = 1$. I precedenti tre criteri infatti erano di tipo statico, mentre questo è di tipo *dinamico*.

Passo 1) Inizialmente $Y = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8\}$, e $X = \emptyset$.

- SCELTA: i rapporti $\frac{c_k}{d_k}$ risultano, nell'ordine, $3/3 = 1$, $5/1 = 5$, $6/2 = 3$, $2/2 = 1$, $1/1 = 1$, $7/3 = 2 + \epsilon$, $1/1 = 1$, e $8/2 = 4$, quindi viene scelto a_1:
- VALUTAZIONE: fissiamo $x_1 = 1$, dato che questa scelta permette di verificare i vincoli I, IV, e VI, non ancora verificati,

ottenendo $Y = \{a_2, a_3, a_4, a_5, a_6, a_7, a_8\}$, $X = \{a_1\}$, e $x = (1, -, -, -, -, -, -, -)$.

Passo 2)

- SCELTA: nell'ordine con cui gli elementi sono scritti nell'insieme Y alla fine del passo precedente, i rapporti risultano $5/1 = 5$, $6/0 = +\infty$, $2/2 = 1$, $1/1 = 1$, $7/1 = 7$, $1/1 = 1$, e $8/1 = 8$ (si noti come sono variati i denominatori, quindi anche i rapporti, rispetto al passo precedente); gli elementi con minor vaore del rapporto sono a_4 e a_5, possiamo scegliere indifferentemente l'uno o l'altro,decidiamo di scegliere a_5
- VALUTAZIONE: fissiamo $x_5 = 1$, dato che permette di verficare il V vincolo che ancora non era verificato

ottenendo così $Y = \{a_2, a_3, a_4, a_6, a_7, a_8\}$, $X = \{a_1, a_5\}$, $x = (1, -, -, -, 1, -, -, -)$.

Si osservi che i denominatori delle frazioni non possono fare altro che rimanere uguali o decrescere passo dopo passo. Quindi, nel momento in cui un denominatore vale 0, dando valore $+\infty$ alla corrispondente frazione, non potendo ulteriormente decrescere, per definizione, non verrà mai più modificato nel corso dell'algoritmo, destinando l'elemento corrispondente a essere considerato

per ultimo, e destinando la corrispondente variabile ad assumere valore 0. L'esito di tale condizione è noto fin dal momento in cui ciò avviene per la prima volta, quindi possiamo senz'altro attribuire valore 0 alle variabili x_h corrispondenti a elementi a_h che hanno $d_h = 0$, non appena questa condizione si verifica. Nel Passo 2) questo può essere fatto per l'elemento a_3. Pertanto il passo 2 si può concludere con i seguenti insiemi: $Y = \{a_2, a_4, a_6, a_7, a_8\}$, $X = \{a_1, a_5, a_3\}$, e $x = (1, -, 0, -, 1, -, -, -)$.

Passo 3)

- SCELTA: nell'ordine con cui gli elementi compaiono nell'insieme Y alla fine del passo precedente, i rapporti risultano $5/1 = 5$, $2/2 = 1$, $7/1 = 7$, $1/0 = +\infty$, e $8/0 = +\infty$; viene quindi scelto a_4;
- VALUTAZIONE: si fissa $x_4 = 1$ perché così facendo si verifica il III vincolo che ancora non era verificato.

Si ottiene $Y = \{a_2, a_6\}$, $X = \{a_1, a_3, a_4, a_5, a_7, a_8\}$, e $x = (1, -, 0, 1, 1, -, 0, 0)$.

Passo 4)

- SCELTA: nell'ordine con cui gli elementi compaiono nell'insieme Y alla fine del passo precedente, i rapporti risultano $5/1 = 5$ e $7/1 = 7$; viene quindi scelto a_2
- VALUTAZIONE: si pone $x_2 = 1$ perché così facendo si verfica il II vincolo che ancora non era verificato.

Si ottiene $Y = \{a_6\}$, $X = \{a_1, a_2, a_3, a_4, a_5, a_7, a_8\}$, e $x = (1, 1, 0, 1, 1, -, 0, 0)$.

Passo 5)

- SCELTA: l'unico elemento attualmente in Y ha rapporto pari a $7/0 = +\infty$; quindi
- VALUTAZIONE: si pone $x_6 = 0$.

L'algoritmo si conclude con $Y = \emptyset$, $X = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8\}$, $x = (1, 1, 0, 1, 1, 0, 0, 0)$ e valore della funzione obiettivo pari a 11.

Ricerca Locale

La Ricerca Locale è un metodo euristico detto euristica di scambio (*interchange heuristics*). Anche la Ricerca Locale deve essere considerata una meta-euristica, nel senso che fornisce uno schema dal quale viene poi estratto l'algoritmo vero e proprio, una volta che sono stati specializzati i vari parametri al problema da risolvere. Il procedimento generale di un algoritmo di Ricerca Locale è il seguente:

- Sia S una soluzione iniziale;
- Si calcoli l'intorno $\mathcal{I}(S)$;
- Sia S' la migliore soluzione dell'intorno $\mathcal{I}(S)$;
- Se S' è migliore di S,
  allora $S := S'$ e si torni all'inizio;
  altrimenti S è la migliore soluzione determinata, stop

L'algoritmo di Ricerca Locale ha bisogno di una soluzione iniziale. Tali soluzioni iniziali possono essere determinate in un modo qualsiasi (applicando un particolare algoritmo, in modo casuale, ...); un'idea è di applicare un Greedy.

La definizione di $\mathcal{I}(S)$, l'intorno di S, dipende dal problema. In particolare, indicando con $\overline{S}$ l'insieme degli elementi che non fanno parte della soluzione S:

Definition 0.3. *Data una soluzione S, si definisce intorno di S, $\mathcal{I}(S)$, l'insieme di tutte le soluzioni ammissibili $\tilde{S}$ ottenibili da S tramite operazioni di:*

- *Rimozione* $\tilde{S} := S \setminus X$*, con* $X \subseteq S$ *e* $|X|$ *"piccola";*
- *Inserimento* $\tilde{S} := S \cup Y$*, con* $Y \subseteq \overline{S}$ *e* $|Y|$ *"piccola";*
- *Scambio:* $\tilde{S} := (S \setminus X) \cup Y$*, con* $X \subseteq S$*,* $Y \subseteq \overline{S}$*,* $|X| = |Y|$*, e* $|X|, |Y|$ *"piccole".*

Si osservi che se la cardinalità di una soluzione S è fissata a priori, allora $\mathcal{I}(S)$ può essere ottenuto solo attraverso operazioni di scambio, in cui, in particolare, $|X = |Y|$.

Che significano i termini "piccola" e "piccole"? Per quantificare tale numero, bisogna tener presente che il numero di elementi in X e/o in Y influenza la dimensione di $\mathcal{I}(S)$, nel senso che tanto più è piccolo è $\mathcal{I}(S)$, tante più iterazioni occorre aspettarsi che vengano effettuate, in generale, prima che l'algoritmo si fermi.

E ancora, il fatto che $|X|$ e $|Y|$ siano piccole quantità ci dice che ogni soluzione $\tilde{S} \in \mathcal{I}(S)$ così determinata non differisce tanto da S.

Per quanto riguarda, infine, il calcolo del costo di ogni soluzione $\tilde{S} \in \mathcal{I}(S)$, si possono seguire due metodi.

Il primo consiste nel calcolo di $c(\tilde{S})$ a partire da tutti i suoi elementi.

Il secondo consiste nel calcolare $c(\tilde{S})$ a partire dal costo $c(S)$ di S, sottraendo i costi c_j degli elementi eliminati (ossia degli $e_j \in X$), e aggiungendo i costi c_j degli elementi inseriti (ossia degli $e_j \in Y$). Formalmente:

$$c(\tilde{S}) = c(S) - \sum_{e_j \in X} c_j + \sum_{e_j \in Y} c_j.$$

Siccome $|X|$ e $|Y|$ sono piccole quantità, vuol dire che la maggior parte degli elementi che era in S lo ritroviamo anche in $\tilde{S}$, quindi il calcolo comprenderà un piccolo numero di addendi (precisamente $1 + |X| + |Y|$). Questo metodo è detto *metodo incrementale* per il calcolo dei costi, e contribuisce a rendere meno onerosa possibile ciascuna iterazione del'algoritmo di Ricerca Locale, dove per *iterazione* si intende la determinazione dell'intorno, la valutazione di ogni sua soluzione, e la scelta della (di una, più in generale) sua soluzione migliore.

Vediamo un esempio.

Esempio: Ricerca Locale applicata al problema del TSP. Si consideri un problema di TSP definito su un grafo completo, orientato (e non simmetrico, generalmente parlando), nel quale, cioè, presa una qualsiasi coppia di nodi, esistono gli archi in entrambe le direzioni e la loro lunghezza è, in generale, diversa (in altre parole, la matrice di adiacenza del grafo il cui generico elemento $l_{i,j}$ indica la distanza associata all'arco (i, j), non è simmetrica, perché esiste almeno una coppia di nodi i e j tale che $l_{i,j} \neq l_{j,i}$).

Un'euristica di Ricerca Locale molto famosa per il TSP su un grafo completo, orientato, e qunidi, in generale, non simmetrico è quella chiamata *3-arcs interchange*. In questa euristica, l'intorno $\mathcal{I}(C)$ di un ciclo hamiltoniano C è l'insieme di tutti i cicli hamiltoniani $\neq C$ che si ottengono da C rimuovendo 3 archi e reinserendone (opportunamente) altri 3. Osserviamo che è importante che gli archi rimossi e gli archi reinseriti siano in ugual numero perché ogni ciclo hamiltoniano su n nodi contiene, per definizione, n archi (questo è un esempio di intorno che viene definito solo attraverso operazioni di scambio).

L'euristica 3-arcs interchange si presenta così:

- Sia C un ciclo hamiltoniano per il grafo dato G;
- Costruisci l'intorno $\mathcal{I}(C)$ di C;
- Calcola il costo di ogni soluzione $\in \mathcal{I}(C)$, e scegline una di costo minimo, sia essa C'.
- Se il costo $c(C')$ è migliore di $c(C)$
  allora $C := C'$ e ritorna all'inizio
  altrimenti fermati: C è il miglior ciclo hamiltoniano determinato

$\mathcal{I}(C)$ è composto da tutti (e soli) i cicli hamiltoniani $\neq C$ che si ottengono da C rimuovendo tre archi di C in tutti i modi possibili e inserendone opportunamente altri tre.

Supponiamo che il grafo dato abbia 6 nodi, e che C, il ciclo hamiltoniano iniziale sia quello che attraversa in sequenza i nodi $1 \to 2 \to 3 \to 4 \to 5 \to 6 \to (1)$ (il nodo finale, uguale a quello iniziale, è racchiuso tra parentesi per ricordare che il ciclo si richiude). L'insieme $E(C)$ degli archi del ciclo è $E(C) = \{(1,2), (2,3), (3,4), (4,5), (5,6), (6,1)\}$ (coppie ordinate perché gli archi sono orientati). Costruiamo l'intorno $\mathcal{I}(C)$ di C. Gli elementi di $\mathcal{I}(C)$ sono tutti i cicli che si trovano nella terza colonna della tabella che segue.

insieme X degli archi rimossi	insieme Y degli archi inseriti	ordine di visita dei nodi nel nuovo ciclo	costo del nuovo ciclo
$\{(1,2),(2,3),(3,4)\}$	$\{(1,3),(3,2),(2,4)\}$	$1 \to 3 \to 2 \to 4 \to 5 \to 6 \to (1)$	...
$\{(1,2),(2,3),(4,5)\}$	$\{(1,3),(4,2),(2,5)\}$	$1 \to 3 \to 4 \to 2 \to 5 \to 6 \to (1)$	...
$\{(1,2),(2,3),(5,6)\}$	$\{(1,3),(5,2),(2,6)\}$	$1 \to 3 \to 4 \to 5 \to 2 \to 6 \to (1)$	...
$\{(1,2),(2,3),(6,1)\}$	$\{(1,3),(6,2),(2,1)\}$	$1 \to 3 \to 4 \to 5 \to 6 \to 2 \to (1)$	...
$\{(1,2),(3,4),(4,5)\}$	$\{(1,4),(4,2),(3,5)\}$	$1 \to 4 \to 2 \to 3 \to 5 \to 6 \to (1)$	...
$\{(1,2),(3,4),(5,6)\}$	$\{(1,4),(5,2),(3,6)\}$	$1 \to 4 \to 5 \to 2 \to 3 \to 6 \to (1)$	...
$\{(1,2),(3,4),(6,1)\}$	$\{(1,4),(6,2),(3,1)\}$	$1 \to 4 \to 5 \to 6 \to 2 \to 3 \to (1)$	...
$\{(1,2),(4,5),(5,6)\}$	$\{(1,5),(5,2),(4,6)\}$	$1 \to 5 \to 2 \to 3 \to 4 \to 6 \to (1)$	...
$\{(1,2),(4,5),(6,1)\}$	$\{(1,5),(6,2),(4,1)\}$	$1 \to 5 \to 6 \to 2 \to 3 \to 4 \to (1)$	...
$\{(1,2),(5,6),(6,1)\}$	$\{(1,6),(6,2),(5,1)\}$	$1 \to 6 \to 2 \to 3 \to 4 \to 5 \to (1)$	...
$\{(2,3),(3,4),(4,5)\}$	$\{(2,4),(4,3),(3,5)\}$	$1 \to 2 \to 4 \to 3 \to 5 \to 6 \to (1)$	...
$\{(2,3),(3,4),(5,6)\}$	$\{(2,4),(5,3),(3.6)\}$	$1 \to 2 \to 4 \to 5 \to 3 \to 6 \to (1)$	...
$\{(2,3),(3,4),(6,1)\}$	$\{(2,4),(6,3),(3.1)\}$	$1 \to 2 \to 4 \to 5 \to 6 \to 3 \to (1)$	...
$\{(2,3),(4,5),(5,6)\}$	$\{(2,5),(5,3),(4,6)\}$	$1 \to 2 \to 5 \to 3 \to 4 \to 6 \to (1)$	...
$\{(2,3),(4,5),(6,1)\}$	$\{(2,5),(6,3),(4,1)\}$	$1 \to 2 \to 5 \to 6 \to 3 \to 4 \to (1)$	...
$\{(2,3),(5,6),(6,1)\}$	$\{(2,6),(6,3),(5,1)\}$	$1 \to 2 \to 6 \to 3 \to 4 \to 5 \to (1)$	...
$\{(3,4),(4,5),(5,6)\}$	$\{(3,5),(5,4),(4,6)\}$	$1 \to 2 \to 3 \to 5 \to 4 \to 6 \to (1)$	...
$\{(3,4),(4,5),(6,1)\}$	$\{(3,5),(6,4),(4,1)\}$	$1 \to 2 \to 3 \to 5 \to 6 \to 4 \to (1)$	...
$\{(3,4),(5,6),(6,1)\}$	$\{(3,6),(6,4),(5,1)\}$	$1 \to 2 \to 3 \to 6 \to 4 \to 5 \to (1)$	...
$\{(4,5),(5,6),(6,1)\}$	$\{(4,6),(6,5),(5,1)\}$	$1 \to 2 \to 3 \to 4 \to 6 \to 5 \to (1)$	...

Si osservi che le seguenti notazioni indicano tutte lo stesso ciclo:

$$1 \to 2 \to 6 \to 3 \to 4 \to 5 \to (1) \qquad 2 \to 6 \to 3 \to 4 \to 5 \to 1 \to (2)$$
$$6 \to 3 \to 4 \to 5 \to 1 \to 2 \to (6) \qquad 3 \to 4 \to 5 \to 1 \to 2 \to 6 \to (3)$$
$$4 \to 5 \to 1 \to 2 \to 6 \to 3 \to (4) \qquad 5 \to 1 \to 2 \to 6 \to 3 \to 4 \to (5)$$

Si osservi che in corrispondenza ad ogni tripla di archi selezionati dal ciclo C ottengo un unico ciclo diverso dal ciclo C stesso. Si consideri, per esempio il ciclo C originario, e sia $X = \{(1,2),(4,5),(6,1)\}$. Ora, se inseriamo gli archi $(1,2),(4,5),(6,1)$ otteniamo evidentemente lo stesso ciclo C, quindi questo inserimento non va bene. Allora ci chiediamo se va bene inserire gli archi $(1,2),(4,1),(6,5)$: in questo caso si chiuderebbero due sottocicli, ossia il grafo C' che ha $E(C') = (E(C) \setminus X) \cup Y$ come insieme di archi, non è un (unico) ciclo che tocca tutti i nodi (ossia hamiltoniano), quindi non è soluzione ammissibile del TSP. Dunque anche questa scelta non va bene. L'unica alternativa rimasta, che è anche l'unica corretta, è quella di inserire gli archi $(4,1),(1,5),(6,2)$ al posto degli archi $(1,2),(4,5),(6,1)$, cosa che dà origine al ciclo C' che ha $E(C') = (E(C) \setminus X) \cup Y = \{(1,2), (2,3), (3,4), (4,5), (5,6), (6,1)\} \setminus \{(1,2), (4,5), (6,1)\} \cup \{(4,1), (1,5), (6,2)\} = \{(2,3), (3,4), (5,6), (4,1), (1,5), (6,2)\}$, come insieme di archi. Questo è il ciclo (hamiltoniano, come ora verfichiamo) che visita i nodi nell'ordine $1 \to 5 \to 6 \to 2 \to 3 \to 4 \to (1)$.

Nel momento in cui costruiamo ognuno dei nuovi 20 cicli ne valutiamo il costo (non indicato nella tabella) pari alla somma dei pesi degli archi che lo compongono. Una volta che abbiamo determinato e valutato tutti i cicli dell'intorno $\mathcal{I}(C)$ di C, possiamo sceglierne uno di costo minimo. Supponiamo che un ciclo hamiltoniano a costo minimo sia il k-esimo della tabella, e sia $c(C_k)$ il suo costo. Ora occorre valutare se $c(C_k) < c(C)$ oppure se $c(C_k) \geq c(C)$. Nel primo caso si pone $C := C_k$ e si riparte (ossia C_k diventa una nuova soluzione ammissibile da cui partire con *un'altra iterazione* della Ricerca Locale, quindi occorrerà calcolare un nuovo intorno, valutarne tutti i cicli, ...). Nel secondo caso invece possiamo senz'altro affermare che nessuna soluzione dell'intorno di C è migliore (secondo la nostra definizione di intorno) di C stessa, quindi l'algoritmo si ferma e propone C come migliore soluzione determinata nel corso dell'intero algoritmo.

E' importante osservare che qualunque sia la tripla di archi eliminati esiste sempre una tripla di archi il cui inserimento genera un ciclo hamiltoniano diverso da quello da cui si era partiti. L'unico motivo per cui ciò potrebbe non succedere è che il grafo non sia completo, e in particolare che dal grafo manchino uno o più dei 3 archi che, se inseriti al posto di quelli rimossi, permettono di ottenere un ciclo hamiltoniano. Ma tutto questo è impossibile perché il TSP è sempre definito su un grafo completo.

Per quanto riguarda la cardinalità $|\mathcal{I}(C)|$ di $\mathcal{I}(C)$, possiamo osservare che siccome non importa l'ordine con cui vengono eliminati dal ciclo gli archi, risulta $|\mathcal{I}(C)| = n(n-1)(n-2)/3!$, che per $n = 6$ è pari a 20, come mostrato in tabella.

Per quanto riguarda il calcolo del costo del generico nuovo ciclo $C' \in \mathcal{I}(C)$, guardiamo quante operazioni facciamo per calcolare il costo $c(C')$ di un ciclo $C' \in \mathcal{I}(S)$ a partire da tutti i suoi elementi, e quante ne faremmo se applicassimo il metodo incrementale. Nel primo caso abbiamo una sommatoria di n termini se il grafo ha n nodi, infatti $c(C') = \sum_{(i,j)\in E(C')} l_{i,j}$, dove $l_{i,j}$ rappresenta la lunghezza dell'arco orientato (i, j) (questa è una sommatoria con 100 termini per un grafo di 100 nodi, è una sommatoria di 3000 termini per un grafo di 3000 nodi, ...).

Se invece adottiamo il metodo incrementale, siccome nell'euristica 3-arcs interchange si ha $|X| = |Y| = 3$, tutti gli archi di C si trovano anche nel ciclo C', tranne i $|X| = 3$ che abbiamo rimosso, i cui costi vanno rimpiazzati dai costi dei $|Y| = 3$ archi inseriti *ex-novo*. Per questo possiamo prendere il costo di C, sottrargli il costo degli archi eliminati (quelli dell'insieme X), e aggiungere il costo dei nuovi archi inseriti (quelli dell'insieme Y), ottenendo

$$c(C') = c(C) - \sum_{(i,j)\in X} l_{i,j} + \sum_{(i,j)\in Y} l_{i,j}$$

Questa sommatoria ha sempre solo 7 termini, indipendentemente dal numero n di nodi del grafo. Ciò presenta grandi vantaggi, soprattutto al crescere del numero dei nodi dell'istanza, e fa sì che il tempo speso nel calcolo dei costi ammonti a $7|\mathcal{I}(S)|$, quantità che è dell'ordine $O(n^3)$ per ogni iterazione dell'algoritmo, da confrontarsi con $O(n|\mathcal{I}(S)|) = O(n^4)$ che è il tempo speso nel calcolo dei costi senza l'utilizzo del calcolo incrementale.

Un ulteriore vantaggio dal punto di vista della complessità dei calcoli si ha andando a valutare solo la variazione $\Delta(C', C)$ del costo di ogni ciclo dell'intorno di C rispetto al costo di C stesso. Evidentemente $\Delta(C', C) = c(C') - c(C) = -\sum_{(i,j)\in X} l_{i,j} + \sum_{(i,j)\in Y} l_{i,j}$ e può essere una quantità positiva, negativa, o nulla (e il ciclo C' sarà rispettivamente peggiore, migliore o uguale al ciclo C). Il migliore ciclo $C' \in \mathcal{I}(C)$ è uno a cui corrisponde un $\Delta(C', C)$ minimo, e solo se $\Delta(C'C) < 0$ C' è migliore di C.

Si noti, infine, che non è possibile sapere con certezza prima dell'inizio dell'esecuzione quante saranno le iterazioni che l'algoritmo effettuerà, perché il criterio di arresto dell'algoritmo non dipende da scelte nostre.

Per concludere, diciamo che l'euristica 3-arcs interchange è la più piccola euristica di Ricerca Locale per un problema di TSP orientato. Lasciamo al lettore il compito di verificare che se si decidesse di scambiare solo uno o due archi tra loro, l'intorno della soluzione iniziale C sarebbe vuoto.

Esempio: Symmetric Travelling Salesman Problem (S-TSP). Lo studio di una euristica di Ricerca Locale al problema del TSP su un grafo completo orientato e simmetrico (simmetrica è, quindi, la matrice di adiacenza il cui generico elemento $l_{i,j}$ indica la distanza associata all'arco (i, j)) è importante perché mostra come la definizione dell'intorno vari al variare delle caratteristiche del problema da risolvere.

L'unica cosa che differenzia il TSP dall'S-TSP è che in quest'ultimo i due archi che connettono, con orientamento diverso, una stessa coppia di nodi, hanno la stessa lunghezza, cioè $l_{i,j} = l_{j,i}$ per ogni coppia di nodi i e j. Naturalmente è possibile applicare l'euristica di scambio dei tre archi, ma in effetti è possibile anche applicare una (più semplice) euristica di scambio di due soli archi (*2-arcs interchange heuristic*).

Consideriamo un ciclo hamiltoniano iniziale C che visiti i nodi nell'ordine $1 \to 2 \to 3 \to 4 \to 5 \to 6 \to 7 \to 8 \to (1)$ ovvero che sia composto dagli archi $\{(1,2), (2,3), (3,4), (4,5), (5,6), (6,7), (7,8), (8,1)\}$. L'intorno $\mathcal{I}(C)$ di C è composto da tutti i cicli hamiltoniani $C' \neq C$ che si ottengono da C rimuovendo due archi e inserendone (opportunamente) altri due. Ad esempio, rimuovendo

$$X = \{(4,5), (8,1)\}$$

le tre coppie di archi che si possono reinserire sono mostrate qua sotto. L'inserimento della prima coppia restituisce il ciclo hamiltoniano di partenza. L'inserimento della terza coppia non dà luogo a un ciclo hamiltoniano perché dà luogo a due sottocicli, che è una soluzione non ammissibile per il nostro S-TSP: questa soluzione, siccome non è un ciclo hamiltoniano, non appartiene a $\mathcal{I}(C)$.

$$Y = \begin{cases} \{(4,5),(8,1)\} & C' = 1 \to 2 \to 3 \to 4 \to 5 \to 6 \to 7 \to 8 \to (1) = C \\ \{(4,8),(5,1)\} & C' = 1 \to 2 \to 3 \to 4 \to 8 \to 7 \to 6 \to 5 \to (1) \\ \{(4,1),(5,8)\} & 1 \to 2 \to 3 \to 4 \to (1), \quad 5 \to 6 \to 7 \to 8 \to (5) \text{ non ammissibile} \end{cases}$$

Lasciamo al lettore il compito di verificare che non si possono scegliere due archi adiacenti perchè in questo caso non si riesce a ricostruire un ciclo hamiltoniano diverso da quello di partenza. Per questo motivo l'intorno $\mathcal{I}(C)$ di C è composto da tutti i cicli hamiltoniani diversi da C' che si ottengono da C attraverso lo scambio di due archi non adiacenti. Il loro numero è $|\mathcal{I}(C)| = n(n-3)/2$. Infatti il primo arco può essere scelto in tutti gli n modi possibili; il secondo, invece, deve essere scelto non adiacente al primo tra gli $n-1$ rimanenti. Siccome gli archi adiacenti al primo sono 2, rimaniamo con $n-3$ possibilità di scelta. Infine, dividiamo per 2 perchè non importa l'ordine con cui gli archi vengono scelti. Come sempre, occorre determinarli tutti, valutarne il costo (lasciamo al lettore il compito di vedere come possa essere condotto in modo incrementale il calcolo del costo di ogni ciclo $\in \mathcal{I}(C)$ a partire dal costo di C), scegliere il migliore, eventualmente ripartire con un'altra iterazione, etc. etc.

Esempio (facoltativo): In questo esempio mostriamo come applicare un algoritmo di Ricerca Locale al problema della Partizione Uniforme di un Grafo (in breve UGP, Uniform Graph Partitioning). Il problema è così definito:

Dato: un grafo $G = (V, E)$ con pesi $d_{i,j}$ per ogni arco $(i,j) \in E$ e con $|V| = 2n$ nodi;

Trovare: una partizione $< A, B >$ dell'insieme V dei nodi;

In modo tale che: $|A| = |B|$ e sia minimo il costo $c(< A, B >) = \sum_{(i,j)\in E:\, i\in A, j\in B} d_{i,j}$ della partizione, ossia sia minima la somma dei pesi degli archi che hanno estremi in classi diverse

Per partizione $< A, B >$ di V si intende la suddivisione di V in due *classi*, ossia in due sottoinsiemi, A e B tali che $A \cup B = \emptyset$ e $A \cap B = V$. Si osservi poi che siccome abbiamo a che fare con una bi-partizione, B è univocamente determinato a partire da A, o viceversa: per esempio $B = V \setminus A = \{v \in V, v \notin A\}$, perché segue dalla definizione di bi-partizione che tutti i nodi che non sono nell'insieme A si trovano nell'insieme B. Quindi, una bi-partizione $< A, B >$, per brevità, può essere rappresentata anche attraverso una sola delle sue parti.

Se non ci fosse il vincolo $|A| = |B|$, il problema sarebbe risolubile polinomialmente con un qualsiasi algoritmo di massimo flusso-minimo taglio (max flow-min cut) che, quindi, rappresenta un rilassamento (di tipo combinatorio) del problema UGP). Il vincolo $|A| = |B|$ complica enormemente le cose. Tuttavia, notare questo è importante perché rimuovendo il vincolo $|A| = |B|$ si ottiene un problema di max flow-min cut che, risolto all'ottimo, ci fornisce un Lower Bound per il valore ottimo $c^*(A, B)$ della funzione obiettivo (è un esempio di rilassamento di tipo combinatorio).

Infine, scrivere $|V| = 2n$ serve per assicurarsi che $|V|$ sia pari, altrimenti non si può successivamente imporre che $|A| = |B|$.

Supponiamo di avere il grafo descritto dalla seguente matrice di adiacenza nodi-nodi (dove il generico elemento $d_{i,j}$ rappresenta il peso del corrispondente arco (se l'elemento è assente vuol dire che l'arco non fa parte del grafo), e la matrice è simmetrica perché il grafo è non orientato):

$D =$

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
1		5										9
2	5		2	6						6	6	4
3		2		8	8							5
4		6	8		9							
5			8				8					
6				9			4		5	6		
7					8	4		3		1		
8							3					
9						5						
10		6				6	1				2	
11		6								2		
12	9	4	5									

Per poter applicare l'algoritmo di Ricerca Locale dobbiamo considerare una partizione iniziale e dobbiamo definire le modalità di costruzione dell'intorno di una qualsiasi soluzione ammissibile.

Sia, quindi, $< A, B >=< \{1, 8, 9, 10, 11, 12\}, \{2, 3, 4, 5, 6, 7\} >$ la partizione iniziale. Il costo di tale partizione è: $c(A, B) = d_{1,2} + d_{8,7} + d_{9,6} + d_{10,2} + d_{10,6} + d_{10,7} + d_{11,2} + d_{12,2} + d_{12,3} = 5 + 3 + 5 + 6 + 6 + 1 + 6 + 6 + 4 + 5 = 41$. Si noti che se estraiamo dalla matrice di adiacenza la sottomatrice che ha come righe tutte e sole quelle corrispondenti ai nodi in A e come colonne tutte e sole quelle corrispondenti ai nodi di B, la somma degli elementi non nulli di questa sottomatrice è esattamente $c(A, B)$. Infatti i soli elementi (non nulli) della sottomatrice sono tutti e soli gli archi che connettono un nodo di A (che corrisponde a una riga della matrice) con un nodo di B (che corrisponde a una colonna della matrice). La sottomatrice $D(< A, B >)$ relativa alla partizione in questione è la seguente:

$D(< A, B >) =$

	2	**3**	**4**	**5**	**6**	**7**
1	5					
8						3
9					5	
10	6				6	1
11	6					
12	4	5				

Passiamo ora alla definizione dell'intorno. Anche in questo caso, come negli esempi del TSP e dell'S-TSP, la cardinalità della soluzione è fondamentale per l'ammissibilità. Come conseguenza l'intorno può venire costruito solo attraverso operazioni di scambio in cui i due insiemi X e Y sono vincolati ad avere la stessa cardinalità. Bisogna fissare tale quantità, per esempio $|X| = |Y| = 1$. Dunque: $\mathcal{I}(A, B) = \{< A', B' >$ tali che $A' = (A \setminus \{x\}) \cup \{y\}$, e $B' = (B \setminus \{y\}) \cup \{x\}$, con $x \in A$ e $y \in B\}$.

Poiché il grafo ha $2n$ nodi, che vanno suddivisi equamente tra A e B, il nodo $x \in A$ può essere scelto in n modi diversi, così come $y \in B$ può essere scelto in altrettanti modi diversi. Per questo motivo, $|\mathcal{I}(A, B)| = n^2$.

Mentre si calcolano tutti le n^2 partizioni di $\mathcal{I}(A, B)$ occorre valutare il costo di ciascuna per poter poi scegliere la migliore. Per valutare il costo $c(A', B')$ della generica partizione $< A', B' > \in \mathcal{I}(A, B)$ si può applicare la definizione secondo la quale $c(A', B') = \sum_{(i,j) \in E:\ i \in A', j \in B'} d_{i,j}$ oppure si può procedere in *modo incrementale* calcolando quali sono (in più e in meno) gli archi che rendono $< A', B' >$ diversa da $< A, B >$. Come abbiamo già osservato nel caso del TSP e dell'S-TSP, questa seconda scelta è di solito più conveniente. Nell'esempio in questione la situazione è un pochino più complessa da descrivere di quanto non fosse nel caso dei cicli hamiltoniani, ma si basa sugli stessi concetti.

In particolare consideriamo quei nodi $x \in A$ e $y \in B$ il cui scambio ha dato luogo alla partizione $< A', B' >$. Fintanto che $x \in A$ gli archi incidenti su x che contribuiscono alla funzione obiettivo

sono tutti e soli quelli che collegano x con nodi di B: chiameremo costo esterno di x la somma dei loro pesi:

$$\text{per } x \in A, \text{ il costo esterno è } E(x) = \sum_{j \in B} d_{x,j}.$$

Al contrario, gli archi che incidono su x e che lo collegano con nodi di A stesso non contribuiscono alla funzione obiettivo, per definizione: chiameremo costo interno di x la somma dei loro pesi:

$$\text{per } x \in A, \text{ il costo interno è } I(x) = \sum_{h \in A} d_{x,h}.$$

Analogo discorso si può fare per il nodo $y \in B$, ribaltando il ruolo degli insiemi A e B Per cui si ha

$$\text{per } y \in B, \text{ il costo esterno è } E(y) = \sum_{j \in A} d_{y,j}$$

$$\text{per } y \in B, \text{ il costo interno è } E(y) = \sum_{h \in B} d_{y,h}.$$

Per organizzare i dati a disposizione, può essere utile arricchire la matrice $D(A, B)$ con le informazioni sui costi interni ed esterni di ciascun nodo:

$D(A,B) =$

		$I(\cdot)$	8	18	23	25	4	12
		$E(\cdot)$	21	5	0	0	11	4
$I(\cdot)$	$E(\cdot)$		**2**	**3**	**4**	**5**	**6**	**7**
9	5	**1**	5					
0	3	**8**						3
0	5	**9**					5	
2	13	**10**	6				6	1
2	6	**11**	6					
0	9	**12**	4	5				

Nel momento in cui spostiamo x dall'insieme A all'altro insieme, il nodo x contribuirà alla nuova funzione obiettivo, cioè al costo della partizione $< A', B' >$, con quello che precedentemente era il costo interno $I(x)$ e non contribuirà più con quello che precedentemente era il costo esterno $E(x)$. Stesso discorso vale nel momento in cui spostiamo il nodo y dall'insieme B all'altro insieme.

In definitiva il costo $c(A', B')$ della nuova partizione $< A', B' > \in \mathcal{I}(A, B)$ può essere così calcolato a partire dal costo $c(A, B)$ della partizione $< A, B >$ in questo modo:

$$c(A', B') = c(A, B) - E(x) + I(x) - E(y) + I(y) + 2d_{x,y}$$

dove l'ultimo termine $2d_{x,y}$ vale 0 se x e y non sono collegati da un arco (infatti in tal caso $d_{x,y} = 0$), mentre se $d_{x,y} \neq 0$, la sua presenza è necessaria per il motivo che ora spieghiamo. Se $d_{x,y} \neq 0$ vuol dire che $x \in A$ e $y \in B$ sono adiacenti perché collegati da un arco il cui peso è $d_{x,y}$,. Siccome i due nodi appartengono a insiemi diversi, il contributo $d_{x,y}$ dell'arco che li collega è (e deve essere) presente in $c(A, B)$. Quando scambiamo tra loro i nodi x e y, otteniamo la partizione $< A', B' >$ dove, nuovamente, il contributo $d_{x,y}$ dell'arco che li collega è (e deve essere) presente in $c(A', B')$ perché è il contributo che collega due nodi in insiemi diversi (infatti ora $x \in B'$, e $y \in A'$). Tuttavia, nella espressione scritta sopra per $c(A', B')$, tra le righe c'é scritto che la quantità $d_{x,y}$ viene sottratta due volte perché è uno dei termini della sommatoria che definisce il costo esterno $E(x)$ di x, ed è anche uno dei termini della sommatoria che definisce il costo esterno $E(y)$ di y. La presenza del termine $2d_{x,y}$ serve proprio a compensare questo "difetto".

La matrice che comprende anche i dati sui costi interni ed esterni rende molto agevole il calcolo di $c(A', B')$ secondo la formula scritta sopra.

A titolo di esempio calcoliamo il costo della partizione $< A', B' >$ ottenuta da $< A, B >$ scambiando tra loro i nodi 10 e 7. Ossia $A' = (A \setminus \{10\}) \cup \{7\} = \{1, 7, 8, 9, 11, 12\}$ (e di conseguenza $B' = (B \setminus \{7\}) \cup \{10\} = \{2, 3, 4, 5, 6, 10\}$). Quindi risulta $c(A', B') = c(AB) - E(10) + I(10) - E(7) + I(7) + 2d_{10,7}$.

Poiché $E(10) = d_{10,2} + d_{10,6} + d_{10,7} = 6 + 6 + 1 = 13$; $I(10) = d_{10,11} = 2$; $E(7) = d_{7,8} + d_{7,10} = 3 + 1 = 4$; $I(7) = d_{7,5} + d_{7,6} = 8 + 4 = 12$; e $d_{10,7} = d_{7,10} = 1$, il costo della nuova partizione è $c(A', B') = 40$.

Per comprendere anche i costi della partizioni che appartengono all'intorno $\mathcal{I}(A,B)$ di $< A, B >$ si può ulteriormente arricchire la matrice $D(A,B)$ scrivendo in ogni elemento il valore $d_{h,k}$ pari al peso dell'arco (h,k) seguito (separato da "/') dal costo della partizione che si ottiene scambiando h con k:

$D(A,B) =$

		$I(\cdot)$	8	18	23	25	4	12
		$E(\cdot)$	21	5	0	0	11	4
$I(\cdot)$	$E(\cdot)$		**2**	**3**	**4**	**5**	**6**	**7**
9	5	**1**	5/42	0/58	0/68	0/70	0/38	0/29
0	3	**8**	0/25	0/51	0/61	0/63	0/31	30/52
0	5	**9**	0/23	0/49	0/59	0/61	5/39	0/44
2	13	**10**	6/29	0/43	0/53	0/55	6/35	1/40
2	6	**11**	6/36	0/50	0/60	0/62	0/30	0/45
0	9	**12**	4/25	5/55	0/55	0/57	0/25	0/40

Valutato il costo di ognuna delle $n^2 = 36$ partizioni che compongono l'intorno $\mathcal{I}(A,B)$ di $< A, B >$. occorre scegliere quella di costo minimo e procedere secondo l'algoritmo. Nel nostro esempio la migliore partizione dell'intorno $\mathcal{I}(A,B)$ di $< A, B >$ è quella che si ottiene scambiando 2 con 9, ossia la partizione $< A', B' >=< \{9,3,4,5,6,7\}, \{1,8,2,10,11,12\} >$ di costo $c(A',B') = c(A,B)$ $-E(2)$ $+I(2)$ $-E(9)$ $+I(9)$ $+2d_{2,9} = 41$ -21 $+8$ -5 $+0$ $+2 * 0 = 23$.

Non appena fissata la nuova partizione da cui ripartire con una nuova iterazione di Ricerca Locale, conviene aggiornare la matrice descrittiva della partizione, e i costi interni ed esterni per poter poi calcolare i costi delle nuove partizioni. Anche l'aggiornamento dei costi interni ed esterni può essere fatto in *modo incrementale*. Ricordando che $< A', B' >$ è stata ottenuta da $< A, B >$ scambiando un nodo $x \in A$ con un nodo $y \in B$.

Ricordando che $< A', B' >$ è stata ottenuta da $< A, B >$ scambiando un nodo $x \in A$ con un nodo $y \in B$, abbiamo che: x, che prima apparteneva ad A, adesso appartiene a B'; y, che prima apparteneva a B, adesso appartiene a A', un qualsiasi nodo $t \neq x$ che prima apparteneva ad A, adesso appartiene a A'; e un qualsiasi nodo $w \neq y$ che prima apparteneva ad B, adesso appartiene a B'.

Consideriamo un nodo $t \in A'$, $t \neq y$. Se t è adiacente a x e a y il contributo $d_{t,x}$ dell'arco (t,x) si sposta dal suo (vecchio) costo interno $I(t)$ al suo (nuovo) costo esterno $E'(t)$, e viceversa, il contributo $d_{t,y}$ dell'arco (t,y) si sposta dal suo (vecchio) costo esterno $E(t)$ al suo (nuovo) costo interno $I'(t)$; in definitiva

$$\text{nuovo costo esterno di un nodo } t \in A', t \neq y: \quad E'(t) = E(t) - d_{t,y} + d_{t,x}$$

$$\text{nuovo costo interno di un nodo } t \in A', t \neq y: \quad I'(t) = I(t) - d_{t,x} + d_{t,y}$$

Consideriamo ora un nodo $w \in B'$, $w \neq x$. Ragionando come abbiamo fatto per il nodo $t \in A'$, $t \neq y$, otteniamo che

$$\text{nuovo costo esterno di un nodo } w \in B', w \neq x: \quad E'(w) = E(w) - d_{w,x} + d_{w,y}$$

$$\text{nuovo costo interno di un nodo } w \in B', w \neq x: \quad I'(w) = I(w) - d_{w,y} + d_{wx}$$

Per nodi non adiacenti si deve considerare nullo il peso dell'arco corrispondente.

Lasciamo al lettore il compito di calcolare tutti i valori della matrice $D(A',B')$ descrittiva della partizione $< A', B' >=< \{9,\ 3,\ 4,\ 5,\ 6,\ 7\}, \{1,\ 8,\ 2,\ 10,\ 11,\ 12\} >$ disegnata di seguito. Si inseriscano, nell'ordine: i pesi degli archi (sono gli stessi di $D(A,B)$ e nella stessa posizione, tranne quelli relativi alla riga e alla colonna dei nodi appena scambiati); si calcolino i nuovi costi interni ed esterni di ogni nodo utilizzando le formule appena viste; si calcolino i costi delle partizioni che si otterrebbero scambiando tra di loro in tutti i modi possibili coppie di nodi appartenenti a classi diverse della partizione.

$D(A', B') =$

		$I(\cdot)$						
		$E(\cdot)$						
$I(\cdot)$	$E(\cdot)$		**9**	**3**	**4**	**5**	**6**	**7**
		1	/	/	/	/	/	/
		8	/	/	/	/	/	/
		2	/	/	/	/	/	/
		10	/	/	/	/	/	/
		11	/	/	/	/	/	/
		12	/	/	/	/	/	/

Fatto ciò, avremo valutato il costo di tutte le partizioni dell'intorno $\mathcal{I}(A', B')$ di $< A', B' >$. In accordo alle istruzioni dell'algoritmo, possiamo cercare, tra queste, la migliore partizione, confrontare il suo costo con $c(A', B')$ ed eventualmente proseguire, etc. etc. .

Un ulteriore vantaggio dal punto di vista della complessità dei calcoli si ha andando a valutare solo la variazione $\Delta_C(A', B', A, B)$ del costo di ogni partizione $< A', B' >$ dell'intorno di una partizione $< A, B >$ rispetto al costo di $< A, B >$ stessa. Evidentemente

$$\Delta_C(A', B', A, B) = c(A', B') - c(A, B) = -E(x) + I(x) - E(y) + I(y) + 2d_{x,y}.$$

Questa quantità può essere positiva, negativa, o nulla (e la partizione $< A', B' >$ sarà rispettivamente peggiore, migliore o uguale alla partizione$< A, B >$). La migliore partizione $< A', B' > \in \mathcal{I}(A, B)$ è una a cui corrisponde un $\Delta_C(A', B', A, B)$ minimo, e $< A', B' >$ è migliore di $< A, B >$ solo se $\Delta(A', B', A, B) < 0$.

La complessità dei calcoli però si può ancora ridurre se si calcola (una sola volta), e si memorizza, il valore $\Delta(x) = -E(x) + I(x)$ per ogni nodo x del grafo perché allora risulta

$$\Delta_C(A', B', A, B) = c(A', B') - c(A, B) = \Delta(x) + \Delta(y) + 2d_{x,y}.$$

Infine, osservando che la quantità $E(x)+I(x)$ è costante (per definizione), e pari a $cost(x)$, possiamo rendere più veloce anche l'operazione di aggiornamento dei valori $E(\cdot)$ e $I(\cdot)$. Infatti, una volta calcolato $E(x)$, la differenza $cost(x) - E(x)$ ci fornisce $I(x)$.

Per esercizio, completare quanto segue nell'ordine $cost(\cdot), E(\cdot), I(\cdot), \Delta(\cdot), \Delta_C(\cdot)$,.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
$cost =$												

$D(A', B') =$

			$\Delta(\cdot)$						
			$I(\cdot)$						
			$E(\cdot)$						
$\Delta(\cdot)$	$I(\cdot)$	$E(\cdot)$		**9**	**3**	**4**	**5**	**6**	**7**
			1	/	/	/	/	/	/
			8	/	/	/	/	/	/
			2	/	/	/	/	/	/
			10	/	/	/	/	/	/
			11	/	/	/	/	/	/
			12	/	/	/	/	/	/

Esempio svolto in aula: Knapsack. Per definire l'intorno di una soluzione ammissibile di Knapsack si potrebbero ammettere le rimozioni, ma le soluzioni che si ottengono sono sì tutte ammissisbili ma hanno una utilità non maggiore della soluzione di partenza, quindi di sicuro non sono quelle che permettono all'algoritmo di procedere. Rimangono allora le operazioni di inserimento e di scambio, entrambe possibili. Di ogni soluzione così ottenuta va verificata l'ammissibilità (ossia occorre controllare che sia rispettato il vincolo). Se la soluzione è ammissibile, allora farà parte dell'intorno, e va calcolata la corrispondente funzione obiettivo, altrimenti non fa parte dell'intorno e non ha senso calcolare il corrispondente valore della funzione obiettivo.