Capitolo 15
Algoritmi euristici

Avviene molto spesso che la risoluzione esatta di un'istanza di un problema **NP**-difficile richieda un tempo eccessivo rispetto alle possibilità pratiche per le quali l'istanza deve essere risolta. In questi casi non resta che affidarsi a procedure che non garantiscono l'ottimalità, ma che sono veloci e forniscono una soluzione spesso accettabile. Algoritmi che operano in questo modo vengono detti *algoritmi euristici* o più semplicemente *euristiche*. Diverse naturalmente possono essere le strategie che stanno alla base di un'euristica.

Vi sono euristiche che privilegiano la velocità di esecuzione, ottenendo una soluzione con una semplice scansione lineare dei dati, in modo simile all'algoritmo greedy per i matroidi, e vengono appunto definite *greedy* anche in questo caso. Normalmente le euristiche greedy producono una soluzione di bassa qualità.

Spesso si possono riconoscere in un problema complesso delle sottostrutture che permettono di decompore il problema in sottoproblemi più semplici, che possono essere risolti in ottimalità. Si tratta poi di aggregare le soluzioni parziali in modo da ottenere la soluzione globale. Spesso in quest'operazione si ottiene informazione sufficiente a rivedere le soluzioni parziali per migliorare la soluzione globale. Questo processo termina quando si ritiene di aver ottenuto una soluzione accettabile o quando la soluzione non è ulteriormente migliorabile.

Un'altra strategia ampiamente usata, in quanto è virtualmente applicabile ad ogni problema, consiste nel generare ricorsivamente una serie di soluzioni ottenute una dall'altra attraverso piccoli miglioramenti. Il tipo di miglioramento che si può realizzare dipende ovviamente dalla struttura del problema in esame. La procedura termina quando non sono più possibili miglioramenti. Questo genere di strategia, nella sua forma più semplice, prende il nome di *ricerca locale*. Sono state anche proposte delle forme più elaborate, due delle quali sembrano dare buoni risultati, una di tipo deterministico, detta *tabu search*, e che si potrebbe chiamare anche *ricerca con memoria*, e l'altra di tipo stocastico, detta *simulated annealing*, che si basa su un'interessante analogia fisica. Inoltre sono stati proposti, sempre come metodi di risoluzione basati su analogie con processi naturali, gli *algoritmi genetici* e le *reti neurali*. L'interesse di queste tecniche per i problemi di ottimizzazione però, a giudizio di chi scrive, si esaurisce nell'analogia. Infatti non sembra possano competere con altri metodi di risoluzione.

15.1. Ricerca locale

Per applicare la ricerca locale ad un problema bisogna preliminarmente definire un sistema di intorni discreti, cioè associare ad ogni soluzione x un sottoinsieme di soluzioni $N(x)$. Vi è grande arbitrarietà nel tipo di intorni che si possono definire. Bisogna tener presente che piccoli intorni rendono ovviamente più rapida la scansione degli elementi in $N(x)$, però i minimi locali discreti con cui la procedura termina sono peggiori di quelli ottenuti con

Algoritmo di ricerca locale
input($x_0 \in F$);
$x := x_0$;
local_optimum := False;
while $\neg$ *local_optimum*
if $\exists y \in F \cap N(x) : \big(f(y) < f(x)\big) \;\wedge\; \big(f(y) \le f(x'), \forall x' \in F \cap N(x)\big)$;
then $x := y$
else *local_optimum* := True;
output(x).

intorni grandi. Sarà compito di chi progetta l'algoritmo decidere la struttura di intorno più conveniente per il problema e per gli scopi prefissi.

Un metodo generale per generare intorni di problemi le cui soluzioni sono vettori binari (moltissimi problemi rientrano in questa categoria) consiste nell'usare la cosiddetta *distanza di Hamming* fra due vettori binari x e y, definita da $d(x,y) := \sum_i |x_i - y_i|$. In altre parole la distanza di Hamming misura quante coordinate vanno cambiate per trasformare un vettore nell'altro. Allora si può definire una famiglia di intorni con parametro λ ponendo

$$N_\lambda(x) := \{y : d(x,y) \le \lambda\}$$

Non deve essere sottovalutato il fatto che si pretende dall'intorno di contenere un buon numero di soluzioni ammissibili, altrimenti la procedura termina rapidamente. Per molti problemi la generazione di soluzioni ammissibili è di per sé difficile e quindi è lecito dubitare che in questi casi un meccanismo di ricerca locale funzioni bene.

L'algoritmo di ricerca locale delineato cerca, per ogni intorno, la soluzione migliore e quindi è obbligato ad esaminare tutte le soluzioni. Si potrebbe adottare la scorciatoia di accettare la prima soluzione generata nella scansione che risulti migliore di quella corrente. Il fatto di rinunciare ad un possibile miglioramento più consistente non significa necessariamente che il risultato finale sia peggiore, anzi spesso avviene proprio il contrario.

La ricerca di una soluzione dell'intorno non è solo di tipo esaustivo, cioè di scansione di singole soluzioni. Per alcuni problemi si può definire l'intorno $N(x)$ come l'unione di insiemi $N_i(x)$, $i := 1, \ldots, p$ dove p è un numero basso e $\min\{f(y) : y \in F \cap N_i(x)\}$ è un problema noto e risolvibile algoritmicamente. In questi casi la ricerca esaustiva consiste nella risoluzione di p problemi.

È evidente che il minimo locale discreto fornito dalla ricerca locale non è soddisfacente nella maggior parte dei casi. Per migliorare la ricerca locale una strategia immediata consiste nel rieseguire la ricerca a partire da molte soluzioni iniziali alternative. Di tutti i minimi locali che vengono prodotti si sceglie alla fine il migliore.

Una seconda strategia migliorativa prevede l'accettazione temporanea di soluzioni peggiori in una generazione sequenziale di un numero basso e predeterminato di soluzioni adiacenti a partire dalla soluzione corrente (*ricerca locale estesa*). Se la soluzione finale della sequenza è migliore di quella iniziale, allora la soluzione corrente viene aggiornata, altrimenti si continua una nuova ricerca dalla soluzione corrente.

Non è vero invece che convenga iniziare la ricerca da una soluzione buona. Normalmente la ricerca locale non migliora di molto una soluzione già buona. È come se si intrappolasse la ricerca in una conca, determinata dalla soluzione iniziale, dalla quale non si possa uscire.

15.1 Esempio. Il TSP è un classico esempio di problema affrontato con le più diverse tecniche, inclusa la ricerca locale, e in questo caso la ricerca locale produce dei risultati

 © 88-08-9059

molto buoni quando i costi sugli archi sono generici. Diverso sarebbe il caso di un TSP con costi sugli archi uguali a 0 o a 1 e si volesse determinare se il grafo indotto dagli archi a costo 0 è o non è hamiltoniano. Per problemi di questo secondo tipo le euristiche non sono di molto aiuto. Sono utili solo quando forniscono la soluzione ammissibile (di costo 0), altrimenti il problema rimane indeciso.

Per il TSP il tipo di intorno che dà buoni risultati si ottiene togliendo da un circuito hamiltoniano due archi arbitrari non adiacenti e 'riattaccando' i due spezzoni di circuito nell'altro senso. Più esattamente, se gli archi che vengono tolti sono (i, j) e (h, k), con i e h appartenenti alla stessa componente connessa del circuito dopo la rimozione degli archi, gli archi che si aggiungono sono (i, k) e (j, h) (figura 15.1). Si noti che, identificando una soluzione con il vettore d'incidenza di un sottoinsieme di archi, è un tipo di intorno N_λ con $\lambda = 4$. Solo valori pari di λ sono ammessi per avere soluzioni ammissibili e per $\lambda = 2$ l'unica soluzione ammissibile dell'intorno è la soluzione stessa (tolto un arco non resta che rimetterlo al suo posto). Quindi il primo intorno significativo si ottiene per $\lambda = 4$.

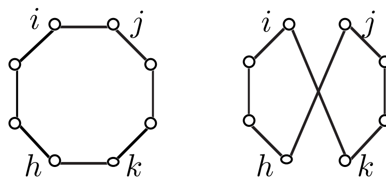

Figura 15.1

Per $\lambda = 6$ si tolgono tre archi non adiacenti e si aggiungono tre archi in modo da formare un circuito. In generale se si tolgono k archi non adiacenti rimangono k pezzi di circuito che si possono riattaccare assieme secondo $(k-1)!$ permutazioni e orientare in due modi. Tenendo conto che ogni soluzione compare due volte (anche nel verso opposto) vi sono $(k-1)!\,2^{k-1} - 1$ soluzioni nell'intorno. Quindi per $k = 3$ ($\lambda = 6$) vi sono 7 soluzioni di cui tre corrispondono al caso $\lambda = 4$ e le altre quattro sono indicate in figura 15.2.

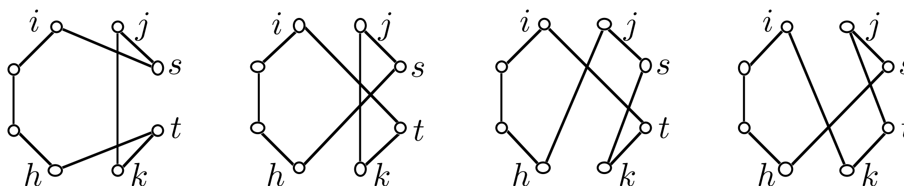

Figura 15.2

Vi è un aspetto favorevole in queste definizioni di intorno e riguarda la possibilità di confrontare rapidamente due soluzioni x e y senza dover valutare $f(x)$ e $f(y)$. Nel caso del TSP il calcolo di $f(x)$ ha complessità $O(n)$ perché richiede la somma di n costi. Ma per i tipi di intorno definiti N_4 e N_6, il calcolo di $f(x) - f(y)$, con $y \in N(x)$, non ha complessità $O(n)$, bensì costante, perché è semplicemente dato da $c_{ij} + c_{hk} - c_{ik} - c_{jh}$, per N_2, e da una espressione simile per N_6.

Bisogna comunque tener presente che una ricerca esaustiva dell'intorno N_4 costa $O(n^2)$ e dell'intorno N_6 costa $O(n^3)$. Se n è molto elevato (decine di migliaia) è più indicato accettare la prima soluzione migliore di ogni intorno (ma in prossimità di un minimo locale l'intorno va esplorato quasi integralmente).

È stato provato empiricamente da Lin [1965] che un buon valore per λ è 6. Sotto opportune ipotesi si è visto che la probabilità che un ottimo locale sia globale è attorno al 5%. Quindi la migliore soluzione fra 100 soluzioni ottenute da altrettante soluzioni iniziali diverse ha una probabilità di essere ottima globale pari a $1 - 0.95^{100} = 1 - 0.0059$ cioè più del 99%.

Un ulteriore miglioramento è stato apportato da Lin e Kernighan [1973] usando una ricerca locale estesa. L'aumento in tempo di calcolo viene più che compensato dall'elevata qualità della soluzione. Infatti questa euristica è ancora oggi considerata la migliore.

In figura 15.3 sono visualizzate tre soluzioni della stessa istanza di TSP (70 nodi a caso su una griglia il cui elemento ha lunghezza unitaria) ottenute dalla ricerca locale che scambia due archi. Le lunghezze dei tre circuiti sono nell'ordine 92.16, 106.68 e 117.8. Come si può già vedere dalle figure non si tratta di soluzioni particolarmente buone. Un netto vantaggio si ottiene se si usa l'intorno che scambia tre archi. In figura 15.4 si vedono tre soluzioni ottenute con questo tipo di ricerca locale. Le lunghezze dei tre circuiti sono rispettivamente 76.36, 76.12 e 76.92. ∎

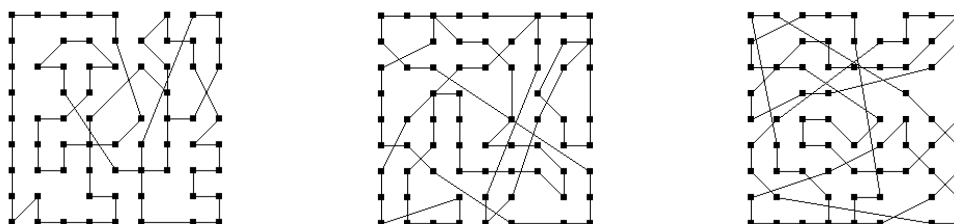

FIGURA 15.3

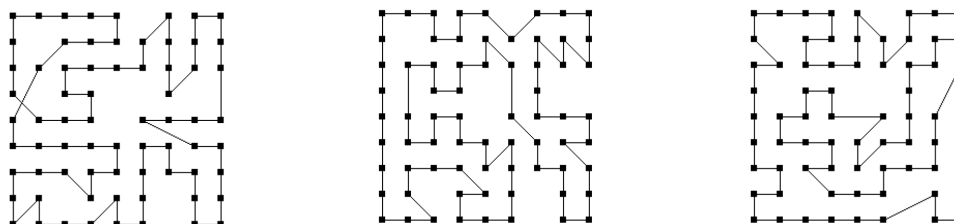

FIGURA 15.4

15.2 ESEMPIO. Il *problema del job-shop* è un problema di schedulazione definito nel seguente modo: sono assegnati n lavori. Ogni lavoro viene eseguito con una successione predeterminata di operazioni. Lavori diversi possono avere un numero diverso di operazioni. Sono date m macchine e ogni operazione viene eseguita da una predeterminata macchina con una nota durata di esecuzione. Sia p_j la durata dell'operazione j. Una macchina può eseguire un'operazione alla volta e non può interrompere un'esecuzione già iniziata. Si vuole trovare la schedulazione che minimizza il massimo tempo di completamento dei lavori.

Si tratta di un problema particolarmente intrattabile. Una famosa istanza con 10 lavori, 10 operazioni per lavoro e 10 macchine, nota come $10 \times 10 \times 10$, fu proposta da Fisher e Thompson [1963]. Le durate furono generate casualmente con una distribuzione uniforme fra 1 e 100 e anche l'assegnazione delle operazioni alle macchine fu generata casualmente.

Questa istanza fu affrontata da molti ricercatori, ma senza successo. Si vedano ad esempio i commenti in Lageweg et al. [1977] che, essenzialmente, riportano un insuccesso. L'ottimo fu trovato infine da Carlier e Pinson [1989] con cinque ore di tempo macchina. Altri metodi esatti si possono trovare in Applegate e Cook [1991].

Il problema può essere modellato con un cosiddetto *grafo disgiuntivo* (figura 15.5), in cui i nodi rappresentano le operazioni, incluse due operazioni fittizie di durata nulla (i nodi quadrati in figura) che identificano l'istante iniziale e quello finale della schedulazione. Nella figura le macchine associate alle operazioni normali sono identificate dal colore del nodo (nero, grigio, bianco). Gli archi si dividono in archi congiuntivi che rappresentano precedenze obbligate fra le operazioni (quelle relative ad ogni lavoro), indicati come archi orientati in figura, e archi disgiuntivi che rappresentano precedenze da determinare (cioè l'ordine di esecuzione delle operazioni di una macchina), indicati come archi non orientati e in tratteggio in figura.

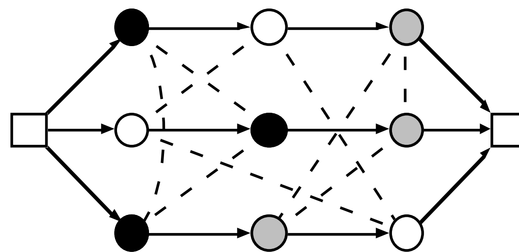

FIGURA 15.5

Una volta stabilita l'orientazione degli archi disgiuntivi (basta fissare un cammino hamiltoniano per i nodi di ogni macchina), trovare la schedulazione è un semplice problema di cammino di lunghezza massima (la lunghezza di un arco orientato è data dalla durata di esecuzione dell'operazione del nodo da cui esce) su un grafo aciclico che in questo caso si risolve con complessità lineare nel numero di operazioni. Supponiamo che le durate dell'istanza in figura 15.5 siano le seguenti (nel seguito ogni operazione è individuata dall'indice 1, 2 o 3 a seconda del lavoro a cui appartiene, numerando dall'alto in basso e dalla lettera N (nero), B (bianco) o G (grigio) a seconda della macchina cui è assegnata):

$$\begin{array}{lll} p_{1N} = 5 & p_{1B} = 8 & p_{1G} = 3 \\ p_{2B} = 7 & p_{2N} = 4 & p_{2G} = 6 \\ p_{3N} = 2 & p_{3G} = 5 & p_{3B} = 4 \end{array}$$

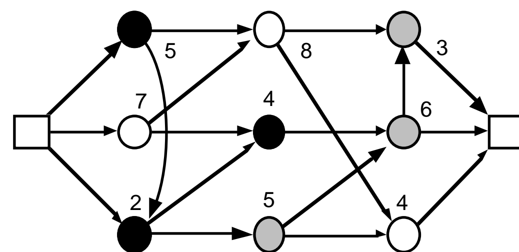

FIGURA 15.6

Se si scelgono le orientazioni indicate in figura 15.6, il tempo minimo di completamento vale 21. In figura è anche indicato il cammino più lungo, il cosiddetto cammino critico. Una

immediata caratterizzazione dell'intorno di una soluzione si ottiene pensando di cambiare, una alla volta, l'orientazione di ogni arco disgiuntivo (purché questo non generi un ciclo e quindi una soluzione inammissibile). Tuttavia è ovvio che cambiare l'orientazione di un arco disgiuntivo non sul cammino critico non può migliorare la soluzione corrente (il cammino critico è ancora presente e quindi il nuovo cammino più lungo non può essere più corto dell'attuale cammino critico). Allora l'intorno è definito da tutte le schedulazioni che si ottengono variando, una alla volta, le orientazioni di archi disgiuntivi sul cammino critico. In termini di λ-intorni questo intorno ha un valore $\lambda = 1$. Non è nemmeno difficile dimostrare che le soluzioni dell'intorno sono tutte ammissibili (cioè non generano cicli). Nell'esempio sono tre le soluzioni dell'intorno e danno luogo ai tre grafi orientati e aciclici in figura 15.7, le cui soluzioni valgono rispettivamente 20, 25 e 22. La prima delle tre è ottimo locale, come si può verificare facilmente. ▮

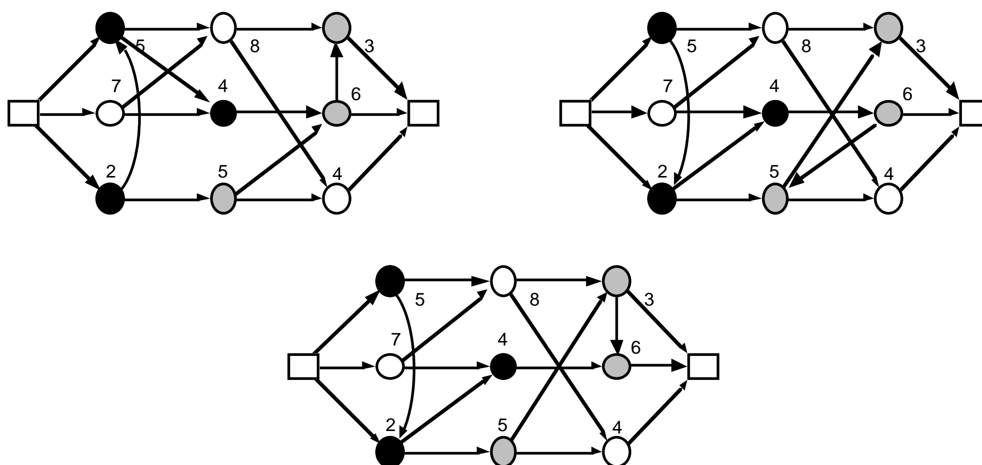

Figura 15.7

15.3 Esempio. Un diverso e più complesso tipo di intorno per il problema del job-shop può essere definito nel seguente modo: si consideri una macchina alla volta e si rischeduli in ottimalità la singola macchina; allora la soluzione che si ottiene è una soluzione dell'intorno.

Per risolvere il problema di schedulare una singola macchina ci si riconduce al problema ad una macchina considerato nell'esempio 14.16. Si consideri ad esempio la soluzione di figura 15.6 e si decida di rischedulare la macchina nera. Allora si eliminano gli archi disgiuntivi relativi a tale macchina (figura 15.8) e si calcolano i cammini più lunghi dal nodo inizio alle operazioni della macchina e i cammini più lunghi dalle operazioni al nodo fine.

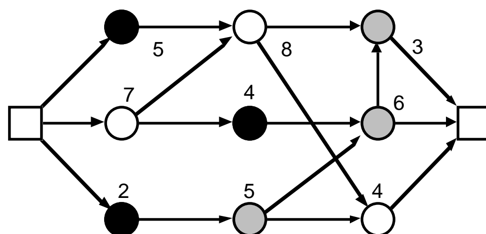

Figura 15.8

Il valore del cammino più lungo dal nodo inizio all'operazione sta ad indicare che l'operazione può iniziare ad essere eseguita solo dopo questo tempo, quindi ha il significato di una data di rilascio r_j. Il valore del cammino più lungo dall'operazione al nodo fine sta ad indicare che dopo l'esecuzione dell'operazione vi è una coda di lavori che rappresenta un tempo che comunque deve trascorrere e di cui si deve tenere conto, quindi hanno il significato di code q_j. Ecco i dati che si ottengono per i tre problemi ad una macchina (rispettivamente nera, bianca e grigia):

	r	p	q
1	0	5	12
2	7	4	9
3	0	2	14

	r	p	q
1	5	8	3
2	0	7	13
3	7	4	0

	r	p	q
1	15	3	0
2	11	6	0
3	7	5	4

Questi problemi si risolvono in generale con l'algoritmo illustrato nell'esempio 14.16. Anche se si tratta pur sempre di un problema **NP**-difficile, tuttavia si può notare che l'ordine dei lavori secondo date di rilascio crescenti è abbastanza correlato, per come i dati sono generati, con l'ordine secondo code decrescenti. Se i due ordini coincidono allora tale ordine è ottimo, altrimenti c'è da aspettarsi qualche sviluppo non eccessivo dell'albero di ricerca.

Nella fattispecie i due ordini coincidono per le tre macchine e quindi le tre schedulazioni ottime (relativamente a problemi ad una macchina) sono: macchina nera $3 \to 1 \to 2$, macchina bianca $2 \to 1 \to 3$, macchina grigia $2 \to 1 \to 3$. Delle tre schedulazioni solo la prima produce un miglioramento e quindi può essere usata per generare una nuova soluzione corrente (è la medesima del primo grafo in figura 15.7).

15.4 Esempio. Il problema del minimo albero di Steiner è stato definito nell'esempio 1.48. È un problema **NP**-difficile e quindi la scelta di un'euristica per risolverlo è giustificata. Sia $G = (N, E)$ il grafo e sia $T \subset N$ l'insieme dei nodi obbligati. Una semplice idea di intorno si basa sull'osservazione che se si decide a priori quale debba essere l'insieme $S \subset N \backslash T$ dei nodi di Steiner, allora il problema si riduce ad un minimo albero di supporto sul grafo indotto $G(S) := (T \cup S, E(T \cup S))$. Se $E(T \cup S)$ è sconnesso la scelta di S non è ammissibile. Quindi l'elemento fondamentale di decisione è il vettore d'incidenza di S come sottoinsieme di $N \backslash T$. Una definizione di intorno N_λ con $\lambda = 1$ considera quindi, dato S, tutti i sottoinsiemi $S \cup \{i\}$, $i \notin T \cup S$, e $S \backslash \{i\}$, $i \in S$. La cardinalità dell'intorno è pertanto $r := |N \backslash T|$.

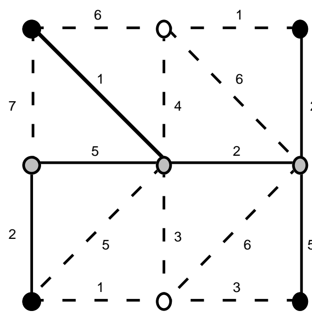

Figura 15.9

A titolo illustrativo si consideri la medesima istanza dell'esempio 1.48 e si prenda come insieme iniziale S quello dei nodi grigi in figura 15.9. L'insieme T è dato dai nodi neri. Il valore della soluzione iniziale è 17. Due delle soluzioni dell'intorno non sono ammissibili. In

figura 15.10 sono indicate le tre soluzioni ammissibili dell'intorno che valgono 18, 15 e 14. Quindi la soluzione iniziale non è un ottimo locale discreto e l'iterazione può ripartire dalla soluzione di valore 14. Si può verificare subito che nell'intorno di questa soluzione è incluso anche l'ottimo globale del problema di valore 12 (figura 1.19).

Bisogna comunque notare che questo intorno è troppo piccolo per poter dare in generale una soluzione soddisfacente. È meglio allargare l'intorno a $\lambda = 2$. In questo caso la cardinalità dell'intorno diventa $\binom{r}{2} + r$ che è 15 nel nostro caso. Nell'esempio specifico, dato che vi sono $2^r = 32$ soluzioni da esplorare, un intorno con 15 soluzioni, quasi la metà del totale, è troppo grande ed infatti il semplice intorno di 5 soluzioni è sufficiente a produrre l'ottimo globale, ma se il grafo è più grande la situazione non è sempre così favorevole. ∎

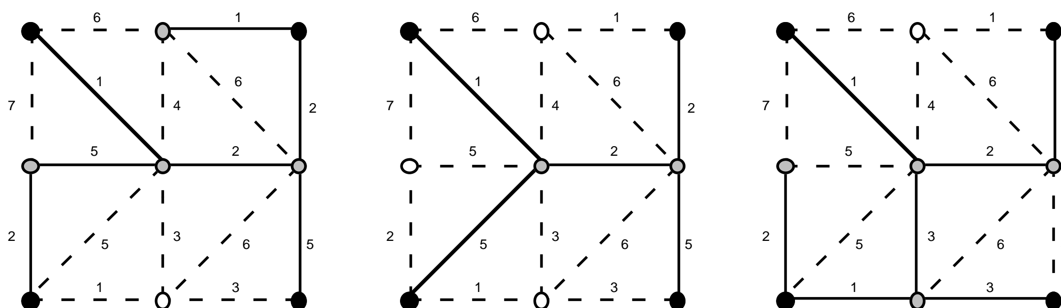

FIGURA 15.10

15.5 ESERCIZIO. Si applichi la ricerca locale ad un problema di knapsack con intorni di tipo N_λ. Cosa succede se si prende $\lambda = 1$? Sotto quali condizioni si migliora la soluzione corrente con $\lambda = 2$? ∎

15.2. Metodi greedy

È impossibile dare un'idea della varietà dei metodi greedy che si usano nelle varie applicazioni, dato che virtualmente ogni progettista di algoritmi ne ha sperimentato qualcuno e che ogni idea si presta in un modo o nell'altro a essere usata come criterio del metodo.

In linea generale un metodo greedy costruisce una soluzione selezionando i suoi costituenti (elementi di un insieme nella maggior parte dei casi) uno alla volta, in base ad un criterio predefinito e senza mai ritornare sulle selezioni già fatte. Quindi tutta l'ingegnosità di un metodo greedy risiede nel particolare criterio che presiede alla selezione.

Siccome tali metodi hanno indubbiamente il requisito di un'elevatissima velocità di calcolo, non costa molto in termini computazionali usare ripetutamente tale metodo, ma con vari criteri alternativi. Delle soluzioni proposte si accetta poi ovviamente la migliore.

Sperimentalmente si è notato un fatto curioso e di cui non si riesce a dare una spiegazione convincente. Quando siano disponibili più criteri di selezione, anziché usare coerentemente sempre lo stesso criterio nella costruzione della soluzione, conviene adottare in modo casuale un criterio diverso ad ogni selezione. Si ottengono soluzioni migliori.

15.6 ESEMPIO. Per il TSP sono molti i metodi greedy proposti e nessuno di essi si è dimostrato superiore agli altri (si veda Lawler et al [1985], Jünger et al [1995]). Tipicamente la soluzione viene costruita aggiungendo un nodo alla volta secondo regole diverse.

1) selezionare il nodo più vicino all'ultimo nodo inserito;
2) dato un circuito parziale, selezionare un nodo a caso e inserirlo fra la coppia più favorevole del circuito parziale; si inizia con un circuito a caso di tre nodi;
3) dato un circuito parziale, selezionare il nodo più vicino al circuito e inserirlo fra la coppia più favorevole del circuito parziale; si inizia con un circuito a caso di tre nodi;
4) dato un circuito parziale, selezionare il nodo più lontano al circuito e inserirlo fra la coppia più favorevole del circuito parziale; si inizia con un circuito a caso di tre nodi;

Nelle figure 15.11 e 15.12 si vedono le soluzioni ottenute rispettivamente con il metodo 1) e il metodo 2). I loro valori sono 80.88 e 74.72.

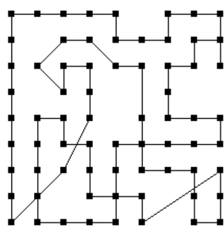

FIGURA 15.11

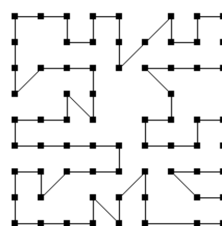

FIGURA 15.12

15.7 ESEMPIO. Il problema del job-shop può essere risolto con dei metodi greedy che si basano su una simulazione temporale della progressione dei lavori. Ad un generico istante t vi sono insiemi di operazioni $J_1, J_2, \ldots, J_m$ pronte per essere eseguite sulle macchine $1, 2, \ldots, m$ rispettivamente. Alle macchine libere all'istante t vengono assegnate operazioni selezionate dal relativo insieme J_k (che vengono quindi aggiornati). Esauriti gli assegnamenti le macchine cominciano le esecuzioni. Il tempo t viene aggiornato al primo istante utile, cioè quello in cui una macchina termina un'esecuzione, diventa disponibile e rende disponibile l'operazione successiva a quella appena terminata. Si procede quindi iterativamente finché tutto è terminato. Naturalmente si inizializza con $t := 0$. Il metodo procede quindi con complessità lineare rispetto al numero di operazioni.

Tutte le volte in cui si deve scegliere un lavoro disponibile da un insieme J_k che ne contenga almeno due possiamo decidere di attuare la scelta in base ad un prefissato criterio. Si noti che non è detto che fra tutte le soluzioni che si possono ottenere in base a tutte le possibili scelte, sia necessariamente presente l'ottimo. L'ottimo potrebbe contemplare l'ulteriore scelta di *non* mettere in esecuzione nessuna operazione disponibile in un dato istante, perché potrebbe essere più conveniente aspettare, magari di poco, la disponibilità di un'operazione importante. Il metodo greedy invece mette sempre in esecuzione una macchina se questa è disponibile e ci sono operazioni da farle fare. I criteri di selezione che si usano normalmente assegnano priorità all'operazione che:

– è diventata disponibile per prima (FCFS - *First Come First Serve*);
– è diventata disponibile per ultima (LCFS - *Last Come First Serve*);
– ha la maggior quantità di tempo-lavoro che deve trascorre dopo il suo completamento, cioè ha la catena di successori più lunga (MWKR - *Most WorK Remaining*);
– ha la durata più corta (SPT - *Shortest Processing Time*);
– ha la durata più lunga (LPT - *Longest Processing Time*);
– ha il più alto numero di successori (MIS - *Most Immediate Successors*).

Queste regole di priorità non sono le uniche presenti in letteratura. Sono però le più usate. Come già precedentemente sottolineato, a queste regole di selezione bisogna aggiungere una regola casuale, in cui la casualità non riguarda tanto l'operazione da scegliere quanto il criterio da adottare. E quest'ultima regola si dimostra normalmente la migliore.

Considerando l'istanza dell'esempio 15.2 e applicando la regola SPT otteniamo la schedulazione in figura 15.13, in alto. Applicando la regola LPT oppure MWR (in questo caso producono lo stesso risultato) si ottiene la seconda schedulazione in figura 15.13. L'ottimo globale è dato dalla terza schedulazione in figura 15.13 (un misto delle due precedenti di fatto).

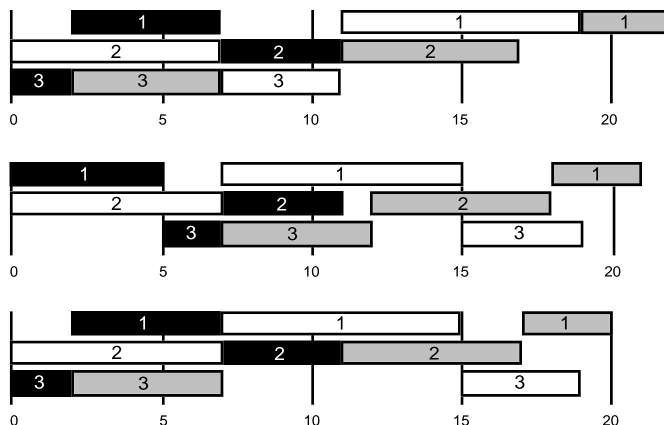

Figura 15.13

Per avere un'idea della qualità di una soluzione prodotta dall'euristica greedy si consideri che l'istanza $10 \times 10 \times 10$ precedentemente citata ha valore ottimo 930, mentre un'euristica casuale produce tipicamente un valore compreso fra 1000 e 1050. ∎

15.8 Esempio. Per il problema del bin packing esistono vari metodi greedy particolarmente efficaci. In realtà si tratta anche di metodi di approssimazione (vedi più avanti). Il problema del bin packing è già stato presentato negli esempi 1.76, 7.40 e 14.6.

Si ordinino gli oggetti per grandezze decrescenti e poi si assegnino gli oggetti (nell'ordine) al contenitore di indice più basso con sufficiente spazio libero da contenere l'oggetto. Si tratta di un algoritmo di complessità $O(n \log n)$. Per quel che riguarda la qualità della soluzione è stato dimostrato da Johnson [1973] (si veda anche Baker [1983], Yue [1991] e Coffman et al.[1997]) che l'errore relativo del valore della soluzione rispetto all'ottimo non è più di 2/9. Questa è una limitazione di caso peggiore e in pratica l'errore è molto ridotto.

L'idea può essere usata per il problema collegato di multiprocessor scheduling, dove il numero dei contenitori è fisso e si vuole minimizzare il massimo riempimento. Sia m il numero dei contenitori. Inizialmente si ipotizza un riempimento ottimo C^* e si risolve un problema di bin packing, con l'algoritmo appena delineato, con questo valore di capacità. Se il risultato fornito dall'algoritmo prevede più di m contenitori allora bisogna rivedere il valore di C^* aumentandolo, altrimenti lo si diminuisce. I valori vengono aumentati o diminuiti secondo una tipica ricerca binaria. È stato dimostrato che, se l'iterazione consiste di k passi, l'errore relativo della soluzione ottenuta non è mai superiore a $6/5 + (1/2)^k$. ∎

15.3. Disaggregazione

Se le dimensioni del problema non consentono un approccio che tenga conto simultaneamente di tutti i vincoli, e se parti del problema possono essere isolate e risolte con metodi noti, ammesso che le altre grandezze del problema abbiano valori fissati, allora in questi casi un approccio che disaggreghi le varie parti del problema e poi trovi il modo di aggregarle, possibilmente in modo iterativo, può essere davvero raccomandabile.

15.9 Esempio. Il problema del job-shop (vedi esempi 15.2 e 15.3) è stato affrontato da Adams et al. [1988] secondo una tecnica che in parte possiamo definire di disaggregazione e in parte di ricerca locale. Rimandiamo agli esempi citati per le definizioni e i concetti connessi. Il problema viene disaggregato secondo le macchine, ovvero si schedulano le macchine singolarmente e non simultaneamente. L'aggregazione delle macchine avviene tramite il passaggio dei dati globali ai singoli problemi ad una macchina. Per essere più specifici si consideri nuovamente l'istanza dell'esempio 15.2 e si tolgano dal grafo disgiuntivo tutti gli archi disgiuntivi, in modo da ottenere il grafo in figura 15.14.

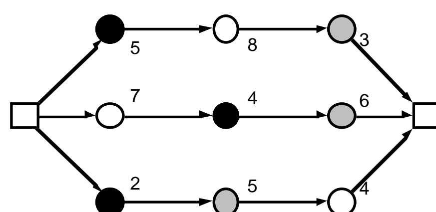

Figura 15.14

Da questo grafo vengono derivate tre istanze di problema ad una macchina nello stesso modo dell'esempio 15.3 (rispettivamente nera, bianca e grigia):

	r	p	q
1	0	5	11
2	7	4	6
3	0	2	9

	r	p	q
1	5	8	3
2	0	7	10
3	7	4	0

	r	p	q
1	13	3	0
2	11	6	0
3	2	5	4

Queste istanze vengono risolte indipendentemente dando luogo alle seguenti soluzioni: macchina nera $1 \to 3 \to 2$ con valore 17, macchina bianca $2 \to 1 \to 3$ con valore 19, macchina grigia $3 \to 2 \to 1$ con valore 20. A questo punto le tre soluzioni potrebbero direttamente venire aggregate definendo gli archi disgiuntivi nel grafo disgiuntivo. Tuttavia in questo modo non si ottiene una buona soluzione. L'aggregazione, per così dire, ha tenuto poco in conto l'interazione reciproca delle varie macchine. Il metodo suggerito da Adams et al. [1988] prende in considerazione solo una delle schedulazioni. Bisogna allora definire un criterio di scelta. È sensato pensare di fissare subito la schedulazione di quella macchina che incide di più sul valore finale della soluzione del job-shop. Tale macchina viene definita critica. Si tratta ora di definire il concetto di criticità. Si consideri il cammino più lungo nel grafo di figura 15.14, che è dato dagli archi del secondo lavoro ed ha lunghezza 17. Le tre macchine, una volta schedulate, incrementano questo valore di 0, 2 e 3 rispettivamente. Quindi la terza macchina è quella che presumibilmente incide di più sul valore della schedulazione finale ed allora viene considerata critica. Si passa quindi al grafo in figura 15.15, il cui cammino più lungo vale 20. Si noti che il valore del cammino più lungo nel grafo disgiuntivo, una

volta inserite le precedenze della macchina, coincide con il valore ottimo del problema ad una macchina per la stessa macchina. Questo succede nella maggior parte dei casi, ma in situazioni complesse ciò non può essere vero, nel senso che il cammino più lungo nel grafo disgiuntivo è maggiore del valore del problema ad una macchina. Sotto quali condizioni avviene questo fatto? A questo riguardo si veda Balas et al. [1995].

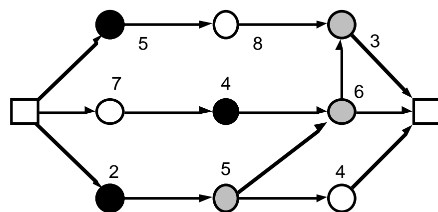

FIGURA 15.15

Da questo grafo vengono dedotte due istanze di problema ad una macchina per le macchine nera e bianca rispettivamente:

	r	p	q
1	0	5	11
2	7	4	9
3	0	2	14

	r	p	q
1	5	8	3
2	0	7	13
3	7	4	0

con ottimi: macchina nera $3 \to 1 \to 2$ con valore 20, macchina bianca $2 \to 1 \to 3$ con valore 20. I valori sono uguali quindi possiamo decidere arbitrariamente quale sia la macchina critica. Sia ad esempio la macchina bianca. Il grafo che si ottiene inserendo la schedulazione della macchina bianca è il medesimo della figura 15.8. A questo punto prima di procedere ad inserire la schedulazione dell'ultima macchina è conveniente operare una ricerca locale sulle macchine già schedulate. Nell'esempio ciò non produce miglioramenti, ma in generale questa fase di ricerca locale, o di riottimizzazione come viene denominata, produce una migliore schedulazione delle macchine già schedulate. In generale il metodo continua in modo iterativo scegliendo la macchina critica, una alla volta, fra quelle non ancora schedulate, e riottimizzando con ricerca locale quelle già schedulate.

Nell'esempio non resta che schedulare la macchina nera producendo la soluzione del primo grafo in figura 15.7 e del terzo diagramma in figura 15.13 (e si tratta effettivamente dell'ottimo).

Questa 'ricetta' è stata denominata *shifting bottleneck procedure* da Adams et al. [1988], in quanto la macchina più critica (quella che realizza il 'collo di bottiglia') varia da iterazione a iterazione. In termini di qualità della soluzione e di tempi di calcolo l'euristica produce risultati estremamente buoni. Ad esempio l'istanza $10 \times 10 \times 10$ già citata viene risolta in pochi secondi con un valore che, a seconda dei dettagli implementativi, è l'ottimo 930 oppure 931. La bontà del metodo ha suggerito un'estensione a problemi in cui siano presenti date di scadenza rigide per i valori (si veda Balas et al. [1998]). ∎

15.10 ESEMPIO. Si consideri un'estensione del problema di multiprocessor scheduling. Ad ogni lavoro sia associata anche una data di scadenza e l'obiettivo sia la minimizzazione del massimo ritardo (il problema originario corrisponde al caso di date di scadenza tutte uguali). Per questo problema esiste un algoritmo sviluppato da Carlier [1987] che trova l'ottimo con una tecnica branch-and-bound. L'algoritmo è alquanto complesso e, a differenza del simile algoritmo per il problema ad una macchina (Carlier [1982]), presenta tempi medi

di esecuzione abbastanza lunghi. Supponiamo di dover sviluppare un algoritmo che debba soprattutto essere veloce. A scopo puramente illustrativo di come si possa decomporre un problema, anche in modi alternativi, presentiamo ora due possibili metodi.

Nel primo metodo il problema viene affrontato omettendo il vincolo sul numero di macchine. I lavori vengono schedulati come se ciascuno disponesse di una propria macchina (cosiddetta risorsa infinita). L'aggregazione verrà fatta imponendo vincoli di precedenza. Consideriamo la seguente istanza: 6 lavori di durate $p_1 = 4$, $p_2 = 5$, $p_3 = 2$, $p_4 = 4$, $p_5 = 6$ e $p_6 = 3$, e di scadenze $d_1 = 7$, $d_2 = 8$, $d_3 = 10$, $d_4 = 10$, $d_5 = 11$ e $d_6 = 12$. Come in casi analoghi trasformiamo le scadenze in code $q_j := \max_i d_i - d_j$, e quindi $q_1 = 5$, $q_2 = 4$, $q_3 = 2$, $q_4 = 2$, $q_5 = 1$ e $q_6 = 0$. Da questi dati ricaviamo il grafo orientato in figura 15.16 (sugli archi è indicato il valore $p_j + q_j$) sul quale si calcola la schedulazione ottima per ogni lavoro con un cammino più lungo dal nodo terminale all'indietro. Questo è equivalente a schedulare i lavori a ridosso delle date di scadenza. Si costruisca ora un diagramma (figura 15.17) che rappresenta il numero di macchine attive in funzione del tempo. È un diagramma che aumenta di 1 negli istanti di inizio di un lavoro e diminuisce di 1 negli istanti di completamento.

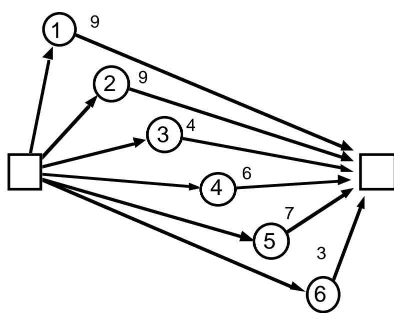

FIGURA 15.16

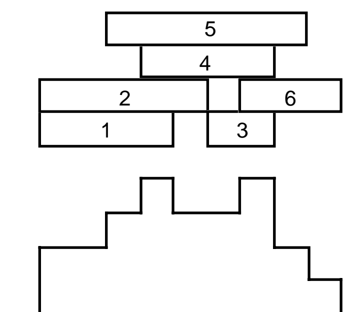

FIGURA 15.17

I due picchi del diagramma indicano che sono necessarie 4 macchine per realizzare la schedulazione. Siccome le macchine sono solo due bisogna livellare i picchi. Un semplice modo per realizzare il livellamento del primo picco consiste nell'anticipare l'evento completamento del lavoro 1 prima dell'evento inizio del lavoro 4 e per il livellamento del secondo picco bisogna anticipare l'evento completamento del lavoro 3 o 4 (scegliamo ad esempio il lavoro 3) prima dell'evento inizio del lavoro 6. Per realizzare questi vincoli basta aggiungere delle precedenze nel grafo (vedi figura 15.18), ottenendo la schedulazione e il diagramma di figura 15.19.

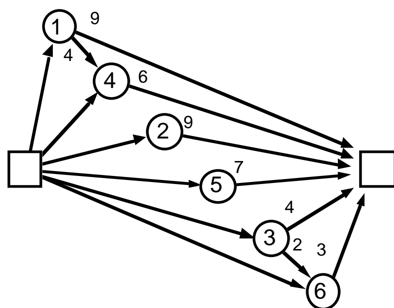

FIGURA 15.18

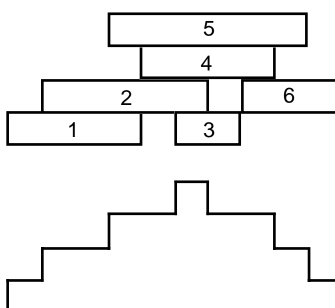

FIGURA 15.19

Aggiungendo la precedenza fra i lavori 2 e 3 si ottiene una schedulazione per la quale bastano tre macchine (figura 15.20). Per arrivare a solo due macchine è necessario aggiungere ancora una precedenza fra 4 e 5, il che porta al diagramma finale di figura 15.21.

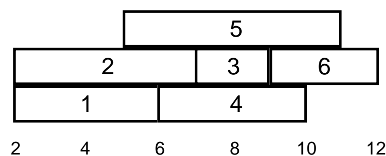

FIGURA 15.20

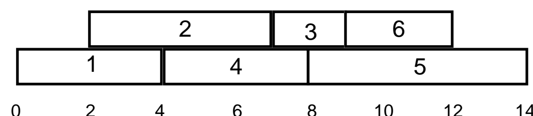

FIGURA 15.21

Questa soluzione schedula tutti i lavori tranne il 5 entro le scadenze. Per il lavoro 5 il ritardo è di tre unità temporali. Siccome l'obiettivo è la minimizzazione del massimo ritardo non si tratta di una soluzione molto buona. Il lettore stesso può migliorare questa soluzione operando una ricerca locale che scambia lavori fra le macchine.

Il metodo di disaggregazione visto è effettivamente troppo elementare per poter dare buoni risultati. Come regola di buon senso, la decomposizione è tanto migliore quanto più sono presenti gli aspetti combinatori del problema dato nei singoli sottoproblemi. Nell'esempio precedente il semplice 'congelamento' di una precedenza, senza valutare altre alternative, fa perdere la ricchezza combinatoria del problema originale e spiega la povertà della soluzione ottenuta.

Il secondo metodo di disaggregazione che presentiamo fa un uso migliore della struttura del problema. Si noti allora che il problema presenta due aspetti interconnessi: l'assegnazione dei lavori alle macchine e la successiva schedulazione. Se le scadenze fossero uguali, sarebbe un problema di multiprocessor scehduling per il quale sono già note euristiche veloci (vedi esempio 15.8) e queste quindi potrebbero essere usate per l'assegnamento. Se ci fosse solo una macchina, la schedulazione ottima ordinerebbe i lavori secondo scadenze crescenti. Essendo in realtà presenti entrambi gli aspetti, si tratta di capire come aggregarli assieme in modo da perseguire l'obiettivo.

Notiamo che l'assegnazione dei lavori alle macchine fatta secondo l'euristica del multiprocessor scheduling tiene conto solo delle durate dei lavori. Per aggiungere l'aspetto relativo alle scadenze possiamo notare che nel momento in cui un lavoro viene inserito nell'ipotetico contenitore-macchina deve esserci spazio libero per la propria durata e per la propria coda. Quindi possiamo modificare l'euristica del multiprocessor scheduling ordinando i lavori per valori decrescenti di $p_j + q_j$, quindi per l'istanza dell'esempio si ha l'ordinamento $p_1 + q_1 = 9$, $p_2 + q_2 = 9$, $p_5 + q_5 = 7$, $p_4 + q_4 = 6$, $p_3 + q_3 = 4$, e $p_6 + q_6 = 3$. Il valore iniziale di capacità viene calcolato semplicemente assegnando i lavori ai due contenitori con la regola di inserire il lavoro corrente nel contenitore meno riempito. Siano R_1 e R_2 i valori di riempimento correnti, ponendo inizialmente $R_1 := R_2 := 0$. Quindi si assegna il lavoro 1 al contenitore 1, aggiornando $R_1 := 4$, il lavoro 2 al contenitore 2 ($R_2 := 5$), il lavoro 5 ad 1 ($R_1 := 4+6 = 11$), il lavoro 4 a 2 ($R_2 := 5 + 4 = 9$), il lavoro 3 a 2 ($R_2 := 9 + 2 = 11$), il lavoro 6 a 2 ($R_2 := 11 + 3 = 14$). Fatta l'assegnazione ($\{1, 5\} \rightarrow$ macchina 1, $\{2, 4, 3, 6\} \rightarrow$ macchina 2)

bisogna schedulare in ottimalità i lavori, in ordine di code decrescenti, quindi sulla macchina 1 si ha la sequenza $\{1 \to 5\}$ con tempi di completamento $C_1 = 4$, $C_5 = 4 + 6 = 10$ e sulla macchina 2 si ha la sequenza $\{2 \to 3 \to 4 \to 6\}$ con tempi di completamento $C_2 = 5$, $C_3 = 5 + 2 = 7$, $C_4 = 7 + 4 = 11$, $C_6 = 11 + 3 = 14$. Il valore di questa schedulazione è $\max_j \{C_j + q_j\} = 14$. Ai fini della ricerca binaria fissiamo quindi come estremo destro il valore di capacità 14 (perché abbiamo la garanzia che una soluzione esiste) e come estremo sinistro il valore $\max_j d_j = 12$ (perché basta schedulare entro le scadenze).

Dobbiamo quindi provare il valore di capacità del contenitore $K := \max_j d_j = 12$ e indichiamo con K_j le capacità residue dei contenitori man mano che i lavori vengono inseriti. Inizialmente allora $K_j := K$, $\forall j$. Dapprima inseriamo il lavoro 1 al contenitore 1, quindi $K_1 := 12-4 = 8$. Il lavoro 2 non può essere assegnato al contenitore 1 e viene assegnato allora al contenitore 2, quindi $K_2 := 12-5 = 7$. Il lavoro 5 può essere assegnato al contenitore 1, e quindi $K_1 := 8-6 = 2$. Si assegna allora il lavoro 4 al contenitore 2 e quindi $K_2 := 7-4 = 3$. A questo punto non vi è più capacità residua disponibile per i rimanenti lavori e bisogna rivedere il valore iniziale di capacità. Non resta che il valore $K := 13$. Si ripetono le assegnazioni: lavoro 1 a 1 ($K_1 := 13 - 4 = 9$), lavoro 2 a 1 ($K_1 := 9 - 5 = 4$), lavoro 5 a 2 ($K_2 := 13 - 6 = 7$), lavoro 4 a 2 ($K_2 := 7 - 4 = 3$), lavoro 3 a 1 ($K_1 := 4 - 2 = 2$), lavoro 6 a 2 ($K_2 := 3 - 3 = 0$). Si ha allora l'assegnazione e schedulazione ($\{1 \to 2 \to 3\}$ per la macchina 1, $\{4 \to 5 \to 6\}$ per la macchina 2 (vedi figura 15.22) con tempi di completamento $C_1 = 4$, $C_2 = 4 + 5 = 9$, $C_3 = 9 + 2 = 11$, $C_4 = 4$, $C_5 = 4 + 6 = 10$, $C_6 = 10 + 3 = 13$, e valore di schedulazione $\max_j C_j + q_j = 13$. Pertanto i lavori non possono essere completati entro le scadenze e il valore di massimo ritardo è 1, causato dai lavori 2, 3 e 6. ∎

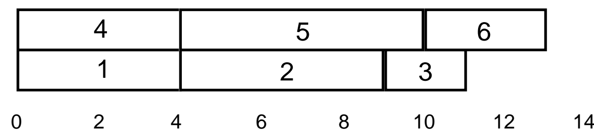

FIGURA 15.22

Un tipica classe di problemi in cui la disaggregazione in sottoproblemi di tipologie già note si presenta naturalmente, è costituita dai problemi di trasporto. A grandi linee le caratteristiche di questi problemi prevedono i seguenti aspetti: i) minimizzazione di percorsi con tappe intermedie, ii) vincoli di natura temporale sulle visite da effettuare durante il percorso, iii) vincoli di capacità dei veicoli.

A tale riguardo possiamo riconoscere le seguenti sottostrutture. Limitatamente ad i) il problema si presenta con le caratteristiche di un TSP o varianti del TSP, quando ad esempio i veicoli siano più di uno; qualora le tappe intermedie possano essere non effettuate allora interviene anche una struttura di knapsack; limitatamente a ii) il problema ha le caratteristiche di un problema di schedulazione; infine rispetto a iii) sono presenti le caratteristiche di un problema di bin packing.

È chiaro allora che è impensabile pensare di risolvere esattamente un problema di questa portata, specie quando i dati sono numerosi. Conviene invece disaggregare e poi ricomporre il problema. Una descrizione dei vari casi non può trovar luogo in questo testo. Ampio materiale bibliografico può essere reperito in Ball et al. [1995]. Ci limitiamo a fornire, abbastanza in dettaglio, un esempio di applicazione reale in cui ben si compendiano questi aspetti di disaggregazione in vari sottoproblemi.

15.11 Esempio. L'applicazione riguarda la gestione del trasporto urbano nella città di Montréal, Canada. A questo problema si sono a suo tempo dedicati, fra gli altri, dei ricercatori della locale università e una descrizione complessiva del problema si può trovare in Blais et al [1990]. Dettagli più specifici del modello matematico si trovano in Lessard et al [1981], Carraresi et al [1982], Rousseau et al [1985]. La descrizione che viene fatta qui del problema è volutamente semplificata in qualche aspetto per facilitarne la comprensione.

Il problema specifico riguarda l'assegnazione degli autisti di autobus alle varie linee con l'obiettivo di soddisfare la domanda di trasporto, di rispettare i vincoli sindacali e di sicurezza degli autisti e di minimizzare i costi.

Per quanto riguarda la domanda di trasporto, questa si presenta in modo molto variable durante la giornata, con un andamento bimodale che presenta un picco nella tarda mattinata e un altro nel tardo pomeriggio. Il modo più naturale per soddisfare la domanda è di variare in corrispondenza il numero di veicoli in circolazione sulle varie tratte. Generalmente un autobus lascia il deposito ad una certa ora, raggiunge un capolinea e percorre avanti e indietro la linea fino ad una ora fissata, dopo la quale, raggiunto il capolinea, fa ritorno in deposito. Durante questo servizio l'autobus può essere guidato da diversi autisti che si danno il cambio in posti predeterminati. In questa applicazione si è deciso di assegnare ad ogni veicolo sempre la medesima tratta. Eventualmente un autobus può rientrare in servizio sulla medesima tratta per soddisfare il picco pomeridiano.

Comunque l'aspetto relativo all'allocazione dei veicoli sulle tratte nei vari periodi della giornata non costituisce un problema, dato che la curva di domanda è la medesima per tutte le linee. La situazione può essere visualizzata come in figura 15.23 dove si vede la curva di domanda (numero di veicoli richiesti in funzione dell'ora della giornata) e il modo di 'coprirla' con l'allocazione di veicoli per vari intervalli di tempo, chiamati *blocchi* (i rettangoli in figura). Indichiamo con B l'insieme dei blocchi che vengono generati da questa prima assegnazione e con b un blocco generico.

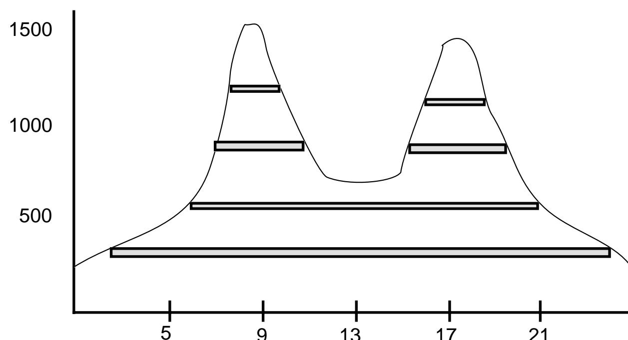

Figura 15.23

All'interno di un blocco sono poi definiti degli istanti di cambio. Questi non possono essere arbitrari in quanto un cambio d'autista può avvenire solo in certe località. Un autista effettua il suo servizio su un pezzo di un blocco, da un istante di cambio ad un altro istante di cambio, poi fa una pausa e successivamente effettua un servizio analogo al primo (eventualmente su un'altra tratta). Quindi un servizio può essere rappresentato da una quadrupla (i, j, h, k) dove i è l'istante di cambio dell'inizio del servizio, j è l'istante di cambio della fine della prima parte del servizio e analogamente per h e k. Quali siano le quadruple ammissibili viene definito dall'insieme delle regole lavorative e dalle norme di sicurezza. Pensiamo ora, solo in via teorica, di generare tutte le quadruple ammissibili. Sia Q l'insieme delle quadruple ammissibili e sia $q \in Q$ una quadrupla generica. Per ogni quadrupla si può calcolare il costo

 © 88-08-9059

c_q, che tiene conto del costo effettivo di utilizzare un autista sulla parte di blocco indicata dalla quadrupla. Il tempo può venire discretizzato facendo riferimento direttamente agli istanti di cambio. Sia T l'insieme degli istanti di cambio e sia t un generico di questi istanti. Con abuso di notazione si indicherà con t anche l'intervallo di tempo compreso fra t e l'istante successivo.

Astrattamente il problema può essere modellato come il seguente problema di copertura d'insiemi a costo minimo:

$$\begin{aligned} \min \quad & \sum_{q \in Q} c_q\, x_q \\ & \sum_{q \in Q} a_{btq}\, x_q \geq 1 \qquad b \in B, \quad t \in T \\ & x_q \in \mathbb{Z}_+ \end{aligned} \tag{15.1}$$

dove $a_{btq} \in \{0,1\}$ a seconda se la quadrupla q copre l'intervallo t del blocco b, e x_q indica quante volte viene usata la quadrupla q. In questa forma tuttavia il problema non si presta ad essere decomposto. Si tenga presente che le dimensioni sono enormi, in quanto vi sono migliaia di righe e milioni di colonne in (15.1). Una formulazione diversa, anche se altrettanto complessa, permetterà però di disaggregare il problema. Si definisca l'insieme di tutti i pezzi di quadrupla (i,j). Data una quadrupla (i,j,h,k), questa genera i pezzi (i,j) e (h,k). Sia P l'insieme di tutti i pezzi e sia p un pezzo generico. Sia $y_{bp} \in \{0,1\}$ una variabile binaria che indica se il pezzo p viene usato per coprire il blocco b. Sia $P^+(t)$ l'insieme dei pezzi che iniziano all'istante t, e sia $P^-(t)$ l'insieme dei pezzi che terminano all'istante t. Sia δ_{tb} una costante che vale 1 se il blocco b inizia in t, vale -1 se il blocco b finisce in t e vale 0 altrimenti. Si consideri allora il seguente problema (Carraresi et al [1982]):

$$\begin{aligned} \min \quad & \sum_{q \in Q} c_q\, x_q \\ & \sum_{q \ni p} x_q = \sum_{b \in B} y_{bp} \qquad\qquad p \in P \\ & \sum_{p \in P^+(t)} y_{bp} - \sum_{p \in P^-(t)} y_{bp} = \delta_{tb} \qquad t \in T,\ b \in B \\ & x_q \in \mathbb{Z}_+, \quad y_{bp} \in \{0,1\} \end{aligned} \tag{15.2}$$

La formulazione (15.2) evidenzia un aspetto tipico di un modello di flusso. Il secondo gruppo di vincoli è un vincolo di conservazione di flusso (negli istanti non di fine o inizio blocco) che indica come tutti i pezzi di servizio che terminano all'istante t corrispondono ad altrettanti veicoli che devono necessariamente proseguire, richiedendo quindi l'attivazione di altrettanti pezzi di servizio.

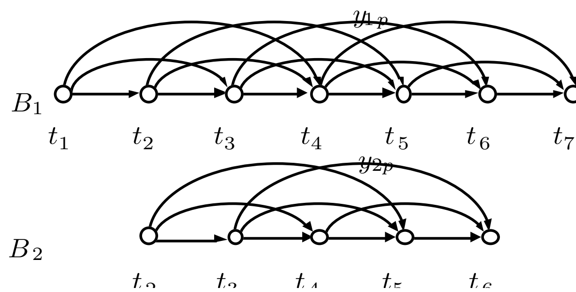

FIGURA 15.24

Si confronti la figura 15.24. Sono evidenziati due blocchi, B_1 e B_2. Il primo blocco copre l'intervallo temporale che va dall'istante t_1 all'istante t_7 e il secondo copre l'intervallo temporale da t_2 a t_7. Ogni istante viene identificato con un nodo di un grafo. Gli archi possono essere identificati con i pezzi associati ai blocchi. Il vincolo di flusso semplicemente stabilisce che, per ogni blocco, gli archi le cui variabili y_{bp} valgono 1 definiscono un cammino dal primo all'ultimo nodo del blocco. L'altro vincolo lega la somma di variabili y_{bp} fatta sui blocchi (in figura sono indicate due di tali variabili) alle quadruple che devono essere attivate.

Si noti ora che se le variabili x_q fossero fissate ad un valore noto, il restante problema nelle variabili y_{bp} diventerebbe un problema di tanti cammini minimi quanti sono i blocchi, con il vincolo particolare che la somma sui blocchi dei pezzi attivati debba essere uguale ad un valore fissato. Purtroppo questo vincolo non si può modellare all'interno delle reti di flusso. Vedremo fra poco come affrontarlo in modo euristico.

Se invece sono fissate le variabili y_{bp} (al valore $\bar{y}_{bp}$), il problema da risolvere è un accoppiamento. Infatti, essendo una quadrupla composta da due pezzi, possiamo pensare ad un grafo in cui ci sono $\sum_p a_p$ nodi con $a_p := \sum_b \bar{y}_{bp}$ e per ogni pezzo p vi sono a_p nodi. Tutti questi sono pezzi che devono essere inseriti in una quadrupla. Siccome una quadrupla mette assieme due pezzi, è chiaro che si può associare ad ogni quadrupla un arco e ciò che si cerca è un accoppiamento perfetto di minimo costo.

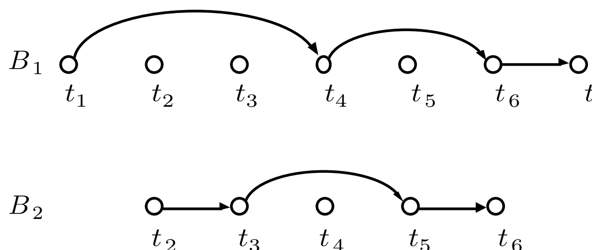

FIGURA 15.25

Ad esempio, con riferimento ai blocchi di figura 15.24, si supponga che la soluzione dei cammini minimi sia quella in figura 15.25 e che solo pezzi consecutivi siano proibiti nel formare una quadrupla. Allora con i 6 pezzi indicati si forma il grafo di figura 15.26 che ha un'unica soluzione di accoppiamento (figura 15.27). In generale ovviamente vi sono molte soluzioni ammissibili e si sceglie quella di minimo costo.

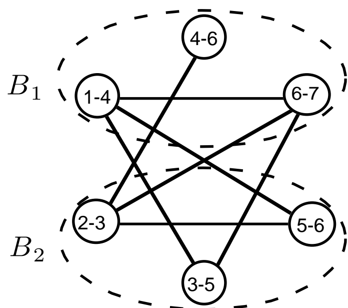

FIGURA 15.26

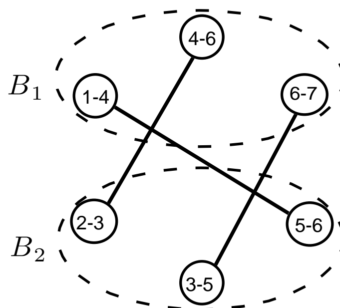

FIGURA 15.27

L'idea è quindi quella di risolvere un problema di minimo accoppiamento dopo che sono stati calcolati i valori delle variaibli y_{bp}. Il problema è ora quello di calcolare i cammini minimi con il vincolo aggiuntivo $\sum_{q \ni p} x_q = \sum_{b \in B} y_{bp}$ e di quale valore assegnare inizialmente alle variabili x_q. Questo secondo problema viene affrontato con un problema rilassato rispetto a (15.2). Sia B_t il numero di blocchi (cioè di veicoli) che coprono l'intervallo t e sia Q_t l'insieme di quadruple che coprono il medesimo intervallo. Il problema

$$\min \Big\{ \sum_{q \in Q} c_q \, x_q : \sum_{q \in Q_t} x_q \geq B_t, \; t \in T, \; x_q \in \mathbb{R}_+ \Big\} \tag{15.3}$$

è un doppio rilassamento di (15.1), perché è stata tolta l'interezza e anche perché i vincoli sono stati aggregati sommando sui blocchi. Per evitare che la soluzione di (15.3) sia poco significativa conviene aggiungere delle diseguaglianze valide per il problema originale. Ad esempio sia $B(s,t)$ il numero di blocchi di lunghezza minima (cioè non ulteriormente spezzabile) che iniziano in s e terminano in t (nelle figure una tale lunghezza minima è stata ridotta ad una unità temporale per esigenze di semplicità di disegno, ma in realtà la lunghezza minima di un pezzo è composta di parecchie unità temporali). Ovviamente si deve avere $\sum_{q \ni p} x_q \geq B(s,t)$, con $p = (s,t)$.

Analogamente, se indichiamo con B_t^0 il numero di blocchi che iniziano in t, si deve avere $\sum_{q \ni p} x_q \geq B_t^0$, con $p = (t,s)$ per ogni s. Aggiungendo questi due vincoli, il rilassamento (15.3) diventa molto affidabile. Ottenuta la soluzione $\bar{x}_q$ (frazionaria in generale) di (15.3) si tratta ora di trovare i cammini ammissibili per il vincolo $\sum_{q \ni p} \bar{x}_q =: d_p = \sum_{b \in B} y_{bp}$. Il problema non è facile e pertanto viene risolto a sua volta in modo euristico. Il problema potrebbe essere modellato come

$$\begin{aligned} \min \quad & \sum_{p \in P} (d_p - \sum_{b \in B} y_{bp})^2 \\ & \sum_{p \in P^+(t)} y_{bp} - \sum_{p \in P^-(t)} y_{bp} = \delta_{tb} \qquad t \in T, \; b \in B \\ & y_{bp} \in \{0, 1\} \end{aligned} \tag{15.4}$$

ma il vincolo quadratico lo rende difficile. Una semplificazione ulteriore consiste nel considerare in (15.4) un blocco alla volta, cioè fissando le variabili $y_{bp} =: \bar{y}_{bp}$ per $b \neq f$ e lasciando variabili y_{fp}. In questo modo l'obiettivo è lineare (ponendo $y_{fp}^2 = y_{fp}$ dato che sono variabili 0-1). Risolvendo quindi iterativamente molti problemi di cammino minimo si perviene ad una soluzione il cui valore non si scosta molto da quello di (15.4). La soluzione ottenuta in questa fase viene ancora sottoposta ad alcuni miglioramenti marginali. Si vedano gli articoli citati. ∎

15.4. Ricerca locale con memoria: tabu search

Il difetto principale della ricerca locale è l'impossibilità di evitare le trappole dovute a dei minimi locali discreti di bassa qualità. Per superare questa difficoltà bisogna quindi anche accettare cambiamenti nella soluzione corrente che peggiorino la soluzione, nella speranza che poi possano condurre verso soluzioni decisamente migliori. In quest'ottica sono state sviluppate negli ultimi anni due metodi, uno di tipo deterministico e l'altro di tipo stocastico, che si sono dimostrati validi strumenti di calcolo, soprattutto quando le dimensioni dei dati o la complessità della struttura impediscano un approccio 'esatto' nella ricerca di un ottimo.

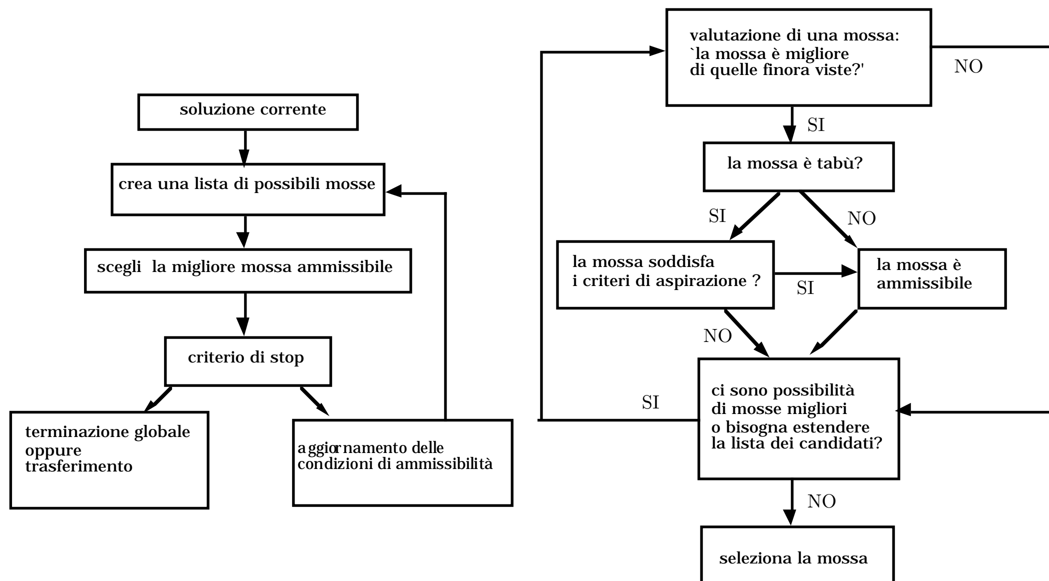

FIGURA 15.28 FIGURA 15.29

Se vengono accettati dei cambiamenti della soluzione, ovvero delle 'mosse', che comportano un peggioramento, bisogna certamente evitare che, fra le mosse successive, siano permesse delle mosse che fanno ritornare indietro alla soluzione iniziale. Mosse del genere vanno impedite e quindi il metodo che genera le soluzioni dovrebbe considerarle come dei 'tabù'. Questa considerazione ha dato il nome di *tabu search* alla tecnica che viene ora esposta. Di fatto, dato che si fa uso costante di una opportuna struttura di memoria nella scelta delle mosse successive, potremmo anche dare a questa tecnica il nome di *ricerca locale con memoria*. La tecnica è stata proposta da Glover [1989,1990a]. Per un più ampio riferimento sul metodo si veda Glover e Laguna [1997].

Va detto che, come ogni ricerca locale, non è un algoritmo esattamente definito che si può applicare senza bisogno di accorgimenti particolari. Si tratta invece di una serie di principi la cui applicazione ad un particolare problema richiede un'approfondita analisi, che va fatta necessariamente caso per caso. Il successo di una particolare applicazione di questa ricerca locale dipende molto anche da come si sono applicati i principi generali.

Nella figura 15.28 viene riepilogato lo schema generale della procedura e in figura 15.29 viene espansa la sottoprocedura relativa alla scelta della mossa. Bisogna distinguere fra struttura di memoria a breve e a lungo termine. Sulla memoria a breve termine si basano le operazioni indicate nelle due figure. La memoria a lungo termine viene usata quando si trova una soluzione che potrebbe essere quella finale, ma si ha ragione di credere di poter ulteriormente migliorarla con una strategia diversa. La memoria a lungo termine interviene appunto in questi casi con lo scopo di diversificare ed intensificare la ricerca.

15.12 ESEMPIO. Solo con degli esempi si può chiarire come le idee appena esposte si tramutano in un algoritmo. Si ribadisce comunque che gli algoritmi di tabu search possono essere molto diversi da problema a problema. L'esempio che riportiamo è basato su Glover [1990b]. Sia dato il grafo in figura 15.30 con costi sugli archi $c_1 = 18$, $c_2 = 6$, $c_3 = 12$, $c_4 = 8$, $c_5 = 2$, $c_6 = 9$, $c_7 = 0$. Si vuol trovare il minimo albero di supporto con i vincoli aggiuntivi che al più uno degli archi 2, 4 e 6 può essere presente nell'albero e che non ci

può essere l'arco 2 senza l'arco 1. Senza i vincoli aggiuntivi il problema sarebbe facilmente risolto con l'algoritmo greedy producendo la soluzione in figura 15.31.

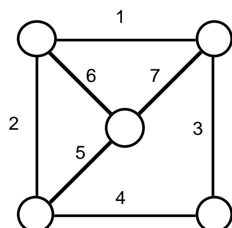

FIGURA 15.30

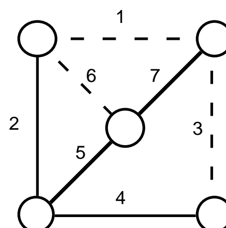

FIGURA 15.31

La soluzione non è però ammissibile per il nuovo vincolo. Si proceda allora identificando innanzitutto una struttura d'intorno. Ad esempio si può decidere che soluzioni adiacenti si ottengono aggiungendo una corda e togliendo un arco del circuito creato. Dobbiamo anche pesare la violazione del vincolo d'ammissibilità. Si assegni un costo pari a 50 ad ogni violazione. Quindi la soluzione iniziale ha un costo pari a 16+100=116.

Si inizia quindi la ricerca locale. La prima mossa consiste semplicemente nella scelta della migliore soluzione dell'intorno. Con i dati forniti la migliore soluzione si ottiene aggiungendo l'arco 1 e togliendo l'arco 2, con il che i vincoli sono rispettati con un costo pari a 28.

A questo punto entra in gioco il meccanismo dei tabù. Per evitare di ricadere nella soluzione precedente si introduce l'arco 1 nella lista tabù. Questa caratterizzazione bloccherà qualsiasi possibilità di togliere l'arco 1 dall'albero in un certo numero di mosse successive. La successiva mossa fa entrare nell'albero l'arco 3 e togliere l'arco 4. Questa operazione alza il costo da 28 a 32, ma viene accettata comunque. Ora anche l'arco 3 diventa tabù. Possiamo assumere che la lista degli elementi tabù contenga solo due elementi, per cui in questo momento la lista è piena. Se ci saranno ulteriori elementi da far entrare, questi faranno uscire gli elementi entrati da maggior tempo.

Nella nuova situazione la miglior mossa è quella di far entrare l'arco 6 e di far uscire l'arco 1. L'arco 1 è tabù e quindi in condizioni normali la mossa non sarebbe ammessa, però si tratta anche di una mossa che soddisfa ampiamente i criteri di aspirazione e quindi viene accettata (si confronti con il diagramma di flusso). L'arco 6 entra nella lista tabù con priorità più alta rispetto all'arco 3. L'arco 1 sarebbe comunque uscito dalla lista tabù in questa fase. Il nuovo albero ha costo 23 ed è un minimo locale.

La miglior mossa, anche se peggiorativa, sarebbe quella di scambiare gli archi 1 e 6. Questa mossa non viene accettata, non tanto perché sia peggiorativa (altrimenti il metodo sarebbe una semplice ricerca locale), ma perché la mossa è stata dichiarata tabù e infatti costituerebbe un ritorno alla soluzione precedente. ∎

Il meccanismo visto può convenientemente essere arricchito permettendo la compresenza di diverse liste tabù basate su idee alternative e, analogamente, su diversi criteri di aspirazione. Questo ovviamente aumenta la flessibilità e l'adattabilità del metodo. La lunghezza delle liste tabù non dovrebbe essere alta altrimenti non si potrebbe parlare di memoria a breve termine. Empiricamente si è visto che una lunghezza intorno al numero 7 funziona bene.

Quando si è trovata una soluzione sfruttando la memoria a breve termine si invocano altri principi per vedere se si può migliorare la situazione. Fra le strategie più semplici (e brutali) si può citare il cambiamento casuale della soluzione. Leggermente più elaborata è la strategia di variare la funzione obiettivo. Molto più complesse sono infine le tecniche che tengono conto della statistica con cui si è sviluppata l'iterazione.

Tutto questo insieme di regole di ricerca, tipicamente adattative, genera normalmente delle buone soluzioni, purché ovviamente il progettisita abbia saputo sfruttare a fondo le caratteristiche strutturali di un problema e a questo fine la conoscenza della struttura matematica più generale di un problema indubbiamente giova. Importante è notare che, come ogni altra euristica, la ricerca a tabù può essere usata in modo complementare con tecniche esatte quali branch-and-bound e branch-and-cut.

15.5. Ricerca locale stocastica: simulated annealing

La tecnica nota come *annichilazione simulata* (*simulated annealing*) è stata proposta da Kirkpatrick et al. [1983] per risolvere problemi difficili di ottimizzazione specialmente quando le dimensioni del problema rendono impossibile ogni risoluzione esatta con metodi noti. Si consulti Aarts e Korst [1989] per una trattazione approfondita del metodo.

L'annichilazione simulata prende origine da concetti di statistica meccanica, in particolare dalla simulazione numerica di un processo di annichilazione, e dall'analogia che si riesce a stabilire fra gli stati fisici di un sistema meccanico e le soluzioni di un problema di ottimizzazione combinatoria, e fra l'energia degli stati fisici e il valore della funzione obiettivo per le soluzioni del problema di ottimizzazione. Quindi, in base all'analogia, raffreddare il sistema meccanico è equivalente a minimizzare la funzione obiettivo.

Un risultato di meccanica statistica afferma che, se $f(x)$ è l'energia dello stato x e T è la temperatura del sistema, allora il sistema fluttua casualmente da stato a stato con una probabilità di visita di ogni stato pari a $e^{-f(x)/K\,T}$, dove K è la costante di Boltzmann. Al diminuire della temperatura il sistema si trova sempre più frequentemente in stati di bassa energia.

Per simulare il processo di annichilazione, gli stati (o equivalentemente le soluzioni ammissibili del problema di ottimizzazione) sono generati casualmente con le seguenti regole:
– un nuovo stato y è generato, a partire dallo stato corrente x, con probabilità q_{xy};
– il nuovo stato y viene accettato se ha energia minore di quella dello stato corrente x;
– il nuovo stato y viene accettato con probabilità $e^{(f(x)-f(y))/K\,T}$ se ha energia maggiore di quella dello stato corrente x.

Il meccanismo è molto simile alla ricerca locale. Se si definisce

$$q_{xy} := \begin{cases} 1/|N(x)| & \text{se } y \in N(x) \\ 0 & \text{altrimenti} \end{cases} \tag{15.5}$$

allora viene presa in modo equiprobabile una soluzione nell'intorno di x. La ricerca locale accetta solo soluzioni migliori. Con il nuovo meccanismo si accettano anche soluzioni peggiori, però con probabilità decrescente all'aumentare della differenza fra $f(x)$ e $f(y)$. La probabilità di accettare una soluzione peggiore dipende anche dal parametro T, la 'temperatura', e cala al diminuire della temperatura. Per $T = 0$ il processo perde le caratteristiche stocastiche e corrisponde esattamente alla ricerca locale, dato che soluzioni peggiori vengono accettate con probabilità 0 (ci sarebbe un problema per soluzioni uguali, ma la cosa non ha importanza perché il valore $T = 0$ non si usa mai).

Le regole di passaggio da una soluzione all'altra definiscono una catena di Markov sull'insieme delle soluzioni con matrice di transizione

$$p_{xy} := \begin{cases} q_{xy}\ \min\left\{1\,;\, e^{(f(x)-f(y))/K\,T}\right\} & \text{se } y \neq x \\ \displaystyle\sum_{z \in N^+(x)} q_{xz}\,(1 - e^{(f(x)-f(z))/K\,T}) & \text{se } y = x \end{cases}$$

dove $N^+(x) := \{y \in N(x) : f(y) > f(x)\}$. Dimostriamo ora il seguente fatto:

15.13 Teorema. *Sia la catena irriducibile e sia q_{xy} simmetrica. Allora la probabilità limite della catena vale*

$$\pi_x = C\, e^{-(f(x)/K\,T)}$$

con C fattore di normalizzazione.

Dimostrazione. Basta far vedere che la probabilità limite soddisfa le equazioni dettagliate di bilancio

$$\pi_x\, p_{xy} = \pi_y\, p_{yx} \qquad \forall x, y$$

Infatti si può scrivere (supponendo $f(y) \geq f(x)$)

$$\pi_y\, p_{yx} = \pi_y\, q_{yx} = \pi_y\, q_{xy} = C\, e^{-(f(y)/K\,T)}\, q_{xy}\, e^{(f(x)/K\,T)}\, e^{-(f(x)/K\,T)} = \pi_x\, p_{xy}$$

∎

L'ipotesi di simmetria $q_{xy} = q_{yx}$ non è sempre soddisfatta nelle applicazioni. Se q viene definita come in (15.5) allora l'ipotesi è soddisfatta quando gli intorni abbiano tutti la stessa grandezza e $y \in N(x) \iff x \in N(y)$.

In base al teorema lo stato a energia più bassa, cioè l'ottimo, ha la più alta probabilità di visita. Se $T \downarrow 0$ la probabilità limite dell'ottimo tende a 1. Però per $T = 0$ la catena non è irriducibile. Di fondamentale importanza è la velocità con cui la catena converge verso la probabilità limite. Il fattore di convergenza, che è dato dal secondo autovalore della matrice di transizione, tende a 1 al tendere di T a 0. Quindi non si può usare direttamente un valore basso di T. Anche se la probabilità di visita dell'ottimo è elevata, si tenga comunque presente che questo valore è un valore limite che potrebbe essere raggiunto solo dopo un numero di iterazioni molto alto.

Queste considerazioni suggeriscono il seguente approccio per ottenere, se non l'ottimo, almeno soluzioni non troppo lontane dall'ottimo: iterare il processo di generazione di nuove soluzioni e allo stesso tempo diminuire lentamente la temperatura T, cosicché il sistema tende abbastanza velocemente alla distribuzione limite, che però varia lentamente tendendo alla distribuzione concentrata sull'ottimo. Più alta è la temperatura e più alta è anche la possibilità di uscire fuori da un minimo locale, ma più numerose anche sono le cattive soluzioni generate.

Il successo del metodo dipende molto da come i valori di T vengono fatti tendere a 0. La regola di diminuzione della temperatura viene chiamata *annealing schedule* e, per ottenere dei buoni risultati, andrebbe determinata con molta cura per ogni singola classe di problemi. Una regola comune abbastanza buona è quella di porre $T := 1/\log k$, con k indice di iterazione.

Va detto che per ottenere buone soluzioni bisogna generare un numero elevatissimo di soluzioni. Del resto il metodo si applica quando non ci sono altri metodi di risoluzione e quindi c'è da attendersi una certa lentezza. Il metodo è stato applicato a molti problemi di ottimizzazione combinatoria. Le applicazioni migliori riguardano il TSP con grafi fino a 20000 nodi (Bonomi and Lutton [1984]) e alcuni problemi VLSI (Gertsbakh [1989]).

La risoluzione della medesima istanza di TSP dell'esempio 15.1 con il metodo di annichilazione simulata produce i tre circuiti in figura 15.0 di lunghezze 75.44, 73.88 e 74.12 rispettivamente.

15.6. Algoritmi genetici e reti neurali

Aggiungiamo qualche breve considerazione su due tecniche che hanno avuto buon successo in alcuni campi applicativi (soprattutto le reti neurali) ma che non sembra finora abbiano dato un contributo significativo nell'ottimizzazione. Queste tecniche hanno un loro fascino

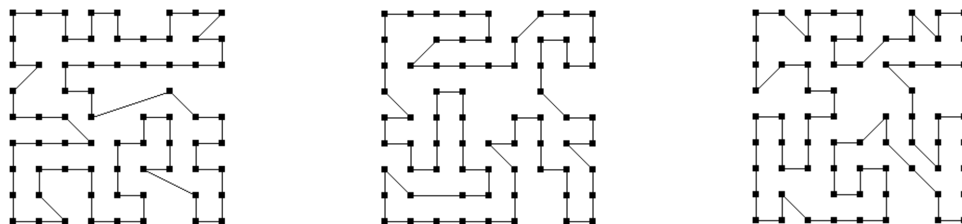

FIGURA 15.32

perché si basano su analogie di tipo biologico (in un caso l'evoluzione di una specie e nell'altro il meccanismo di percezione neurale). Però, al di là della curiosità per l'analogia, non vi sono motivi per credere che la difficoltà strutturale di problemi **NP**-difficili debba essere catturata da metodi di lontanissima origine.

Gli algoritmi genetici si basano sull'idea che una soluzione può essere paragonata ad un individuo di una specie biologica. Un insieme di soluzioni costituisce quindi una popolazione. Il valore di una soluzione corrisponde al grado di adattabilità all'ambiente dell'individuo. Come in natura due individui generano figli, così due soluzioni generano altre soluzioni che ereditano, parzialmente, le caratteristiche dei genitori. Quando una popolazione (di soluzioni) ha prodotto una nuova generazione di soluzioni, si opera una drastica selezione e si mantengono in vita solo le soluzioni migliori. Il processo poi continua finché il valore globale della popolazione non migliora più. Una breve rassegna di queste tecniche si trova esposta in Colorni et al. [1994].

Questa idea è stata applicata al TSP, che in qualche modo si adatta in modo naturale all'idea di generazione. Infatti un circuito si può spezzare in due, e quindi due soluzioni generano una nuova soluzione spezzandosi in due e fondendo assieme gli spezzoni dell'uno e dell'altro circuito. Il metodo rimane tuttavia ben lontano dalle prestazioni di algoritmi esatti per il TSP, per non parlare delle euristiche di ricerca locale.

Più complesso sarebbe descrivere come costruire una rete neurale che debba risolvere un problema di ottimizzazione. Ci sono tuttavia delle difficoltà di principio che rendono problematica l'applicazione delle reti neurali ai problemi **NP**-difficili. Bruck e Goodman [1990] hanno sottolineato come l'esistenza di una rete neurale che risolva un problema di ottimizzazione e che sia di dimensione polinomiale nell'istanza implicherebbe **NP** = **co-NP**, anche lasciando alla rete neurale un tempo esponenziale per raggiungere uno stato di stabilità. Il punto della dimostrazione è che costruire una rete neurale per una data istanza è equivalente a risolvere l'istanza, e se la rete è di grandezza polinomiale la costruzione è polinomiale. Il chiaro vantaggio delle reti neurali nelle varie applicazioni in cui sono state usate con successo è dovuto al loro intrinseco parallelismo. Purtroppo però i problemi **NP**-difficili richiedono un parallelismo di tipo esponenziale.

15.7. Approssimazione

Concludiamo il capitolo con un accenno agli algoritmi approssimati. In realtà l'argomento meriterebbe molto più di un cenno, data l'importanza teorica dell'approssimazione nella caratterizzazione della complessità computazionale di molti problemi ed in generale nella comprensione delle relazioni fra le varie classi di complessità. Tuttavia una esposizione significativa della teoria dell'approssimazione richiederebbe uno spazio eccessivo per questo testo.

Inoltre, tranne alcuni casi, gli algoritmi approssimati non forniscono soluzioni necessariamente buone dal punto di vista pratico. Il lettore interessato alla teoria dell'approssimazione è invitato a consultare la raccolta Hochbaum [1997]. Ci limitiamo a fornire le definizioni fondamentali e a citare alcuni risultati

Innanzitutto si tenga presente che in problemi di ottimizzazione combinatoria il termine approssimazione non ha un significato generico di 'sufficientemente vicino', ma un preciso significato tecnico dato dalla seguente definizione.

15.14 DEFINIZIONE. *Un algoritmo δ-approssimato per un problema X è un algoritmo polinomiale che, per ogni istanza i di X, fornisce una soluzione di valore $w(i)$ tale che*

$$1 \leq \frac{w(i)}{v(i)} \leq \delta \qquad \forall i \in X$$

se X è un problema di minimizzazione, e

$$1 \leq \frac{v(i)}{w(i)} \leq \delta \qquad \forall i \in X$$

se X è un problema di massimizzazione, e dove $v(i) > 0$ è il valore ottimo dell'istanza i e δ viene detto fattore di approssimazione. ∎

Quindi l'errore relativo è sempre limitato da $\delta - 1$. Si noti che per problemi di massimo la definizione fa riferimento ai reciproci $1/w(i)$ e $1/v(i)$. Questa scelta è infatti più significativa. Se, ad esempio, tutte le istanze di un problema di massimizzazione hanno valori di funzione obiettivo non negativi e un algoritmo fornisce sistematicamente una soluzione di valore nullo, ha senso considerare tale errore uguale ad infinito piuttosto che uguale ad uno.

Può essere sorprendente scoprire che per alcuni problemi non può esistere alcun algoritmo δ-approssimato per nessun valore di δ non importa quanto grande, a meno che $\mathbf{P} = \mathbf{NP}$. Il TSP appartiene a questa categoria. Infatti se esistesse un tale algoritmo allora si potrebbero assegnare lunghezze 1 agli archi di un grafo $G = (N, E)$ e lunghezze $\delta\, n$ agli archi complementari e risolvere un problema di TSP sul grafo completo. Se G è hamiltoniano, un circuito ottimo è hamiltoniano in G ed ha lunghezza n, mentre un circuito non hamiltoniano ha lunghezza almeno $(n-1) + n\,\delta$. Siccome $(n - 1 + n\,\delta)/n = \delta + (n-1)/n > \delta$, l'algoritmo approssimato fornirà sempre un circuito hamiltoniano. Se invece il grafo non è hamiltoniano l'algoritmo approssimato (come un qualsiasi altro algoritmo) fornirà un circuito non hamiltoniano. Quindi si sarebbe in grado di decidere in tempo polinomiale se un grafo è hamiltoniano oppure no.

D'altra parte una classe ristretta di problemi di TSP in cui valga la diseguaglianza triangolare, cioè $c_{ij} \leq c_{ik} + c_{kj}$, $\forall i, j, k$, ammette un famoso algoritmo $\frac{3}{2}$-approssimato dovuto a Christofides [1976] (si veda anche Papadimitriou e Steiglitz [1982], oppure Bern e Eppstein [1997]). Per la stessa classe di problemi si può dimostrare che in generale le euristiche d'inserzione (esempio 15.6) forniscono un valore che è $1 + \lceil \log_2 n \rceil$ volte il valore ottimo (non sono algoritmi approssimati, perché il fattore d'approssimazione cresce con le dimensioni dell'istanza), mentre l'inserzione più vicina è un genuino algoritmo 2-approssimato. Inoltre un minimo locale per $\lambda = 4$ (scambio di due archi) ha un valore che è $O(\log n)$ il valore ottimo.

15.15 ESERCIZIO. L'algoritmo di Christofides calcola dapprima il minimo albero di supporto T. Siano N_d i nodi di grado dispari in T. Poi calcola l'accoppiamento di costo minimo M su $(N_d, E(N_d))$. Il grafo $T \cup M$ è euleriano. Trovato un circuito euleriano su $T \cup M$, lo si trasforma in hamiltoniano saltando tutti i nodi ripetuti. Dimostrare che l'algoritmo è $\frac{3}{2}$-approssimato. ∎

Le euristiche per il bin-packing viste nell'esempio 15.8 sono algoritmi approssimati (il fattore d'approssimazione non è costante ma tende da sopra ad un valore finito, che viene definito come fattore d'approssimazione asintotico). Anche il knapsack è approssimabile e per di più con un fattore d'approssimazione arbitrario. Sia dato da risolvere il problema $\max\{c(J) : a(J) \le K,\ J \subset N\}$. Si normalizzino ora i coefficienti della funzione obiettivo come

$$\hat{c}_i := \lfloor c_i\, M \rfloor \quad \forall i \in N$$

dove $M := n^2/(C\,\varepsilon)$ e $\varepsilon > 0$ è una fissata costante arbitraria. Si noti che qualsiasi siano i coefficienti dell'obiettivo originario, il valore massimo dell'obiettivo normalizzato è $C\,M = n^2/\varepsilon$. Allora la complessità del problema rivisto è $O(n\,C\,M) = O(n^3/\varepsilon)$ ed è una complessità *polinomiale* per ε costante. Ovviamente la nuova soluzione $\hat{J}$ ha un valore $\hat{v}$ (rispetto all'obiettivo originario) in generale diverso dal valore v^* dell'ottimo J^* originario, tuttavia la differenza di valore può essere limitata. Infatti

$$\hat{v} = \sum_{j \in \hat{J}} c_j \ge \frac{1}{M} \sum_{j \in \hat{J}} \lfloor c_j\, M \rfloor \ge \frac{1}{M} \sum_{j \in J^*} \lfloor c_j\, M \rfloor \ge \sum_{j \in J^*} c_j - \frac{n}{M}$$

da cui $v^* - \hat{v} \le n/M$. Inoltre $\hat{v} \ge c_i$, $\forall i$, ovvero $\hat{v} \ge C/n$, e quindi

$$\frac{v^*}{\hat{v}} - 1 \le \frac{n^2}{C\,M} = \varepsilon$$

e l'errore può essere reso arbitrariamente piccolo (ma a scapito di una complessità crescente). Quando ciò sia possibile si parla di schema di approssimazione.

15.16 Definizione. *Una famiglia di algoritmi* $(1+\varepsilon)$*-approssimati per un problema X viene detta: 1)* schema di approssimazione polinomiale *se ogni algoritmo della famiglia è polinomiale rispetto alla grandezza dell'istanza per ogni fissato* $\varepsilon > 0$*; 2)* schema di approssimazione completamente polinomiale *se ogni algoritmo della famiglia è polinomiale rispetto alla grandezza dell'istanza e a* $1/\varepsilon$ *(non alla codifica di* $1/\varepsilon$*).* ∎

Quindi il problema del knapsack ammette uno schema di approssimazione completamente polinomiale.