METODI 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.).
In generale, quindi, un metodo euristico come quello *greedy* non condurrà all'ottimo del problema, ma ad una soluzione "ottima" secondo l'euristica che si è scelti di seguire. Proprio perché determinano una soluzione ammissibile, gli algoritmi *greedy* possono essere utilizzati per determinare un *bound* 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.
Le idee base di un algoritmo *greedy* sono:

1) Incrementale: la soluzione viene costruita per gradi ("*from scratch*"), 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 se esso farà parte della soluzione (e con che "valore"), oppure no;
2) No *backtracking*: la decisione che l'elemento faccia parte (e con che valore) della soluzione o meno è definitiva, cioè non viene rimessa in discussione durante l'intero corso dell'algoritmo (ossia, non si elimina mai un elemento inserito nella soluzione e non si inserisce mai un elemento già scartato)
3) *Greedy selection*: la selezione avviene con un meccanismo "goloso" ("*greedy*" significa proprio "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 e2) se esso farà parte della soluzione (e con che valore o meno).

In altre parole, l'algoritmo conosce inizialmente l'insieme A degli elementi dai quali dovrà estrarre un sottoinsieme che rappresenta la soluzione. Istante per istante esso mantiene una tri-partizione $<X,Y,W>$ di A, dove X è composto di quegli elementi che sono stati esaminati e che si è deciso che faranno parte della soluzione (e dei quali si conosce il valore all'interno della soluzione finale), Y è il sottoinsieme di quegli elementi che sono stati esaminati e che si è deciso che non faranno parte finale, e W è l'insieme degli elementi in attesa di valutazione. Si noti che all'inizio $W=A$ e X e Y soon vuoti, e che alla fine si avrà che X rappresenta la soluzione, $Y=A\backslash X$, e W è vuoto.

Si osservi che per uno stesso problema si possono definire diversi criteri *greedy*, ottenendo diversi algoritmi *greedy*.

Esaminiamo il funzionamento di tale metodo attraverso due esempi:

a) 2 algoritmi *greedy* (con criteri diversi) per un problema di *knapsack* binario;
b) 2 algoritmi *greedy* (con criteri diversi) per un problema di *set covering*.

Due algoritmi greedy per un problema di knapsack binario

Sia dato il seguente problema:

$$\begin{aligned} \max\ & 15x_1 + 13x_2 + 14x_3 + 11x_4 + 12x_5 \\ \text{s.t.}\ & 5x_1 + x_2 + 2x_3 + 0.5x_4 + 0.6x_5 \leq 6.4 \\ & x \in B^5 \end{aligned}$$

Supponiamo di adottare il seguente criterio *greedy* : "scegliere l'elemento x_j tale che $c_j=max\{c_h\}$ tra tutti gli elementi rimasti. Una volta scelto quale elemento valutare rimane da capire, sulla base del vincolo del problema, se esso andrà inserito nella soluzione o meno: se il vincolo è verificato ponendo $x_j = 1$, allora l'elemento x_j viene inserito nella soluzione, altrimenti si pone $x_j = 0$ e non lo si inserisce nella soluzione.

Passo1) Inizialmente $W=\{a_1,a_2,a_3,a_4,a_5\}$, X e Y sono vuoti Risulta: $c_1 = \max_{h \in W}\{c_h\}$. L'oggetto 1 potrà essere inserito nello zaino se è rimasto spazio a sufficienza. In questo caso sì, poiché dal vincolo risulta 5 (volume oggetto 1) ≤ 6.4 (spazio residuo, che essendo all'inizio è tutto quello dato). Quindi, possiamo porre $x_1 = 1$ (che, contemporaneamente equivale a dire che l'elemento 1 farà parte della soluzione e che il suo valore nella soluzione è 1). Se inseriamo il vincolo $x_1 = 1$ nella formulazione e lo "applichiamo", successivamente lo possiamo eliminare, insieme alla variabile x_1 che non è più usata ottenendo il seguente problema semplificato:

$$\begin{aligned} \max\ & 13x_2 + 14x_3 + 11x_4 + 12x_5 + 15 \\ \text{s.t.}\ & x_2 + 2x_3 + 0.5x_4 + 0.6x_5 \le 1.4 \\ & x \in B^4 \end{aligned}$$

Ad ogni passo dell'algoritmo conserveremo l'informazione relativa alla soluzione parziale trovata; nel nostro caso: $W=\{a_2,a_3,a_4,a_5\}$, $X=\{a_1$ (con valore 1)$\}$, e Y è vuoto.

Passo 2) Risulta: $c_3 = \max_{h \in W}\{c_h\}$. Tuttavia, non è possibile inserire l'oggetto 3 nello zaino, non essendo sufficiente lo spazio residuo (infatti: 2 > 1.4!). Quindi $x_3 = 0$, e la formulazione risulta così semplificata:

$$\begin{aligned} \max\ & 13x_2 + 11x_4 + 12x_5 + 15 \\ \text{s.t.}\ & x_2 + 0.5x_4 + 0.6x_5 \le 1.4 \\ & x \in B^3 \end{aligned}$$

Risulta $W=\{a_2,a_4,a_5\}$, $X=\{a_1$ (con valore 1)$\}$, e $Y=\{a_3\}$.

Passo 3) Risulta: $c_2 = \max_{h \in W}\{c_h\}$. Possiamo inserire l'oggetto 2 nello zaino perché il volume da esso occupato risulta inferiore allo spazio residuo nello zaino (1≤1.4). Perciò, inseriamo x_2 nella soluzione, con valore 1, e semplifichiamo il problema:

$$\begin{aligned} \max\ & 11x_4 + 12x_5 + 28 \\ \text{s.t.}\ & 0.5x_4 + 0.6x_5 \le 0.4 \\ & x \in B^2 \end{aligned}$$

Risulta $W=\{a_4,a_5\}$, $X=\{a_1$ (con valore 1), a_2 (con valore 1)$\}$, e $Y=\{a_3\}$.

Passo 4) Risulta: $c_5 = \max_{h \in W}\{c_h\}$. Tuttavia, 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:

$$\begin{aligned} \max\ & 11x_4 + 28 \\ \text{s.t.}\ & 0.5x_4 \le 0.4 \\ & x \in B \end{aligned}$$

Risulta $W=\{a_4\}$, $X=\{a_1$ (con valore 1), a_2 (con valore 1)$\}$, e $Y=\{a_3,a_5\}$.

Passo 5) Chiaramente , siccome W contiene un solo elemento, questo sarà l'elemento scelto per la valutazione. Dato che il suo volume non permette di verificare il vincolo, infatti $0.5 > 0.4$, non inseriremo l'oggetto 4 nello zaino:

$$\begin{aligned} \max\ & 28 \\ \text{s.t.}\ & 0 \le 0.4 \end{aligned}$$

Risulta Wvuoto, $Y=\{a_3, a_4,a_5\}$, e la soluzione trovata $X=\{a_1$ (con valore 1), a_2 (con valore 1)$\}$ ha valore 28.

Supponiamo, ora, di scegliere un altro criterio *greedy*, e precisamente: "scegliere un elemento $x_j \in W$ che massimizzi il rapporto utilità/peso ($\frac{c_j}{a_j} = \max_{h \in W}\left\{\frac{c_h}{a_h}\right\}$) tra tutti gli elementi rimasti, ossia quelli in W, dopodiché verificare il vincolo: se il vincolo risulta verificato, inserire l'elemento x_j nella soluzione ($x_j = 1$), altrimenti non inserirlo ($x_j = 0$) e applicare nuovamente il criterio *greedy*".
I rapporti utilità/peso risultano: x_1: 15/5=3; x_2: 13/1=13; x_3: 14/2=7; x_4: 11/0.5=22; x_5: 12/0.6=20.
Quindi è già stabilito l'ordine in cui verrano valutati gli oggetti, passso per passo, e precisamente: l'oggetto 4, il 5, il 2, il 3 e l'1.

Passo 1) Scelgo da W l'oggetto 4 perchè $\frac{c_4}{a_4} = \max_{h \in W}\left\{\frac{c_h}{a_h}\right\}$. Poiché il vincolo è verificato ($0.5 \le 6.4$), introduco x_4=1 nella formulazione e semplifico:

$$\begin{aligned} \max\ & 15x_1 + 13x_2 + 14x_3 + 12x_5 + 11 \\ \text{s.t.}\ & 5x_1 + x_2 + 2x_3 + 0.6x_5 \le 5.9 \\ & x \in B^4 \end{aligned}$$

Risulta $W=\{a_1,a_2,a_3,a_5\}$, $X=\{a_4$ (con valore 1)$\}$, e Y vuoto.

Passo 2) Scelgo l'oggetto 5 perché, tra quelli non ancora valutati è l'unico che massimizza il criterio fissato, infatti $\frac{c_5}{a_5} = \max_{h \in W}\left\{\frac{c_h}{a_h}\right\}$. Poiché il vincolo è verificato ($0.6 \le 5.9$), introduco x_5=1 nella formulazione e semplifico:

$$\begin{aligned} \max\ & 15x_1 + 13x_2 + 14x_3 + 27 \\ \text{s.t.}\ & 5x_1 + x_2 + 2x_3 \le 5.3 \\ & x \in B^3 \end{aligned}$$

Risulta $W=\{a_1,a_2,a_3\}$, $X=\{a_4$ (con valore 1), a_5. (con valore 1)$\}$, e Y vuoto.

Passo 3) Scelgo l'oggetto 2 perché, tra quelli non ancora valutati è l'unico che massimizza il criterio fissato, infatti $\frac{c_2}{a_2} = \max_{h \in W}\left\{\frac{c_h}{a_h}\right\}$. Poiché il vincolo è verificato ($1 \leq 5.9$), introduco x_2=1 nella formulazione e semplifico:

$$\begin{aligned} \max\ & 15x_1 + 14x_3 + 40 \\ \text{s.t.}\ & 5x_1 + 2x_3 \leq 4.3 \\ & x \in B^2 \end{aligned}$$

Risulta W=$\{a_1,a_3\}$, X=$\{$ a_2 (con valore 1), a_4. (con valore 1), a_5 (con valore 1)$\}$, e Y vuoto.

Passo 4) Scelgo l'oggetto 3 perché $\frac{c_3}{a_3} = \max_{h \in W}\left\{\frac{c_h}{a_h}\right\}$. Poiché il vincolo è verificato ($2 \leq 4.3$), introduco x_3=1 nella formulazione e semplifico:

$$\begin{aligned} \max\ & 15x_1 + 54 \\ \text{s.t.}\ & 5x_1 \leq 2.3 \\ & x \in B \end{aligned}$$

Risulta W=$\{a_1\}$, X=$\{$ a_2 (con valore 1), a_3 (con valore 1), a_4 (con valore 1), a_5 (con valore 1)$\}$, e Y vuoto.

Passo 5) Scelgo l'oggetto 1 perché $\frac{c_1}{a_1} = \max_{h \in W}\left\{\frac{c_h}{a_h}\right\}$. Poiché il vincolo non è verificato ($5 > 2.3$), introduco x_1=0 nella formulazione e semplifico:

$$\begin{aligned} \max\ & 54 \\ \text{s.t.}\ & 0 \leq 2.3 \end{aligned}$$

Risulta Wvuoto, Y=$\{a_1\}$, e la soluzione trovata X=$\{$ a_2 (con valore 1), a_3 (con valore 1), a_4 (con valore 1), a_5 (con valore 1)$\}$ ha valore 54.

Nota: Con questo secondo criterio è stato trovato un ottimo migliore. Infatti, abbiamo 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=6.3 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.

Due algoritmi greedy per un problema di set covering

Valuteremo un esempio numerico, seguendo i seguenti passi:

a) formulazione del problema;
b) applicazione dell'euristica di tipo *greedy* al problema ridotto.

a) $\quad \text{Min}\ \ 3x_1 + 5x_2 + 6x_3 + 2x_4 + x_5 + 7x_6 + x_7 + 8x_8$

1		1					1	≥ 1
	1				1			≥ 1
			1					≥ 1
1		1			1			≥ 1
				1		1	1	≥ 1
1			1		1			≥ 1

$x_i \in \{0,1\} \quad i=1, 2, \ldots, n$

Ogni colonna della matrice dei coefficienti rappresenta un elemento, mentre ogni vincolo rappresenta un sottoinsieme di $A = \{a_1, a_2, a_3, \ldots, a_8\}$. L'insieme di tali sottoinsiemi può essere così rappresentato: $\Omega = \{\{a_1, a_3, a_8\}, \{a_2, a_6\}, \{a_4\}, ...\}$. L'obiettivo sarà quello di selezionare un sottoinsieme $S = \{a_{j1}, a_{j2}, \ldots\} \subseteq A$ in modo tale che venga selezionato almeno un elemento da ogni sottoinsieme $A_i \in \Omega$ e che il costo $c(S) = \sum_{a_i \in S} c_i$ di tale sottoinsieme sia minimo.

b) Possono essere sfruttate due euristiche di tipo *greedy* diverse e precisamente basate sui due seguenti criteri *greedy*:

- Tra le variabili che permettono di verificare un vincolo non ancora soddisfatto, ne scelgo una di costo minimo;
- Sia x_j una variabile tale che: $\frac{c_j}{d_j} = \min_k \left\{ \frac{c_k}{d_k},\ x_k \text{ free} \right\}$, dove d_h è il numero di vincoli non soddisfatti in cui la variabile x_h è presente. Allora $x_j = 1$.

Applicando il primo criterio si conosce a priori l'ordine con cui verranno valutati gli elementi, che risulta: $a_5, a_7, a_8, a_4, a_1, a_2, a_3, a_6$. 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, e poi tra a_4 e a_8: in questi casi avremmo potuto scegliere indifferentemente l'ordine a_5,a_7 oppure a_7,a_5, e/o a_4,a_8 piuttosto che a_8,a_4). Ogni elemento viene valutato: se permette di verificare vincoli non ancora verficati, ossia se permette di coprire sottoinsiemi non ancora coperti, viene inserito nella soluzione, altrimenti, se così non avviene, è inutile inserirlo nella soluzione perchè non aumenterebbe il numero dei sottoinsiemi coperti (ossia dei vincoli verificati), mentre aumenterebbe il costo della funzione obiettivo, contrariamente al nostro obiettivo di mantenerla più piccola possibile. Ad esempio:

- Scegliamo a_5 da W, e siccome la scelta di questo elemento permette di verficare il quinto vincolo lo inseriamo nella soluzione con valore $x_5 = 1$;
  risulta $X=\{ a_5$ (con valore 1)$\}$, Y vuoto, e $W=\{a_1,a_2,a_3,a_4,a_6,a_7,a_8\}$.
- Ora scegliamo a_7 da W, essendo rimasto l'unico a minimizzare il criterio scelto. Poiché x_7 è presente solo in vincoli già soddisfatti (infatti solo nel quinto), possiamo non inserirla nella soluzione: $x_7 = 0$;
  e risulta $X=\{ a_5$ (con valore 1)$\}$, $Y=\{ a_7\}$, e $W=\{a_1,a_2,a_3,a_4,a_6,a_8\}$
- …….

Proseguendo in questo modo, il primo algoritmo *greedy* fornisce la seguente soluzione:

risulta $X=\{a_1,a_2,a_4,a_5,a_8$ (tutti con valore 1)$\}$, $Y=\{a_3,a_6,a_7\}$, e Wvuoto, con valore 13 della funzione obiettivo.

Applichiamo, ora, il secondo criterio *greedy*. In questro 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, in funzione dei vincoli rimasti non verificati e che tale elemento permetterebbe di verificare (nel criterio sono i denominatori). I precedenti tre criteri possono essere definiti *statici*, mentre l'ulitmo può definirsi *dinamico*.

Passo 1) Inizialmente X e Y sono vuoti, $W=\{a_1,a_2,a_3,a_4,a_5,a_6,a_7,a_8\}$, e i rapporti risultano, nell'ordine, 3/3=1, 5/1=5, 6/2=3, 2/1=2, 1/1=1, 7/3=2+ε, 1/1=1, e 2/2=1. Viene scelto a_1: siccome permette di verficare i vincoli I, IV, e VI, non ancora verificati, tael elemento viene inserito nella soluzione. Quindi risulta $X=\{a_1$ (con valore 1)$\}$, Y vuoto, $W=\{a_2,a_3,a_4,a_5,a_6,a_7,a_8\}$, e valore della funzione obiettivo pari a 3.

Passo 2) Nell'ordine con cui gli elementi compaiono nell'insieme W alla fine del passo precedente, i rapporti risultano 5/1=5, 6/0=+∞, 2/1=2, 1/1=1, 7/1=7, 1/1=1, e 2/1=2 (Si noti come sono variati rispetti al passo precedente). Viene scelto a_5: siccome permette di verficare il V vincolo che ancora non era verificato, tale elemento viene inserito nella soluzione. Quindi risulta $X=\{a_1,a_5$ (entrambi con valore 1)$\}$, Y vuoto, $W=\{a_2,a_3,a_4,a_6,a_7,a_8\}$, e valore della funzione obiettivo pari a 4.

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 +∞ alla corridpondente 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 anche a rimanere fuori dalla soluzione. Tuttavia, dato che l'esito di tale condizione è noto fin dal momento in cui ciò avviene, possiamo senz'altro inserire nell'insieme Y gli elementi che hanno un denominatore pari a zero, non appena ciò si verifica. Nel passo 2) questo può essere fatto per l'elemento a_3. Pertanto il passo 2 si può conludere con i seguenti insiemi: $X=\{a_1,a_5$ (entrambi con valore 1)$\}$, $Y=\{a_3\}$, $W=\{a_2,a_4, a_6,a_7,a_8\}$, e valore della funzione obiettivo pari a 4.

Passo 3) Nell'ordine con cui gli elementi compaiono nell'insieme W alla fine del passo precedente, i rapporti risultano 5/1=5, 2/1=2, 7/1=7, 1/0=+∞, e 2/0=+∞. Viene scelto a_4: siccome permette di verficare il III vincolo che ancora non era verificato, tale elemento viene inserito nella soluzione. Per quanto osservato precedentemente a riguardo degli elementi che presentano un denominatore pari a 0, risulta $X=\{a_1,a_4,a_5$ (tutti con valore 1)$\}$, $Y=\{a_3,a_7,a_8\}$, $W=\{a_2,a_6\}$, e valore della funzione obiettivo pari a 6.

Passo 4) Nell'ordine con cui gli elementi compaiono nell'insieme W alla fine del passo precedente, i rapporti risultano 5/1=5 e 7/1=7. Viene scelto a_2: siccome permette di verficare il II vincolo che ancora non era verificato, tale elemento viene inserito nella soluzione. Risulta $X=\{a_1,a_2,a_4,a_5$ (tutti con valore 1)$\}$, $Y=\{a_3,a_7,a_8\}$, $W=\{a_6\}$, e valore della funzione obiettivo pari a 11.

Passo 5) L'unico elemento attualmente in W ha rapporto pari a 7/0=+∞. Quindi l'algoritmo si conclude con $X=\{a_1,a_2,a_4,a_5$ (tutti con valore 1)$\}$, $Y=\{a_3,a_6,a_7,a_8\}$, e W vuoto, e valore della funzione obiettivo pari a 11.

Oss.: Si noti che, a differenza di quanto succedeva nella risoluzione del problema di *knapsack* con un'euristica di tipo *greedy*, nel caso del Set Covering l'inserimento di un elemento nella formulazione dipende solo dalla convenienza per la funzione obiettivo in quanto comunque non violerà mai i vincoli (se permette di verificare 1 o più vincoli, sarà conveniente inserirlo nella soluzione, altrimenti non ha senso sostenere il suo costo, e quindi non verrà inserito nella soluzione)

LA RICERCA LOCALE (*Local search*):

Esaminiamo, ora, delle tecniche euristiche basate sulla ricerca locale, anche dette euristiche di scambio (*interchange heuristics*).
Il procedimento che viene seguito è il seguente:

- Sia S una soluzione iniziale;
- Si calcoli l'intorno $\Im(S)$ di S;
- Si determini la migliore soluzione di $\Im(S)$, sia S';
- Se S' è migliore di S, allora $S := S'$ e si torni all'inizio, altrimenti S è la migliore soluzione determinata, stop.

Commenti:

1) 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, con una ricerca per ispezione e così via…); tuttavia, in genere, vengono scelte delle euristiche di tipo *greedy*.
2) La definizione di intorno $\Im(S)$ di S dipende dal problema.
   Esempio: Sia dato un problema P tale che ogni soluzione ammissibile è rappresentata da una bipartizione di un insieme di elementi A, così definito: $A = \{a_1, a_2, a_3, a_4, a_5\}$. Sia data una soluzione iniziale $<S, \overline{S}>$ così definita: $S = \{a_1, a_3\}$, $\overline{S} = \{a_2, a_4, a_5\}$.
   Un intorno $\Im(S)$ di tale soluzione può, ad esempio, essere definito come l'insieme di tutte le soluzioni ottenibili da $<S, \overline{S}>$ attraverso lo scambio di un elemento di S con un elemento di $\overline{S}$.
   Allora, $\Im_1(S) = \{<(a_2, a_3), (a_1, a_4, a_5)>, <(a_4, a_3),(a_1, a_2, a_5)>, <(a_5, a_3), (a_1, a_2, a_4)>, <(a_1, a_2), (a_3, a_4,a_5)>, <(a_1, a_4), (a_2, a_3,a_5)>, <(a_1, a_5), (a_2, a_3, a_4)>\}$

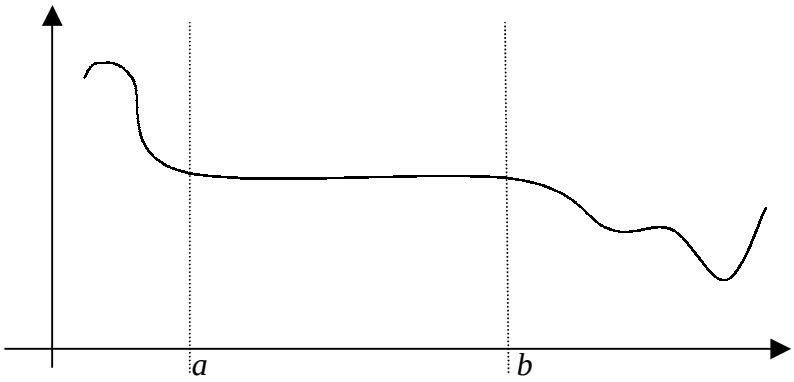
   Abbiamo detto che i miglioramenti locali sono effettuati scambiando "pochi" elementi della soluzione con "pochi" elementi fuori dalla soluzione (in ugual numero o no, a seconda del problema). Che significa, però, il termine "pochi"? Per quantificare tale numero, bisogna tener presenti le seguenti osservazioni:

   - Il numero di elementi da spostare influenza la dimensione di $\Im(S)$;
   - Tanto più è piccolo è $\Im(S)$, tante più iterazioni verranno effettuate prima di determinare l'ottimo.

3) In alcuni casi si comporta meglio una versione dell'algoritmo in cui l'ultimo passo è così definito:

   - Se *S'* è non peggiore di *S* allora *S* := *S'* e si torni all'inizio, altrimenti *S* è la migliore soluzione determinata, stop.

   In particolare, questa versione funziona migliore se la funzione obiettivo del problema presenta delle regioni di dimensioni grandi rispetto alle dimensioni di $\Im(S)$ in cui essa ha valore costante.
   Ad esempio, si consideri il caso seguente:



   La regione (*a*, *b*) è a valore costante.
   Questa definizione, in effetti, aiuta a "superare" queste regioni e ricominciare a determinare soluzioni migliori successivamente.

4) Tuttavia, si corre il rischio di terminare in un **ottimo locale**. Per attenuare tale rischio, si può applicare l'algoritmo a partire da diverse soluzioni iniziali, e scegliere il migliore ottimo locale che viene trovato; ovviamente, maggiore è il numero di soluzioni iniziali da cui si parte, minore sarà il rischio di ottenere un ottimo locale che non sia anche globale.

Supponiamo di applicare un algoritmo di Ricerca Locale in cui la transizione da *S* a *S'* viene accettata solo se presenta un miglioramento "maggiore di una certa soglia δ fissata a priori" ossia se il passo " Se *S'* è migliore di *S*, allora *S* := *S'* " viene sostituito dal seguente

Se la soluzione *S'* è migliore di una quantità δ rispetto alla soluzione *S*, allora *S* := *S'*.

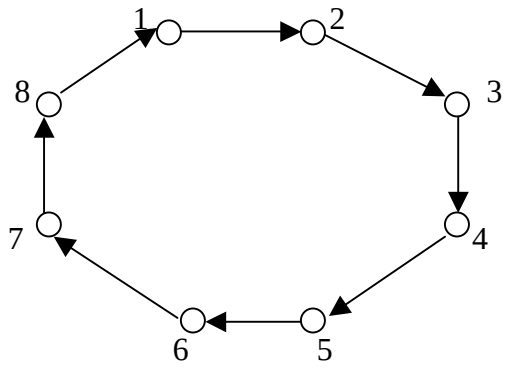
In questo caso, la soluzione che si ottiene dall'applicazione dell'algoritmo potrebbe anche non essere un ottimo locale (funzione della definizione di intorno utlizzata). Infatti, può succedere che nell'intorno della soluzione finale S_F vi siano delle soluzioni migliori ma non sufficientemente da

essere accettate (in altre parole il loro costo c verifica: $c(S_F)-\delta \leq c < c(S_F)$, per un problema di min, $c(S_F) < c \leq c(S_F)+\delta$, per un problema di max, dove $c(S_F)$ rappresente il costo della soluzione S_F).

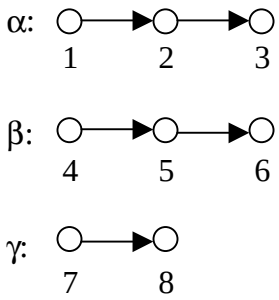
Esempio: *Ricerca locale applicata al problema del TSP*

In questo caso, viene usata l'euristica detta *3-arc interchange*, che funziona secondo i seguenti passi:

- Sia dato un grafo G ed un qualsiasi ciclo hamiltoniano C su G; seleziona tre archi di C in tutti i modi possibili. Ad esempio, sia dato il seguente ciclo hamiltoniano (soluzione iniziale):

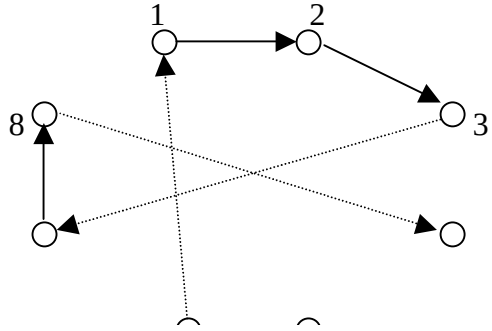


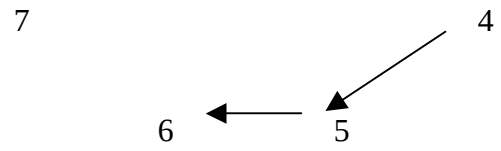
Supponiamo di selezionare gli archi (3,4),(6,7),(8,1); eliminando tali archi dal ciclo otteniamo tre cammini:



Tali cammini possono essere collegati tra loro solo in due modi:

- ✓ α,(3,4),β,(6,7),γ,(8,1) ➔ è proprio il ciclo dal quale siamo partiti;
- ✓ α,(3,7),β,(8,4),γ,(6,1) ➔ è il seguente ciclo hamiltoniano:





Dunque, in corrispondenza ad ogni tripla di archi selezionati dal ciclo *C* ottengo un *unico* ciclo diverso dal ciclo *C* stesso. Di conseguenza, la cardinalità dell'intorno del ciclo *C* è $O(n^3)$.

- Calcolare il ciclo "alternativo" in corrispondenza a ciascuna tripla (in questo modo si arriva a definire tutto $\Im(C)$);
- Scegliere il ciclo *C'* di costo minore;
- Se il costo di *C'* è minore del costo di *C*, allora ripetere a partire da *C'*; altrimenti *C* è ottimo locale rispetto alla scelta dei tre archi.

Oss.: Si osservi che la *3-arc interchange* è la più piccola euristica di ricerca locale per un problema di TSP. Infatti, se si decidesse, ad esempio, di scambiare solo due archi, l'intorno della soluzione iniziale *C* sarebbe vuoto.[1]

Def.: Data una soluzione *S*, si definisce **intorno** di *S*, $\Im(S)$, l'insieme di soluzioni ottenibili da *S* tramite operazioni di:

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

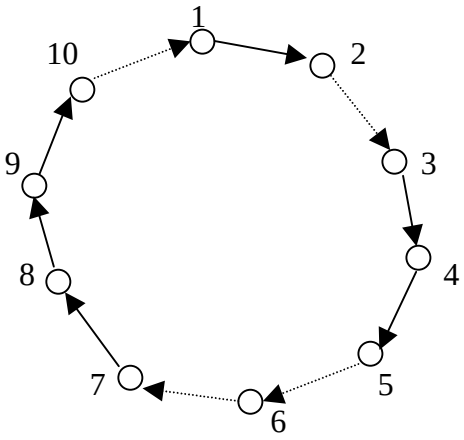
Se la cardinalità di una soluzione *S* è fissata a priori, allora $\Im(S)$ è definito solo tramite operazioni di scambio, in cui, in particolare, $|X| = |Y|$[2].

Come può essere calcolato il costo del nuovo ciclo *C'* a partire dal costo (noto) del ciclo di partenza *C*?
Basta osservare che: $c(C') = c(C)$ - costo degli archi tolti + costo degli archi inseriti. In totale, i termini da sommare saranno, dunque, sette.

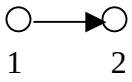
Esercizio 12: Dato un grafo *G* con $n = 10$ nodi, applicare un'euristica che scambi quattro archi del ciclo, mostrando che si ottengono sei cicli, uno dei quali è il ciclo iniziale.

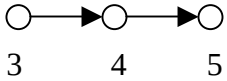
*Soluzione **es.12**:*
Un ciclo che passi per i dieci nodi del grafo *G* può essere rappresentato senza perdita di generalità nel modo seguente (eventualmente, infatti, possiamo rinumerare i nodi per fare in modo che il ciclo hamiltoniano coincida con quello in figura):

[1] Come vedremo in seguito, per un problema di S-TSP (TSP simmetrico) sarà sufficiente l'euristica *2-arc interchange*.
[2] Un esempio è l'euristica vista precedentemente per problemi di TSP detta *3-arc interchange*.

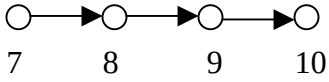


Supponiamo di eliminare dal ciclo i quattro archi tratteggiati, cioè la quadrupla di archi Q_1 = {(2,3),(5,6),(6,7),(10,1)}. In questo modo, possono essere evidenziati quattro cammini:

α:


β:


γ:


δ:


I cicli che si possono ottenere da questi cammini sono i seguenti[3]:

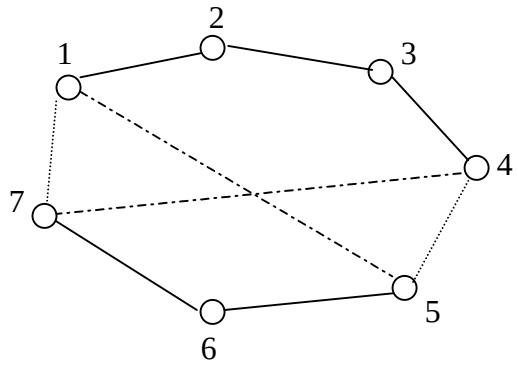
1) $\alpha_1\beta_1\gamma_1\delta_1$ (che coincide con il ciclo di partenza)
2) $\alpha_1\beta_1\delta_1\gamma_1$
3) $\alpha_1\gamma_1\beta_1\delta_1$
4) $\alpha_1\gamma_1\delta_1\beta_1$
5) $\alpha_1\delta_1\beta_1\gamma_1$
6) $\alpha_1\delta_1\gamma_1\beta_1$

Perciò, per ogni quadrupla selezionata, si ottengono sei cicli, di cui uno uguale a quello di partenza. Ora, le quadruple possono essere scelte in $O(n^4)$ modi diversi; poiché ogni quadrupla dà luogo a cinque nuovi cicli, in totale verranno esaminati $5 \cdot O(n^4) \cong O(n^4)$ cicli diversi.

[3] Da notare, ad esempio, che il ciclo rappresentato come $\gamma_1\beta_1\delta_1\alpha_1$ coincide con il ciclo rappresentato come $\alpha_1\gamma_1\beta_1\delta_1$ e verrà, quindi, contato una sola volta.

Esempio: *Symmetric Travelling Salesman Problem (S-TSP)*

Le istanze di S-TSP sono tutte caratterizzate dal fatto che $d_{ij} = d_{ji} \quad \forall\, i, j \in N$ (dove N è l'insieme dei nodi del grafo). Ciò equivale a dire che il grafo è non orientato.
In questo caso, come anticipato, è sufficiente un'euristica di scambio di due archi del ciclo hamiltoniano di partenza (*2-arc interchange*). Ad esempio, rimovendo gli archi (4,5) e (7,1) dal grafo in figura, si possono ottenere due cicli, uno dei quali è uguale al ciclo di partenza; di conseguenza, un'euristica di *2-arc interchange* assicura che verrà esaminato esattamente un ciclo diverso da quello di partenza.



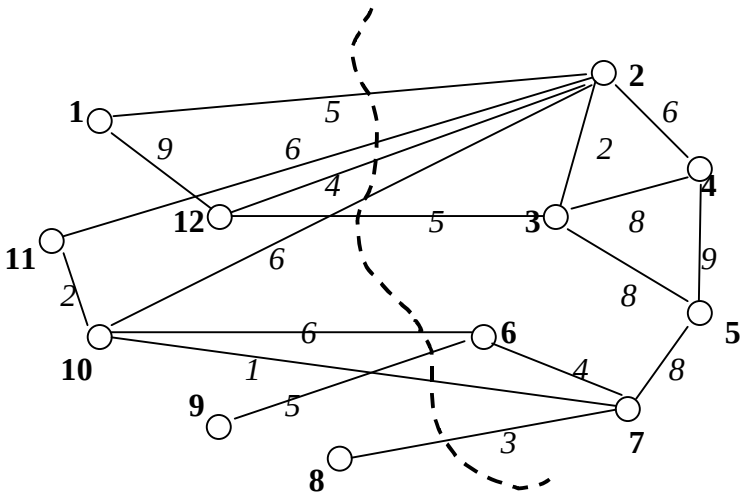
ciclo 1: 1 → 2 → 3 → 4 → 5 → 6 → 7 (ciclo di partenza)
ciclo 2: 1 → 2 → 3 → 4 → 7 → 6 → 5 (nuovo ciclo)

Esempio: *Uniform Graph Partitioning*

Dato: un grafo $G = (V, E)$ con pesi $d_{ij} \quad \forall\, (i, j) \in E$ e con $|V| = 2n$ nodi;
Trovare: una partizione $<A, B>$ dell'insieme V dei nodi ($V = A \cup B$ e $A \cap B = \varnothing$);
In modo tale che: $|A| = |B|$ e $c(A, B) = \sum_{\substack{i \in A \\ j \in B}} d_{ij}$ sia minimo.

Oss.: 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). Il vincolo $|A| = |B|$ complica enormemente le cose. Tuttavia, questa osservazione può essere sfruttata per dire che rimovendo il vincolo $|A| = |B|$ si ottiene un problema di max flow – min cut che, risolto all'ottimo, ci fornisce un LB per $c^*(A,B)$. Si osservi che quest'ultimo non è altro che un esempio di rilassamento di tipo combinatorio.

Supponiamo di avere il seguente grafo, descritto dalla seguente matrice di adiacenza nodi – nodi e di avere la partizione evidenziata in figura:

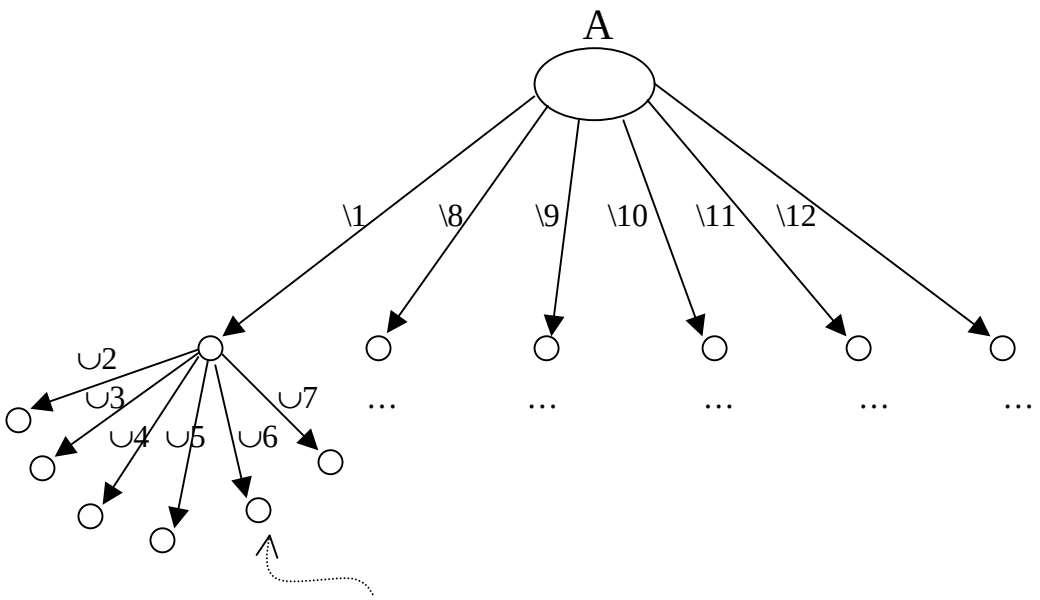


	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
1		*5*										*9*
2	*5*		*2*	*6*						*6*	*6*	*4*
3		*2*		*8*	*8*							*5*
4		*6*	*8*		*9*							
5			*8*				*8*					
6				*9*			*4*		*5*	*6*		
7					*8*	*4*		*3*		*1*		
8							*3*					
9						*5*						
10		*6*				*6*	*1*				*2*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

Si noti che, poiché il grafo è non orientato, la matrice risulta simmetrica.

La partizione evidenziata in figura è la seguente: $A = \{1, 8, 9, 10, 11, 12\}$[4].
Il costo di tale partizione è: $c(A, B) = 5 + 12 + 5 + 4 + 6 + 5 + 3 + 1 = 41$.
L'intorno di tale soluzione è definito nel modo che segue: $\Im(A, B) = \{<A', B'>$ tali che $A' = (A \setminus \{x\}) \cup \{y\},\ B' = (B \setminus \{y\}) \cup \{x\},$ con $x \in A$ e $y \in B\}$.
Poiché il grafo ha $2 \cdot n$ nodi, possiamo scegliere x in n modi diversi ed y in altrettanti modi diversi.
Per questo motivo, l'intorno di $<A, B>$ ha n^2 elementi, cioè: $|\Im(A, B)| = n^2$.
Graficamente, ciò può essere facilmente verificato:

[4] Si osservi che la partizione $<A, B>$ può essere sempre rappresentata semplicemente elencando i nodi facenti parte dell'insieme A, dato che $B = \overline{A} = \{v_i \in V : v_i \notin A\}$.



Ad esempio, questa partizione <*A'*, *B'*> appartenente all'intorno di <*A*, *B*> corrisponde a $A' = A \setminus \{1\} \cup \{6\}$.

Per valutare il costo di <*A'*, *B'*> bisogna procedere ex novo oppure si può sfruttare il fatto che è noto il costo di <*A*, *B*>? Naturalmente, è più conveniente utilizzare tutti i dati che abbiamo; per far ciò, possiamo sfruttare le nozioni di *costi interno* ed *esterno*.

Def.: Si definisce **costo interno *I(v)***, con $v \in X$, la seguente quantità:

$$I(v) = \sum_{i \in X} d_{vi}$$

In altre parole, il costo interno $I(v)$ di un nodo $v \in X$ è definito come la somma dei pesi associati agli archi da $v \in X$ verso nodi dello stesso insieme X.

Def.: Si definisce **costo esterno *E(v)***, con $v \in X$, la seguente quantità:

$$E(v) = \sum_{j \in \overline{X}} d_{vj}$$

In altre parole, il costo esterno $I(v)$ di un nodo $v \in X$ è definito come la somma dei pesi associati agli archi da $v \in X$ verso nodi dell'altro insieme $\overline{X}$.

Supponiamo di voler calcolare il costo della partizione $A' = A \setminus \{10\} \cup \{7\}$. Sfruttando i concetti di costo interno ed esterno, possiamo calcolare $c(A', B')$ a partire da $c(A, B)$ nel modo seguente:

$$c(A', B') = c(A, B) - E(10) + I(10) - E(7) + I(7) + 2d_{10,7}^{5}.$$

[5] Si noti che il peso dell'arco (10,7) va aggiunto due volte perché, sottraendo E(10) e E(7) dal costo della partizione <A,B>,anche $d_{10,7}$ viene sottratto due volte, cosa che non vogliamo che accada, perché l'arco in questione è tagliato anche nella partizione <A',B'>.

Poiché $E(10) = 13$; $I(10) = 2$; $E(7) = 4$; $I(7) = 12$; $d_{10,7} = 1$, il costo della nuova partizione è: $c(A', B') = 40$.
Poiché $c(A', B') < c(A, B)$, la partizione $<A', B'>$ viene accettata perché più conveniente della partizione $<A, B>$.
Si osservi che conviene aggiornare ad ogni passo costi interni ed esterni, per poterli sfruttare per il calcolo del costo della nuova partizione ottenuta. Ad esempio:

$E'(6) = E(6) + d_{6,7} - d_{10,6}$;
$I'(6) = I(6) - d_{6,7} + d_{10,6}$.

Infatti, mentre $10 \in A$ e $6, 7 \in B$, dopo lo scambio dei nodi 10 – 7 si ha che $7 \in A'$ e $10, 6 \in B'$.

Questo calcolo va ripetuto per tutti i nodi adiacenti a quelli che sono stati spostati da una partizione all'altra per mezzo dell'operazione di scambio effettuata.

Esercizio 13: Determinare due soluzioni appartenenti ad $\Im(A', B')$, e calcolarne il costo, scegliere la migliore soluzione tra le due e aggiornare costi interni ed esterni.

*Soluzione **es.13**:*
Innanzitutto, calcoliamo i costi interni ed esterni associati alla partizione iniziale $<A, B>$.

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

Come abbiamo già visto, il costo di questa partizione è $c(A, B) = 41$.
Consideriamo, ora, la partizione $<A', B'>$ ottenuta tramite lo scambio del nodo 7 con il nodo 10. Per aggiornare il valore dei costi interni ed esterni, andremo a considerare soltanto quei nodi che sono adiacenti al nodo 7 e/o al nodo 10. Risulta, allora:

v	$I'(v)$	$E'(v)$
1	9	5
2	14	15
3	18	5
4	23	0
5	17	8
6	6	9
7	3	13
8	3	0
9	0	5
10	12	3
11	0	8
12	9	9

Il costo di questa nuova partizione è $c(A', B') = c(A, B) – E(10) + I(10) – E(7) + I(7) + 2d_{10,7} = 41 – 13 + 2 – 4 + 12 + 2 \cdot 1 = 40$.
Essendo $c(A', B') < c(A, B)$, l'ottimo corrente deve essere aggiornato in modo tale che risulti che la soluzione ottima corrente sia la partizione $<A', B'>$.
Consideriamo, ora, la partizione $<A'', B''>$ ottenuta, ad esempio, tramite lo scambio del nodo 9 con il nodo 6. Aggiornando costi interni ed esterni, si ottiene:

v	$I''(v)$	$E''(v)$
1	9	5
2	14	15
3	18	5
4	23	0
5	17	8
6	4	11
7	7	9
8	3	0
9	0	5
10	6	9
11	0	8
12	9	9

Il costo di questa nuova partizione è $c(A'', B'') = c(A', B') – E'(9) + I'(9) – E'(6) + I'(6) + 2d_{9,6} = 40 – 5 + 0 – 9 + 6 + 2 \cdot 5 = 42$.
Essendo $c(A'', B'') > c(A', B')$, l'ottimo corrente non deve essere aggiornato.

TECNICHE PER MIGLIORARE LA RICERCA LOCALE:

Abbiamo visto che il problema principale della ricerca locale è il rischio di terminare in un ottimo locale. Cosa fare, allora, per "scappare" da un ottimo locale? Le soluzioni possibili sono due:

a) Applicare la ricerca locale ad un certo numero di soluzioni iniziali, come già precedentemente osservato, ed è una tecnica che permette di definire un algoritmo che consiste semplicemente nella ripetuta applicazione dell'algoritmo di Ricerca Locale, e nella valutazione della migliore tra le soluzioni finali ottenute a partire da diverse soluzioni iniziale;
b) Accettare transizioni verso soluzioni peggiori, in modo da sfuggire dall'ottimo locale raggiunto fino a quel punto.

Supponiamo di valutare un problema di minimizzazione: applicando la ricerca locale, partendo da una soluzione iniziale S_0, si ha una sequenza di soluzioni con costo non crescente:

$$S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow ... \rightarrow S_k$$

Quando risulta $c(S) > c(S_k) \ \forall S \in \Im(S_k)$, l'algoritmo termina e S_k è l'ottimo locale.
Accettare transizioni verso soluzioni peggiori significa scegliere $S' \in \Im(S)$ nel modo seguente: scegli $S' \in \Im(S)$ tale che $c(S') = \min\{c(\tilde{S}), \tilde{S} \in \Im(S)\}$. Ciò significa che, partendo da una soluzione S_{k-1}, è consentito accettare transizioni anche verso soluzioni S_k tali che $c(S_k) > c(S_{k-1})$. Ad esempio, consideriamo la seguente sequenza di soluzioni:

$S_0 \rightarrow$	$S_1 \rightarrow$	$S_2 \rightarrow$	$S_3 \rightarrow$	... $\rightarrow$	S_k
100	98	96	99	…	94

Nel caso in cui non avessimo accettato transizioni verso soluzioni di costo peggiore, la ricerca locale si sarebbe interrotta con la soluzione S_2, che però risulta essere un ottimo locale, ma non globale. Accettando soluzioni peggiori (ad esempio S_3, di costo pari a 99), siamo riusciti a scappare dall'ottimo locale S_2 e a raggiungere dopo k iterazioni un altro ottimo locale S_k di costo minore del costo di S_2.

Tuttavia, anche in questo caso si corre un rischio: quello di tornare in soluzioni già visitate. Ad esempio:

$S_0 \rightarrow$	$S_1 \rightarrow$	$S_2 \rightarrow$	$S_1 \rightarrow$	...
50	48	45	48	…

Bisognerà, quindi, vietare all'algoritmo di "tornare sui propri passi". L'algoritmo che segue tiene conto delle considerazioni appena fatte.

LA TABU SEARCH:

1) Inizializzare la *tabu-list*: inizialmente essa è vuota; sia *ott_corrente* = +∞ (se stiamo considerando un problema di max, altrimenti, nel caso di problemi di min, *ott_corrente* viene inizializzato a -∞) e sia *sol_ott_corrente* = nessuna;
2) Sia S una soluzione iniziale; se $c(S)$ è migliore di *ott_corrente*, allora *ott_corrente* := $c(S)$ e *sol_ott_corrente* := S;
3) Finché non è soddisfatto il criterio di arresto:
    a) Scegli un sottoinsieme $\Im'(S) \subseteq \Im(S)$ di soluzioni non tabù (cioè, che non appartengono alla *tabu-list*);
    b) Scegli $S' \in \Im(S)$ di costo minimo;

c) $S := S'$[6]; aggiorna la *tabu-list*.

Il criterio d'arresto è in genere scelto in uno dei modi seguenti:

- ✓ Numero di iterazioni effettuate;
- ✓ Numero di iterazioni effettuate senza aver ottenuto un miglioramento.

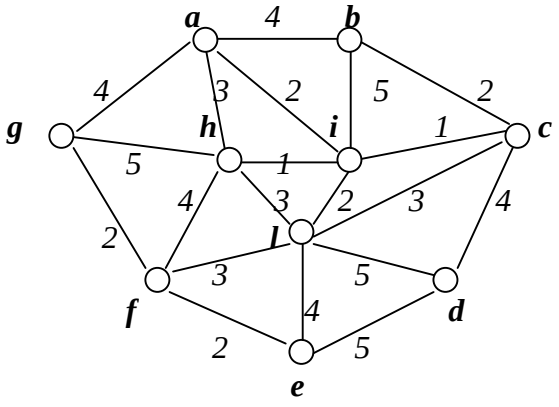
Oss.: Si osservi che: $\Im(S) = \{S' : S' \in \Im(S), S' \notin \text{tabu} - \text{list}\} \supseteq \Im'(S)$.

Le dimensioni della lista tabù vanno scelte accuratamente, tenendo conto delle seguenti considerazioni:

- La lista tabù non deve contenere troppe soluzioni, altrimenti il confronto diventerebbe troppo oneroso;
- Le dimensioni di tale lista dipendono da come è stato definito $\Im'(S)$;
- In genere, scegliere come dimensione della *tabu-list* il valore 2 o 3 non basta per allontanarsi a sufficienza dall'ottimo locale trovato;
- In genere, 7 è un buon numero.

Esempio: *Uniform graph partitioning*

Applichiamo la *tabu-search* al problema di *uniform graph partitioning* sul seguente grafo planare:



- Dimensioni della *tabu-list*: 7;
- Criterio d'arresto: 5 iterazioni;
- Intorno di una soluzione S: $\Im(S)$[7];
- Soluzione iniziale: $<S_0, \overline{S_0}>$, rappresentata nel modo seguente: $S_0 = \{a, b, f, h, g\}$ e di costo pari a 18.

[6] [6] Si osservi che tale istruzione implica che venga accettata sempre la transizione verso la soluzione di costo minimo tra le soluzioni dell'intorno, anche se il suo costo risulta peggiore di quello dell'ottimo corrente.

[7] L'intorno di una soluzione è definito nel modo seguente: $\Im(S_0, \overline{S}_0) = \{<S_0', \overline{S}_0'> \text{ tali che } S_0' = (S_0 \setminus \{x\}) \cup \{y\}, \overline{S}_0' = (\overline{S}_0 \setminus \{y\}) \cup \{x\}, \text{ con } x \in S_0 \text{ e } y \in \overline{S}_0\}$.

Esaminiamo i passi dell'algoritmo attraverso una matrice formata da una riga per ogni nodo in S_0 e da una colonna per ogni nodo in $\overline{S_0}$; in questo modo, vengono evidenziati esclusivamente i pesi degli archi del taglio corrispondente alla partizione $<S_0, \overline{S_0}>$.

			D	*6*	*14*	*7*	*-5*	*8*
			E	*2*	*0*	*2*	*8*	*6*
			I	*8*	*14*	*9*	*3*	*14*
D	***E***	***I***		***c***	***d***	***e***	***i***	***l***
9	*2*	*11*	***a***				2/+8	
-3	*7*	*4*	***b***	2/+7			5/+2	
1	*5*	*6*	***f***			2/+12		3/+15
8	*4*	*12*	***h***				1/+5	3/+22
11	*0*	*11*	***g***					

Dove:

- ***I*** indica il costo interno ed ***E*** il costo esterno, mentre ***D*** = ***I*** - ***E***;
- Nelle caselle del tipo (*i*, *j*) sono indicati: il peso dell'arco (*i*, *j*) e l'incremento che subirebbe la funzione obiettivo se venisse scambiato il nodo $i \in S_0$ con il nodo $j \in \overline{S_0}$;
- Il costo della partizione è dato dalla somma dei pesi degli archi (*i*, *j*) riportati in tabella.

L'intorno di S_0 può essere definito come l'insieme di tutte le soluzioni ottenute scambiando un nodo di S_0 con un nodo di $\overline{S_0}$. Formalmente:

$\Im(S_0) = \{<S', \overline{S'}> : S' = S_0 \setminus \{x\} \cup \{y\}, \ \overline{S'} = \overline{S_0} \setminus \{y\} \cup \{x\} \ \text{ con } \ x \in S_0, y \in \overline{S_0}\}$; oppure:

$\Im(S_0) = \{<S', \overline{S'}> : |S' \cap S_0| = n - 1 \text{ e } |S'| = n\}$.

Poiché i nodi in S_0 possono essere scelti in n modi diversi e così anche i nodi in $\overline{S_0}$, $|\Im(S_0)| = n^2 = 25$.

La ricerca di una nuova soluzione avviene in $\Im'(S_0) \subseteq \Im(S_0)$, che è definito nel modo seguente: $\Im'(S_0) = \{<S', \overline{S'}> \in \Im(S_0)$, non tabù, $(x,y) \in E\}$. In pratica, ci limitiamo a considerare le soluzioni dell'intorno che si ottengono attraverso lo scambio di nodi adiacenti.

Come si può osservare facilmente, $|\Im'(S_0)| = 7$, che è pari al numero degli archi tagliati.

Oss.: L'incremento nella funzione obiettivo che si avrebbe nel caso in cui si decidesse di scambiare i due nodi *x* e *y* è stato calcolato con la solita formula:

$$c(S') = c(S_0) + [I(x) - E(x)] + [I(y) - E(y)] + 2d_{xy} = c(S_0) + D(x) + D(y) + 2d_{xy}.$$

Ad esempio, scambiando il nodo *a* con il nodo *i*, si ottiene una nuova partizione, il cui costo è pari a

$c(S_0) + D(a) + D(i) + 2d_{ai} = 18 + 4 + 4 = 26$. L'incremento della funzione obiettivo sarebbe, allora, pari a 26 - 18 = +8.

Tra tutte le soluzioni appartenenti a $\Im'(S_0)$, quella che ha costo minimo è la seguente:

$$<S_1, \overline{S_1}> \text{ dove } S_1 = \{a, f, h, g, i\};\ \overline{S_1} = \{b, c, d, e, l\}$$

ottenuta da $<S_0, \overline{S_0}>$ tramite lo scambio dei nodi dell'arco (b, i).
E', così, terminata la prima iterazione. Restano da aggiornare le seguenti quantità:

- Soluzione: $S_0 \rightarrow S_1$;
- *Tabu-list*: $\{(b, i)\}$;
- Soluzione corrente: $<S_0, \overline{S_0}>$[8], di costo pari a 18.

			D	-7	8	14	7	4
			E	9	1	0	2	8
			I	2	9	14	9	12
D	**E**	**I**		**b**	**c**	**d**	**e**	**l**
5	4	9	**a**	4/+6				
1	5	6	**f**				2/+12	3/+11
8	4	12	**g**					
10	3	13	**h**					3/+20
-5	8	3	**i**	5[9]	1/+5			2/+3

Adesso si deve ricalcolare la matrice di adiacenza e procedere con una seconda iterazione dell'algoritmo.

In questo caso: $|\Im'(S_1)|$ = numero degli archi del taglio che non appartengono alla lista tabù = 6.

Tra tutte le soluzioni appartenenti a $\Im'(S_1)$, quella che ha costo minimo è la seguente:

$$<S_2, \overline{S_2}> \text{ dove } S_2 = \{a, f, h, g, l\};\ \overline{S_2} = \{b, c, d, e, i\}$$

ottenuta da $<S_1, \overline{S_1}>$ tramite lo scambio dei nodi dell'arco (i, l).

[8] Si noti, infatti, che il costo della partizione $<S_0, \overline{S_0}>$ è minore del costo della partizione $<S_1, \overline{S_1}>$; ciò significa che è stata appena accettata una transizione verso una soluzione peggiore.
[9] E' inutile calcolare l'incremento della funzione obiettivo in questo caso. Infatti i nodi *b* ed *i* non verranno sicuramente scambiati tra loro, dato che l'arco che li collega appartiene alla lista tabù.

E', così, terminata la seconda iterazione. Gli aggiornamenti da fare sono i seguenti:

- Soluzione: $S_0 \rightarrow S_1 \rightarrow S_2$;
- *Tabu-list*: {(*b*, *i*), (*i*, *l*)};
- Soluzione ottima corrente: $<S_0, \overline{S_0}>$, di costo pari a 18.

L'algoritmo prosegue analogamente a quanto visto finora finché non è soddisfatto il criterio di arresto, cioè finché non viene raggiunto il massimo numero di iterazioni (pari a 5).

Oss.: Abbiamo visto finora che la *tabu-list* è stata rappresentata elencando gli archi che connettono i due nodi che sono stati scambiati di insieme in ogni iterazione dell'algoritmo. Essa, tuttavia, può essere rappresentata elencando semplicemente i nodi che sono stati scambiati di insieme. Ad esempio, al termine della prima iterazione, la *tabu-list* può essere rappresentata nei due modi che seguono:

	b	***c***	***d***	***e***	***l***
a	4/+6				
f				2/+12	3/+11
g					
h					3/+20
i	5	1/+5			2/+3

- *Tabu-list*: {(*b*,*i*)};
- *Tabu-list*: {*b*, *i*}.

Tuttavia, scegliendo la seconda rappresentazione della lista tabù, avrei considerato come soluzioni tabù tutte quelle appartenenti alla colonna *b* oppure alla riga *i* della matrice:

In questo modo viene, dunque, vietato di spostare il nodo *b* dall'insieme $\overline{S_1}$ ed il nodo *i* dall'insieme S_1 e non semplicemente di scambiare i due nodi tra di loro! Si sarebbe, perciò, escluso non solo lo scambio (*b*, *i*), ma anche gli scambi (*a*, *b*), (*i*, *c*) e (*i*, *l*). Dunque, la definizione (e,quindi anche la rappresentazione) della lista tabù influisce sul comportamento dell'algoritmo.
In questo caso, l'algoritmo avrebbe operato nel modo seguente. Tra le tre soluzioni che formano $\Im'(S_1)$ scelgo quella che comporta il miglioramento più rilevante (o, come in questo caso avviene, il peggioramento di minore entità): $<S_2', \overline{S_2'}>$, dove $S_2' = \{a, g, h, i, l\}$ e $\overline{S_2'} = \{b, c, d, e, f\}$ (che è ottenuta tramite lo scambio dei nodi *f* ed *l* ed ha costo pari a 31).
Anche in questo caso, l'algoritmo prosegue finché non è verificato il criterio di arresto.