Capitolo 14

Programmazione lineare intera: enumerazione implicita

Il metodo più ampiamente usato per risolvere problemi di ottimizzazione combinatoria e di programmazione intera particolarmente difficili consiste in un'enumerazione di tutti i casi possibili. Siccome normalmente il numero di casi possibili cresce in modo esponenziale con le dimensioni dell'istanza, l'enumerazione deve procedere in modo ragionato, scartando a priori tutti quei casi che, in modo certo, non possono produrre l'ottimo. Se l'enumerazione è progettata bene, il tempo di calcolo può essere abbattuto a valori accettabili, anche se, in presenza di problemi **NP**-difficili, non dobbiamo aspettarci tempi di risoluzione polinomiali.

Forniremo ora i principi generali di questo metodo di enumerazione, definita implicita proprio perché i sottoinsiemi di soluzioni scartati dall'enumerazione vengono, per così dire, enumerati implicitamente. Questo metodo viene anche indicato con il termine *branch-and-bound* per i motivi che verranno ora precisati.

L'enumerazione implicita si può rappresentare con un cosiddetto *albero di ricerca*. Un albero di ricerca è un albero con radice e la radice è associata all'insieme di tutte le soluzioni ammissibili. I nodi successori di un dato nodo sono associati a sottoinsiemi propri dell'insieme associato al nodo predecessore e ne formano una partizione. Le foglie dell'albero corrispondono a singole soluzioni oppure all'insieme vuoto.

L'insieme ammissibile può essere suddiviso in molti modi alternativi. Quale di questi sia conveniente dipende dalla struttura del problema e dalle operazioni che si intendono fare sul problema. La suddivisione corrisponde alla decisione sul *branching* delle soluzioni ammissibili.

Il secondo aspetto cruciale consiste nell'elaborazione di strategie che permettano di tagliare sottoalberi dell'albero di ricerca. Queste strategie si basano sulla ricerca di limitazioni inferiori dei minimi dei sottoproblemi associati ad ogni nodo. Questo tipo di decisione corrisponde al *bounding* del problema. Una limitazione inferiore è utile quando è più grande del valore di una soluzione ammissibile nota. Infatti in questo caso il valore ottimo del sottoproblema non può essere migliore della soluzione nota e quindi una enumerazione delle soluzioni ammissibili del sottoproblema è inutile.

Le tecniche poliedrali viste nei capitoli precedenti servono appunto a generare limitazioni inferiori il più alte possibili. L'uso combinato delle tecniche di enumerazione impicita con quelle poliedrali ha permesso di produrre algoritmi particolarmente veloci anche per problemi notoriamente difficili. Queste tecniche combinate prendono il nome di *branch-and-cut*.

14.1. Alberi di ricerca: regole di suddivisione

Consideriamo il seguente problema generale di ottimizzazione:

$$\begin{aligned} v := \min\ & f(x) \\ & x \in F \subset E \end{aligned} \tag{14.1}$$

dove F è l'insieme finito delle soluzioni ammissibili (F può anche essere vuoto) e l'insieme ambiente E non è necessariamente finito. Supponiamo che F sia implicitamente descritto da un algoritmo che, per ogni $x \in E$, verifica in tempo polinomiale se $x \in F$ oppure no. Siamo inoltre interessati al seguente problema di ammissibilità:

$$\text{dato } F \subset E \text{ si trovi } x \in F \text{ oppure si decida che } F = \emptyset \tag{14.2}$$

Per risolvere questi problemi si associa all'insieme E un *albero di ricerca* secondo le seguenti regole:

- ogni nodo dell'albero è (associato a) qualche sottoinsieme di E;
- la radice è E;
- i successori E'' di un nodo E' sono sottoinsiemi di E' tali che i sottoinsiemi $E'' \cap F$ non vuoti formano una partizione di $E' \cap F$.
- i nodi E' tali che E' è vuoto oppure è dato da un singolo elemento non hanno successori. Altri nodi senza successori possono essere definiti in base a regole particolari.

Se E è finito, l'albero ha necessariamente un numero finito di nodi. Altrimenti, la regola di enumerazione deve essere progettata in modo da produrre un numero finito di nodi. Ad ogni nodo dell'albero è inoltre associato il problema (14.1) (oppure il problema (14.2)) ristretto al sottoinsieme E' relativo al nodo. Come vedremo, il metodo di enumerazione implicita cerca di risolvere tali problemi nei vari nodi dell'albero. Se in un nodo si riesce a risolvere il problema associato al nodo, non è ovviamente necessario risolvere il problema in nessuno dei nodi successori del nodo, che quindi non vengono generati esplicitamente. Questa circostanza favorevole non si verifica soltanto quando si riesce a risolvere il sottoproblema del nodo, ma anche quando si ottiene informazione sufficiente a concludere che l'ottimo non è incluso nel sottoinsieme del nodo. Questo aspetto verrà approfondito alla fine della sezione.

L'idea di fondo del metodo è, come si vede, quella di dividere un insieme troppo grande in sottoinsiemi più piccoli e, sperabilmente, più trattabili. Per tale motivo viene anche indicato, anche se in modo etimologicamente improprio, come *divide et impera*.

14.1 Esempio. Si consideri l'insieme

$$F := \{x \in \{0,1\}^n : A\,x \leq b\} \subset E := \{x \in [0,1]^n : A\,x \leq b\}$$

Un possibile albero di ricerca è un albero binario di profondità al massimo n. La radice è a profondità zero. Una possibile strategia di suddivisione potrebbe associare ai due discendenti di un nodo a profondità $k-1$ le scelte $x_k := 0$ e $x_k := 1$ rispettivamente. Quindi la radice è l'insieme E, mentre i due successori della radice sono rispettivamente gli insiemi $\{x \in [0,1]^n : A\,x \leq b,\ x_1 = 0\}$ e $\{x \in [0,1]^n : A\,x \leq b,\ x_1 = 1\}$. Ogni cammino dalla radice ad un nodo a profondità k corrisponde ad un definito assegnamento di valori 0-1 alle variabili $x_1, \ldots, x_k$. Se l'insieme E' corrispondente è vuoto non vi sono successori. ∎

14.2 Esercizio. Si descriva esplicitamente l'albero di ricerca associato al problema di knapsack dell'esempio 1.72 (si veda anche l'esempio 5.15). ∎

14.3 Esempio. Si consideri un problema di programmazione lineare intera con insieme ammissibile definito da

$$F := \{x \in \mathbb{Z}^n : A\,x \le b,\, x \ge 0\} \subset E := \{x \in \mathbb{R}^n : A\,x \le b,\, x \ge 0\}$$

Usando regole di suddivisione che impongono alternativamente il vincolo $x_i \le z$ oppure $x_i \ge z+1$ con z intero, l'insieme E viene successivamente ridotto. Siccome minimizzare su $E' \cap F$ è equivalente a minimizzare su E' se l'inviluppo convesso di $E' \cap F$ coincide con E', possiamo stabilire la regola che un nodo non ha successori quando $\text{conv}(E' \cap F) = E'$.

Sia ad esempio dato

$$E := \left\{ x \in \mathbb{R}^2 : \begin{array}{c} 2\,x_1 + 6\,x_2 \le 15 \\ x_1 + x_2 \le 4 \\ x_1 \ge 0 \quad x_2 \ge 0 \end{array} \right\} \qquad F := E \cap \mathbb{Z}^2$$

In figura 14.1 è raffigurato l'albero di ricerca. I vincoli sugli archi dell'albero rappresentano i vincoli aggiuntivi rispetto a quelli già presenti nel cammino dalla radice al nodo. Sono inoltre raffigurati per ogni nodo gli insiemi E' (evidenziati in grigio), E e l'inviluppo convesso di F. Gli insiemi dei nodi 5 e 7 sono vuoti. Nei nodi 1, 4 e 8 si verifica $\text{conv}(E' \cap F) = E'$. ∎

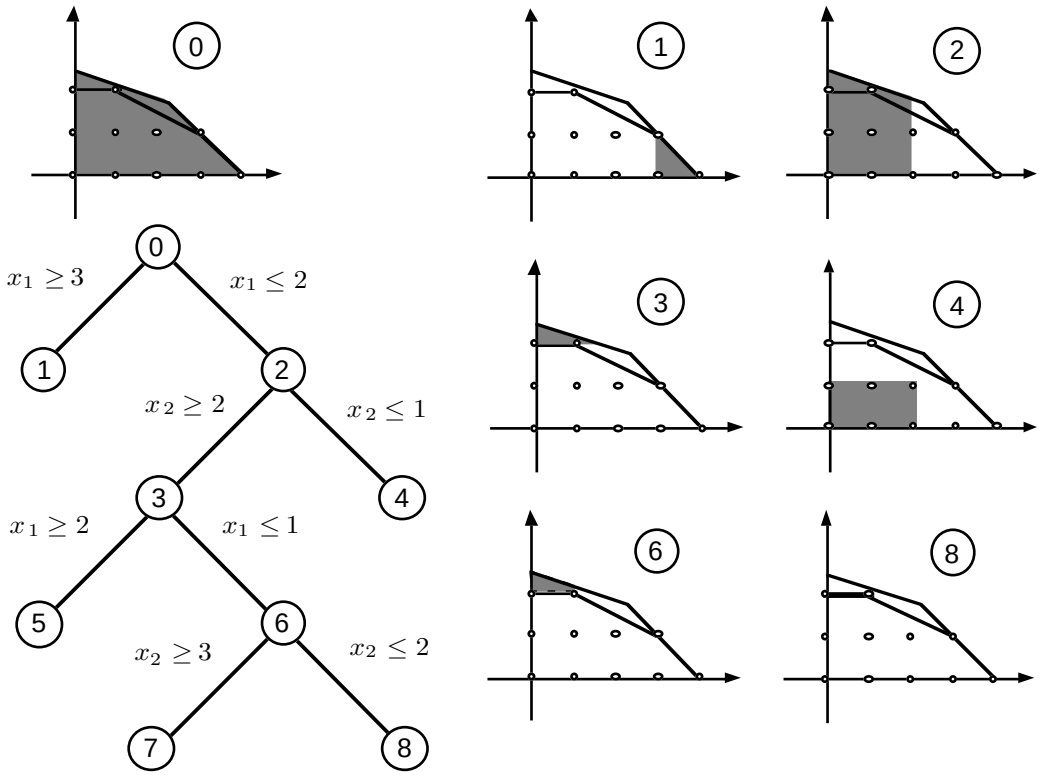

Figura 14.1

14.4 Esempio. Si consideri il TSP asimmetrico. Dato un grafo orientato completo $G = (N, A)$ sia E l'insieme di tutti i circuiti orientati e F l'insieme di tutti i circuiti hamiltoniani

orientati. I nodi E' dell'albero di ricerca sono insiemi di circuiti con la proprietà che alcuni archi sono obbligati ad essere presenti in ogni circuito di E' e altri archi sono esclusi da tutti i circuiti di E'. Siano $A^+(E')$ e $A^-(E')$ questi sottoinsiemi di archi.

La radice corrisponde all'insieme di tutti i circuiti, o alternativamente corrisponde alla scelta $A^+(E) = A^-(E) = \emptyset$ dato che nessun arco è a priori vincolato ad essere presente o assente in ogni circuito. Sia E' un nodo dell'albero di ricerca. Se C è un circuito non hamiltoniano allora nessun circuito hamiltoniano può contenere simultaneamente tutti gli archi di C. Siano $e_1, \ldots, e_p$ tali archi. L'obiettivo è di ottenere successori che non contengano a turno uno degli archi del circuito. Per evitare che soluzioni ammissibili siano presenti in più successori è opportuno vincolare ulteriormente i vari successori, secondo il seguente ragionamento: ogni circuito hamiltoniano o non contiene e_1 oppure lo contiene. Non vi sono altre alternative. A loro volta i circuiti che contengono e_1, o non contengono e_2 oppure lo contengono. Continuando, i circuiti che contengono e_1 ed e_2, o non contengono e_3 oppure lo contengono. Si può continuare fino ad esaurimento degli archi del circuito C osservando però che l'obbligo di appartenenza non può essere esteso contemporaneamente a tutti gli archi di C. Allora i successori $E_1, \ldots, E_p$ di E' sono gli insiemi di circuiti vincolati da

$$
\begin{array}{ll}
A^+(E_1) := A^+(E') & A^-(E_1) := A^-(E') \cup \{e_1\} \\
A^+(E_2) := A^+(E') \cup \{e_1\} & A^-(E_2) := A^-(E') \cup \{e_2\} \\
A^+(E_3) := A^+(E') \cup \{e_1, e_2\} & A^-(E_2) := A^-(E') \cup \{e_3\} \\
\ldots\ldots\ldots\ldots & \ldots\ldots\ldots\ldots \\
A^+(E_p) := A^+(E') \cup \{e_1, e_2, \ldots, e_{p-1}\} & A^-(E_p) := A^-(E') \cup \{e_p\}
\end{array}
$$

Potrebbe succedere che qualche definizione di A^+ ed A^- sia inconsistente se ad esempio si richiede ad un arco di essere presente in entrambi gli insiemi. In questo caso tale successore non viene generato. Come si vede questa regola di suddivisione richiede la definizione di un circuito C ad ogni nodo dell'albero di ricerca. Vedremo più avanti un esempio di questa tecnica. ∎

14.5 Esempio. Nei problemi di schedulazione ad una macchina vi sono n lavori che devono essere eseguiti uno alla volta su una macchina e bisogna stabilire l'istante d'inizio delle varie esecuzioni, cioè la schedulazione. Possono essere presenti vincoli di varia natura che limitano le schedulazioni ammissibili. Inoltre può essere definita una funzione obiettivo che valuta la qualità della schedulazione. La difficoltà di questo tipo di problemi consiste nel trovare la sequenza ottima fra le $n!$ sequenze (incluse quelle non ammissibili). Una volta definita la sequenza, trovare la schedulazione (cioè definire anche gli istanti di tempo) non rappresenta un problema, in quanto questa operazione è spesso risolvibile con la programmazione lineare oppure con la programmazione dinamica.

L'insieme E può essere allora identificato con l'insieme delle permutazioni di n elementi. Ad ogni nodo dell'albero di ricerca sono associati due successori. In uno dei successori un assegnato lavoro i deve sempre precedere un altro assegnato lavoro j, mentre nell'altro successore i deve sempre seguire j.

Con questo tipo di suddivisione l'albero ha una profondità che può variare da un minimo di $(n-1)$ ad un massimo di $n\,(n-1)/2$. Si veda in figura 14.2 un esempio con $n = 3$.

Si noti che anche il problema del TSP potrebbe essere modellato in questo modo. Infatti si tratta di determinare la sequenza di visita dei nodi a partire da un nodo arbitrariamente fissato come iniziale. La scelta di una particolare suddivisione dell'insieme ammissibile dipende da quanto i sottoproblemi nei singoli nodi assomiglino al problema originario, nonostante i vincoli imposti per suddividere l'insieme. I metodi tipicamente usati per il TSP si riferiscono

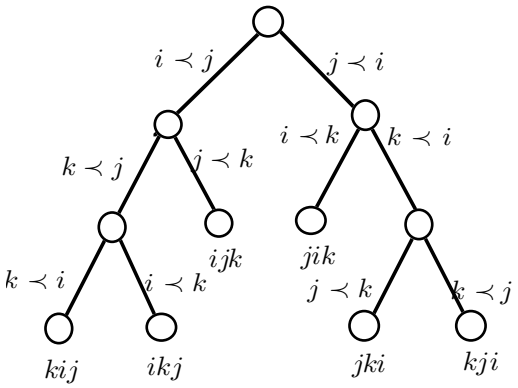

FIGURA 14.2

alla struttura di circuito di un grafo e meno a quella di sequenza e quindi è più adatta una suddivisione del tipo di quella indicata nell'esempio 14.4. Per la schedulazione può essere più indicata invece una suddivisione come quella in figura 14.2. ∎

14.6 ESEMPIO. Il principio di suddividere l'insieme ammissibile secondo una partizione e non una copertura è molto importante, in quanto permette di evitare sprechi computazionali. In alcuni problemi tuttavia la partizione può essere solo apparente, perché vi possono essere soluzioni diverse in base al modello impiegato, ma equivalenti dal punto di vista del problema originario.

Si consideri ad esempi il problema del bin packing. Questo problema può essere modellato come un problema di programmazione lineare 0-1 nel seguente modo:

$$\begin{aligned} \min \quad & \sum_{i=1}^{m} y_i \\ & \sum_{j=1}^{n} a_j x_{ij} \ \leq C\, y_i \qquad i := 1, \ldots, m, \quad j := 1, \ldots, n \\ & \sum_{i=1}^{m} \quad x_{ij} \ = 1 \qquad\quad j := 1, \ldots, n \\ & x_{ij} \in \{0,1\}\,,\ y_i \in \{0,1\} \end{aligned} \tag{14.3}$$

dove a_j è la dimensione dell'oggetto j-esimo, C è la capacità di ogni contenitore e m è una limitazione superiore (calcolabile con una qualsiasi euristica) al numero di contenitori necessari. Le variabili x_{ij} indicano se l'oggetto j viene assegnato al contenitore i. I vincoli esprimono il fatto che l'assegnazione di un qualsiasi oggetto ad un contenitore attiva la corrispondente variabile y che indica appunto l'utilizzo di un contenitore. Inoltre ogni oggetto deve essere assegnato ad un contenitore, come risulta dal vincolo di assegnamento.

Trattandosi di variabili binarie è naturale utilizzare un albero di ricerca che pone le variabili binarie alternativamente ai valori 0 o 1. Il fatto critico tuttavia non risiede nell'albero di ricerca quanto piuttosto nel modello (14.3). Per ogni assegnamento di oggetti a contenitori vi sono in (14.3) $m!$ soluzioni equivalenti che si ottengono permutando arbitrariamente i contenitori fra loro. Quindi un albero di ricerca indotto da (14.3) contiene un numero di foglie che è $m!$ volte più grande del numero strettamente necessario. Si deve dedurre che (14.3) è un cattivo modello del problema.

Per ovviare a questo inconveniente si pensi che l'oggetto numero 1, ad esempio, può essere assegnato al contenitore 1 senza perdita di generalità. Fatta questa scelta, l'oggetto 2 può trovarsi insieme all'oggetto 1 oppure no. In questo secondo caso possiamo arbitrariamente assegnarlo al contenitore 2 senza pregiudicare alcuna soluzione ammissibile. Ragionando in questo modo l'oggetto 3 può essere assegnato ai contenitori 1, 2 o 3, ma non ad altri. Basta allora modificare nel modello (14.3) il vincolo di assegnamento in

$$\sum_{i=1}^{\min\{j,m\}} x_{ij} = 1 \qquad j := 1, \ldots, n$$

Con questo semplice accorgimento l'albero di ricerca ha dimensioni più ridotte con notevole risparmio computazionale. ∎

C'è da tener presente che, anche se le linee generali della regola di suddivisione sono decise a priori e sono le medesime per tutte le istanze di un determinato problema, le scelte particolari della suddivisione vengono decise solo nel momento in cui si devono generare i successori di un certo nodo. Infatti la visita dell'albero di decisione produce anche dell'informazione aggiuntiva sul problema che può essere utilmente sfruttata nel momento di operare la suddivisione.

14.2. Alberi di ricerca: limitazioni inferiori

Affrontiamo ora il problema di come operare sull'albero di decisione. Per quanto detto, ad ogni nodo E' dell'albero di ricerca è associato il sottoproblema

$$\begin{aligned} v' := \min\ & f(x) \\ & x \in F' := F \cap E' \end{aligned} \tag{14.4}$$

Tuttavia, se il problema originario è **NP**-difficile, è **NP**-difficile, nella maggior parte dei casi anche il sottoproblema (14.4). Anziché risolvere (14.4) si cerca invece una sua limitazione inferiore $\bar{v}'$.

Un ulteriore ingrediente del metodo di enumerazione implicita consiste nella possibilità di generare, nei vari nodi dell'albero, 'buone' soluzioni ammissibili, secondo metodi dipendenti dal problema in esame, ma in ogni caso polinomiali. Viene definita come *incombente* la migliore soluzione ammissibile generata. Sia $\hat{v}$ il suo valore.

La risoluzione del problema (14.1) viene effettuata visitando un sottoinsieme di nodi dell'albero di ricerca, dove l'espressione 'visita di un nodo' significa identificazione di una limitazione inferiore al sottoproblema associato al nodo ed eventuale generazione di una soluzione ammissibile. Durante la risoluzione alcuni nodi sono identificati come *attivi* ed è fra questi che viene scelto il successivo nodo da visitare. Inizialmente l'unico nodo attivo è la radice. La risoluzione termina quando non vi sono più nodi attivi.

La visita di un nodo termina con il confronto fra la limitazione inferiore calcolata $\bar{v}'$ e il valore $\hat{v}$ dell'incombente e, a seconda dell'esito del confronto, si procede secondo una delle due alternative:

- $\bar{v}' \geq \hat{v}$: siccome $v' \geq \bar{v}' \geq \hat{v}$ si può concludere che l'ottimo (globale) non si trova in E'; allora si può fare a meno di visitare il sottoalbero di E' e si possono visitare altri nodi attivi senza generarne di nuovi;
- $\bar{v}' < \hat{v}$: in questo caso non si può trarre nessuna conclusione sul valore ottimo di (14.4), per cui si è costretti a visitare i successori di E' che vengono resi quindi attivi.

Nel primo caso il numero di nodi attivi viene diminuito di uno, mentre nel secondo caso viene aumentato almeno di uno. Si noti che se E'' è successore di E' si ha necessariamente $\bar{v}' \leq v' \leq v''$ e quindi, anche senza visitare E'', è già disponibile una limitazione inferiore per il sottoproblema E'', ereditata dal nodo generatore. Questa circostanza può essere sfruttata nel caso in cui venga generata una nuova soluzione ammissibile migliore dell'incombente. Allora il confronto fra il nuovo incombente e la limitazione inferiore ereditata può direttamente eliminare il nodo attivo (ma non ancora visitato) dalla lista dei nodi attivi.

Durante la visita dell'albero conviene monitorare la differenza fra $\hat{v}$ e $\tilde{v}$, definito come la minore fra le limitazioni inferiori dei nodi attivi. Necessariamente $\tilde{v} \leq v \leq \hat{v}$ e quindi $\hat{v} - \tilde{v}$ fornisce una limitazione superiore dell'errore assoluto dell'incombente. Se tale errore viene valutato come accettabile, si può terminare anticipatamente la visita dell'albero e accettare l'incombente al posto dell'ottimo. In pratica questa strategia è molto usata perché permette di ottenere apprezzabili vantaggi di tempo, ed è particolarmente raccomandata quando la funzione obiettivo è costruita in modo artificioso per ottenere soluzioni 'buone' secondo vari criteri alternativi e quindi l'ottimo matematico non ha un preciso significato di ottimalità reale. Si noti che la visita dell'albero termina regolarmente solo quando $\hat{v} = \tilde{v}$.

Non vi sono regole a priori su quale sia la migliore strategia di scelta del nodo fra quelli attivi. Fra quelle più comuni citiamo la *ricerca in profondità* e la *ricerca sulla limitazione*.

Nella ricerca in profondità si seleziona per la visita l'ultimo nodo attivato. In questo modo si perviene rapidamente alle foglie dell'albero, con possibilità quindi di generare velocemente soluzioni ammissibili. Questo metodo è raccomandato quando sia difficile generare soluzioni ammissibili con metodi euristici ad hoc. Un altro vantaggio del metodo consiste nel basso numero di nodi attivi, che risulta limitato dalla profondità dell'albero, moltiplicata per il numero di successori di ogni nodo meno uno. Allora se l'albero di ricerca è binario, come nell'esempio 14.1, vi sono al massimo n nodi attivi. Il fatto di avere pochi nodi attivi è molto importante quando, come nel caso della programmazione lineare intera, l'informazione da memorizzare per ogni nodo attivo è molto grande.

Nella ricerca sulla limitazione si seleziona il nodo che ha la limitazione inferiore più bassa. Questa scelta ha l'effetto di aumentare subito il valore $\tilde{v}$ e quindi innanzitutto di ridurre lo scarto $\hat{v} - \tilde{v}$. Inoltre, se le limitazioni inferiori sono vagamente correlate con i rispettivi valori ottimi, si sceglie un sottoinsieme con la più alta probabilità di contenere l'ottimo.

Nel caso si debba risolvere il problema di ammissibilità (14.2), tutto il meccanismo delineato di limitazioni inferiori e superiori non è ovviamente applicabile. Si tratta semplicemente di visitare l'albero e interrompere la ricerca non appena si trova una soluzione ammissibile. È chiaro che se l'istanza non è ammissibile è necessario visitare tutto l'albero di ricerca con pesanti conseguenze computazionali.

Anche se è vero che un problema di ammissibilità può essere trasformato in un problema di ottimizzazione, questo trucco non paga. Se l'istanza è ammissibile, una limitazione inferiore è già nota ed è data dal valore ottimo che è stato costruito per identificare le istanze ammissibili. Se l'istanza non è ammissibile, un valore di $\tilde{v}$ più alto di tale valore normalmente è disponibile solo alla fine della visita dell'albero. La conclusione è che decidere se l'istanza di un problema **NP**-completo è di tipo 'sì' o di tipo 'no' costituisce veramente un problema ostico. Si pensi che, a differenza di un problema di ottimizzazione, non c'è la possibilità di accontentarsi di una soluzione abbastanza buona. Le soluzioni possono essere solo ammissibili o non ammissibili!

Un'analisi del tempo medio di visita di un albero di ricerca è stato effettuata da Stone e Sipala [1986] prendendo in esame il seguente modello computazionale. Sia dato un albero binario completo e per ogni nodo i suoi due immediati successori non diventano attivi con probabilità costante p. La visita si effettua in profondità. La visita dell'albero termina

quando non ci sono più nodi attivi, oppure quando si raggiunge una foglia qualsiasi alla profondità massima. Questo modello computazionale si adatta meglio agli alberi derivati da problemi di ammissibilità piuttosto che da problemi di ottimizzazione, ma rimane significativo anche per questi ultimi.

In modo forse soprendente il tempo di visita medio, per un valore costante di p, cresce linearmente con la profondità dell'albero e non quindi con il numero dei nodi, che invece cresce esponenzialmente con la profondità dell'albero. Normalmente la profondità dell'albero è un parametro polinomiale rispetto ai dati dell'istanza. La dipendenza del tempo medio da p è una curva a campana centrata nel valore $p = 1/2$, dove si ha il massimo tempo medio di visita. Per valori di p che tendono a 1 vengono generati sempre meno nodi attivi e quindi il tempo medio decresce. Per valori di p che tendono a 0, vengono generati con sempre maggior probabilità tutti i discendenti di un nodo facilitando quindi il raggiungimento di una foglia.

Il punto cruciale di questo modello è l'ipotesi di probabilità costante in ogni nodo dell'albero. In un'enumerazione implicita di un albero non vengono generati nodi attivi se si è ottenuta informazione sufficiente per concludere che la visita del sottoalbero è inutile. Siccome l'informazione di questo tipo (ad esempio le semplici limitazioni inferiori) cresce con la profondità dell'albero, anche la probabilità p cresce con la profondità, ed è questa caratteristica a far aumentare in modo esponenziale i tempi di visita.

Quindi il caso di probabilità costante dovrebbe essere considerato come un caso limite, non ottenibile in problemi normali. Tuttavia il principio di far crescere la probabilità il meno possibile da un livello all'altro dell'albero dovrebbe guidare il progetto dell'albero di ricerca e degli schemi di suddivisione. Si noti che l'idea di avere limitazioni inferiori il più alte possibili rientra in quest'ottica.

14.3. Rilassamento d'interezza

La strategia più ampiamente usata per produrre limitazioni inferiori consiste nella risoluzione di un problema strettamente collegato a quello originale (14.1).

14.7 Definizione. *Il problema*

$$\begin{aligned} \bar{v} := \min\ & \bar{f}(x) \\ & x \in \bar{F} \subset \bar{E} \end{aligned} \tag{14.5}$$

viene chiamato rilassamento *del problema (14.1) se*

1) $F \subset \bar{F}$ *e* $E \subset \bar{E}$

2) $\bar{f}(x) \leq f(x)$, $\forall x \in F$

∎

Allora ovviamente $\bar{v} \leq v$ e il problema (14.5) fornisce una limitazione inferiore alla soluzione ottima.

Naturalmente il problema rilassato deve essere molto più facile di quello originale. Fra i vari tipi di rilassamento quello probabilmente più usato è quello che toglie il vincolo d'interezza in un problema di programmazione lineare intera e viene appunto chiamato *rilassamento di interezza*. Ad esempio dovendo risolvere

$$\begin{aligned} \min\quad & c\,x \\ & A\,x \geq \mathbf{1} \\ & x \in \{0,1\}^n \end{aligned} \tag{14.6}$$

si risolve invece ad ogni nodo dell'albero di ricerca il problema rilassato con il vincolo $x \in \{0,1\}^n$ sostituito da $x \in [0,1]^n$. La suddivisione viene effettuata in base alla soluzione rilassata. I successori di un certo nodo E' sono i nodi corrispondenti alle scelte $x'_i = 0$ e $x'_i = 1$ con x'_i una qualsiasi componente frazionaria della soluzione rilassata x'.

I sottoproblemi successori di un sottoproblema sono più piccoli, perché alcune variabili hanno un valore fissato a priori e quindi di fatto non ci sono. Risolvere ex-novo questi sottoproblemi sarebbe troppo dispendioso computazionalmente e si preferisce risolverli a partire dalla soluzione ottima del problema che li ha generati. Tuttavia l'ottimo del problema generatore non è ammissibile per i sottoproblemi (a causa dei vincoli aggiuntivi), per cui si deve operare con il simplesso duale (vedi sezione 7.11). Questa scorciatoia computazionale ha un prezzo però, perché bisogna memorizzare l'inversa della matrice di base dei nodi non più attivi ma con successori attivi. Solo adottando una strategia di ricerca in profondità c'è la garanzia che il numero di matrici da mantenere in memoria sia al più pari alla profondità dell'albero. Con altre strategie tale numero potrebbe crescere fino a saturare la memoria e interrompere la risoluzione (fatto tutt'altro che infrequente con istanze grandi).

Per quanto il rilassamento d'interezza sia molto semplice da applicare e proprio per questo sia presente nella maggior parte dei pacchetti di programmazione lineare intera, pure è bene che venga usato con una chiara consapevolezza delle sue possibilità e dei suoi limiti. La qualità della limitazione inferiore che si ottiene dipende grandemente dalla struttura del problema e quindi del modello che si decide di adottare. Vedremo ora tre casi diversi.

14.8 Esempio. Si consideri il seguente problema ad una macchina: sono assegnati tre lavori di durata $p_1 := 3$, $p_2 := 2$ e $p_3 := 4$. I lavori dovrebbero essere completati preferibilmente agli istanti $d_1 := 6$, $d_2 := 7$, $d_3 := 6$. Se il tempo di completamento C_j è diverso da d_j si incorre in una penalità pari a $W_j := \max\left\{w_j^-\,(d_j - C_j)\;;\; w_j^+\,(C_j - d_j)\right\}$ dove sono assegnati i seguenti pesi: $w_1^- := 2$, $w_1^+ := 3$, $w_2^- := 1$, $w_2^+ := 4$, $w_3^- := 3$, $w_3^+ := 3$. Bisogna minimizzare $\sum_j W_j$.

Modelliamo il problema in questo modo: sia J l'insieme dei lavori e sia $T := \left\{1, 2, \ldots, \hat{t}\right\}$ l'insieme degli istanti di tempo (il problema ha dati interi e quindi il tempo può essere discretizzato), dove $\hat{t}$ è un valore abbastanza alto da garantire che la soluzione ottima abbia i lavori completati entro $\hat{t}$. Per per ogni lavoro $j \in J$ e per ogni $t \in T$ sia definita la seguente variabile 0-1:

$$x(j,t) = \begin{cases} 1 & \text{se il lavoro } j \text{ è completato all'istante } t \\ 0 & \text{altrimenti} \end{cases}$$

Possiamo quindi associare ad ogni variabile $x(j,t)$ il coefficiente di costo

$$c(j,t) := \max\left\{w_j^-\,(d_j - t_j)\;;\; w_j^+\,(t_j - d_j)\right\}$$

Le variabili $x(j,t)$ devono essere soggette a due tipi di vincoli: bisogna imporre che i lavori siano eseguiti e quindi si deve avere

$$\sum_{t=p_j}^{\hat{t}} x(j,t) = 1 \qquad j \in J \tag{14.7}$$

e si deve evitare che lavori diversi vengano eseguiti negli stessi istanti, e quindi si deve imporre

$$\sum_{j \in J} \sum_{s=t}^{t+p_j-1} x(j,t) \leq 1 \qquad t \in T \tag{14.8}$$

La matrice definita da (14.8) ha una struttura molto particolare. Con i dati dell'esempio è:

$$\left(\begin{array}{cccccccccccccccccccccccccccccccccccccccccccccccc}
1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\
1&1&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0\\
1&1&1&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0\\
0&1&1&1&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0\\
0&0&1&1&1&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0\\
0&0&0&1&1&1&0&0&0&0&1&1&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0\\
0&0&0&0&1&1&1&0&0&0&1&1&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0\\
0&0&0&0&0&1&1&1&0&0&1&1&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&1&1&1&0&1&1&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&1&1&1&1&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&1&1&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&0&1&1&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0&1&1&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&1&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&1&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&1&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&0&0&0&0&0&0
\end{array}\right) \tag{14.9}$$

e ogni colonna rappresenta una possibile schedulazione di un particolare lavoro (gli 1 corrispondono agli istanti di tempo in cui la macchina è occupata con il lavoro). Una matrice di 0 e 1 in cui in ogni colonna gli 1 sono tutti adiacenti fra loro è totalmente unimodulare (facile dimostrazione che si può lasciare come esercizio). Tuttavia la matrice del problema non è totalmente unimodulare in quanto bisogna anche aggiungere il vincolo di assegnamento (14.7) che è nell'esempio

$$\left(\begin{array}{cccccccccccccccccccccccccccccccccccccccccccccccc}
1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&1&1&1&1&1&1&1&1
\end{array}\right) \tag{14.10}$$

La matrice che si ottiene componendo (14.9) con (14.10) non è più totalmente unimodulare, però mantiene molto di tale struttura, e quindi è lecito attendersi che le limitazioni inferiori che si ottengono con un rilassamento di interezza siano abbastanza buone. Così è infatti. Risolvendo il rilassamento d'interezza con i dati dell'esempio si ottiene

$$\begin{array}{lll}
\bar{x}(1,3) = 1/2 & \bar{x}(2,2) = 1/2 & \bar{x}(3,10) = 1/2\\
\bar{x}(1,6) = 1/2 & \bar{x}(2,8) = 1/2 & \bar{x}(3,6) = 1/2
\end{array}$$

e tutte le altre variabili uguali a zero. Il valore di questa soluzione è $\bar{v} = 13.5$. Come si vede in figura 14.3 il risultato è una combinazione convessa di due schedulazioni impossibili. Mentre le due schedulazioni sono impossibili, la loro combinazione convessa è ammissibile per i vincoli.

Come limitazione inferiore si può usare il valore $\lceil \bar{v} \rceil$ dato che il valore della soluzione ottima deve essere intero. Si scelga di suddividere il problema come $x(1,3) := 0$ e $x(1,3) := 1$. Se si visita per primo il nodo relativo a $x(1,3) := 0$ si ottiene la soluzione intera $x(1,9) = 1$, $x(2,2) = 1$, $x(3,6) = 1$, di valore 14 (figura 14.4) e pertanto ottima. In questo caso non serve visitare l'altro nodo perché ha la stessa limitazione inferiore.

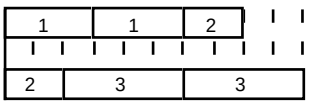

FIGURA 14.3

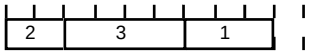

FIGURA 14.4

Se invece si visita prima il nodo $x(1,3) := 1$ si ottiene la soluzione non intera $x(2,5) = 1/2$, $x(2,7) = 1/2$, $x(3,7) = 1/2$, $x(3,11) = 1/2$, di valore 16, che genera i due nodi $x(2,5) := 0$ e $x(2,5) := 1$. Adottando la strategia di ricerca sulla limitazione, si dovrebbe visitare il nodo $x(1,3) := 0$ e si otterrebbe l'ottimo, con conseguente eliminazione dai nodi attivi dei due nodi appena generati, la cui limitazione inferiore è superiore a 14. Adottando invece la strategia di ricerca in profondità, si dovrebbe imporre $x(2,5) := 0$ oppure $x(2,5) := 1$

(oltre a $x(1,3) := 1$). Si ottengono nel primo caso le soluzioni $x(3,7) = 1$ e $x(2,9) = 1$ e nel secondo caso $x(3,9) = 1$, ambedue di valore 17. L'albero di ricerca completo è pertanto quello raffigurato in figura 14.5 dove sono indicate le scelte di suddivisione e i valori delle limitazioni inferiori.

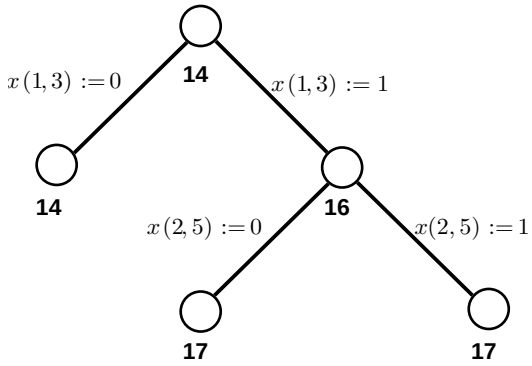

FIGURA 14.5

Il vantaggio di questo modello consiste appunto nella bontà delle limitazioni inferiori. Un altro vantaggio è dato dalla flessibilità del modello rispetto a diverse formulazioni della funzione obiettivo. Infatti questa può essere una qualsiasi funzione $\sum_{j \in J} f_j(C_j)$ con f_j funzioni arbitrarie. Se si vuole anche limitare $\max_{j \in J} f_j(C_j)$ al di sotto di una soglia prefissata si può tener conto di questa esigenza limitando le colonne a quelle per cui tale vincolo è rispettato. Lo svantaggio del modello è che la matrice dei vincoli tende a diventare molto grande all'aumentare dei dati. ∎

14.9 ESEMPIO. Si riprenda l'esempio precedente che ora modelliamo in modo diverso. Sia t_j una variabile che indica il tempo di completamento del lavoro j e sia x_{ij} una variabile 0-1 che vale 1 se i precede j e 0 se j precede i. Per legare il vincolo di precedenza ai valori della variabile si imponga per ogni coppia (i, j) la seguente coppia di vincoli

$$t_i \leq t_j - p_j + M\,(1 - x_{ij})$$
$$t_j \leq t_i - p_i + M\,x_{ij}$$

dove M è un valore sufficientemente grande da rendere ridondante il vincolo quando il suo coefficiente vale 1. Questo trucco viene indicato normalmente come *metodo big-M*. La funzione obiettivo si esprime come $\sum_{j \in J} W_j$ con W_j vincolato come

$$W_j \geq w_j^-\,(d_j - t_j) \qquad\qquad W_j \geq w_j^+\,(t_j - d_j)$$

Usando il valore $M := 20$ si ottiene come risultato del rilassamento d'interezza:

$$x_{12} = 0.2 \qquad x_{13} = 0.15 \qquad x_{23} = 0.05$$

e $t_1 = 6$, $t_2 = 7$, $t_3 = 6$, con valore ottimo quindi 0! Una tale limitazione inferiore è praticamente inutilizzabile.

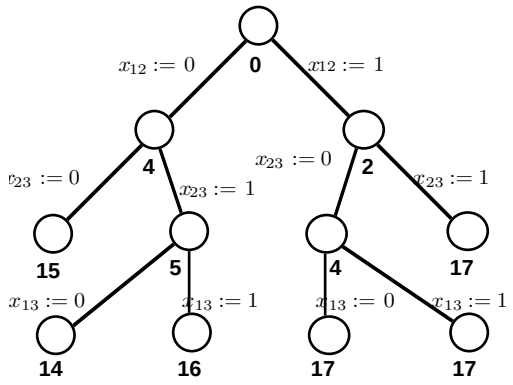

FIGURA 14.6

Preseguendo nella risoluzione del problema si genera un albero di ricerca (del tutto simile a quello dell'esempio 14.5) raffigurato in figura 14.6. Si noti che la limitazione inferiore cresce a valori sufficientemente elevati solo in corrispondenza delle foglie dell'albero, cioè quando i valori imposti di x_{ij} generano un ordine totale. Di fatto è necessaria una visita completa dell'albero. I tempi di calcolo diventano proibitivi già con piccole istanze (ad esempio 10 lavori). Quindi questo modello è altamente sconsigliabile e la ragione è dovuta alla pessima qualità delle limitazioni inferiori e, a sua volta, la ragione di questo fatto risiede nella presenza della costante M, che rende ridondante il vincolo non solo se il valore del coefficiente di M è 1, ma anche per quasi tutti i valori compresi fra 0 e 1. È chiaro allora che rilassare il vincolo d'interezza è quasi equivalente ad abolire i vincoli di precedenza fra i lavori! ∎

14.10 ESEMPIO. Il problema della *localizzazione di impianti con vincoli di capacità* (*capacitated plant location problem*) prevede di determinare i siti ottimi dove costruire degli impianti (ad esempio centrali elettriche) all'interno di un insieme specificato di siti minimizzando i costi fissi e i costi di gestione e supponendo soddisfatta la domanda dell'utenza.

I dati del problema prevedono: un insieme di m possibili siti, un insieme di n clienti, per ogni sito i un costo fisso d_i in cui si incorre solo se l'impianto è costruito e una prevista capacità C_i dell'impianto, per ogni cliente j una domanda di D_j unità di servizio e per ogni coppia cliente j e impianto i un costo di servizio c_{ij} per unità di servizio erogata. Allora il problema si può modellare come

$$\begin{aligned}
\min \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}\, x_{ij} + \sum_{i=1}^{m} d_i\, y_i \\
& \sum_{j=1}^{n} x_{ij} \le C_i\, y_i \qquad i := 1, \ldots, m \\
& \sum_{i=1}^{m} x_{ij} = D_j \qquad j := 1, \ldots, n \\
& x_{ij} \ge 0, \qquad y_i \in \{0, 1\}
\end{aligned}$$

dove x_{ij} è la quantità di servizio che verrà fornita al cliente j dall'impianto i e y_i è una variabile 0-1 che indica se l'impianto viene o non viene costruito.

Si suppongano i seguenti dati: 6 possibili siti ($m := 6$), 10 clienti ($n = 10$), domande $D := (\,100, 50, 40, 80, 100, 60, 60, 40, 100, 20\,)$, valori di capacità e costi fissi dati da

	1	*2*	*3*	*4*	*5*	*6*
C	250	150	300	200	500	600
d	900	700	1300	800	2300	3000

e costi correnti dati da

	1	*2*	*3*	*4*	*5*	*6*
1	5	8	10	10	5	7
2	2	8	6	5	4	3
3	10	10	5	2	10	3
4	3	6	4	3	7	2
5	9	6	7	7	5	5
6	10	20	5	8	9	10
7	6	8	4	8	9	3
8	7	8	4	4	6	3
9	5	5	5	9	2	4
10	8	8	5	5	2	10

In figura 14.7 si vede l'albero di ricerca, dove per ogni nodo il successore di sinistra corrisponde alla scelta $y_i := 0$ e quello di destra alla scelta $y_i := 1$. Sono evidenziati i nodi che sono attivi per una ricerca in profondità in cui si sceglie prima il successore di sinistra. I nodi sono numerati secondo quest'ordine. Si è scelta come variabile da suddividere quella in cui il valore è più vicino ad 1/2. Questa scelta si basa sull'idea che sono queste le variabili per le quali c'è più incertezza e che quindi è meglio fissare subito. I risultati dei calcoli fatti nei nodi dell'albero sono riportati nella tabella di figura 14.7, dove i valori di y_i evidenziati in grassetto si riferiscono alle scelte imposte, mentre i valori in grassetto di $\bar{v}$ evidenziano una soluzione intera.

14.11 Esercizio. Quale sarebbe la successione di visita dei nodi nell'esempio precedente se si effettuasse una ricerca secondo la limitazione? Si visiterebbero meno nodi o forse se ne dovrebbero aggiungere altri? ∎

14.4. Rilassamento lagrangiano

Un altro tipo di rilassamento molto usato è il cosiddetto *rilassamento lagrangiano*. Il valore della funzione duale $L(u)$ (per una opportuna formulazione del problema duale) costituisce una limitazione inferiore all'ottimo per la proprietà di dualità debole. La migliore limitazione inferiore di questo tipo si ottiene ovviamente in corrispondenza dell'ottimo duale, però, se è computazionalmente dispendioso trovare l'ottimo duale, è più conveniente valutare $L(u)$ per valori di u abbastanza vicini all'ottimo duale. È in questo spirito che i metodi di quasi ascesa, delineati nel capitolo 4, diventano utili.

Va subito detto che il rilassamento lagrangiano può non dare i risultati sperati proprio quando la facilità di calcolo della funzione duale ne consiglierebbe l'uso.

	$\bar{v}$	y_1	y_2	y_3	y_4	y_5	y_6
0	5020	0.92	0	0.2	0.2	0.44	0.166
1	5340	1	0	0.2	0.3	**0**	0.466
2	5426.66	1	0.666	0.4	0.9	**0**	**0**
3	$+\infty$	*	*	**0**	*	**0**	**0**
4	5500	1	0.4	**1**	0.3	**0**	**0**
5	5520	1	**0**	**1**	0.5	**0**	**0**
6	$+\infty$	*	**0**	**1**	**0**	**0**	**0**
7	5560	0.6	**0**	**1**	**1**	**0**	**0**
8	$+\infty$	**0**	**0**	**1**	**1**	**0**	**0**
9	**5920**	**1**	**0**	**1**	**1**	**0**	**0**
10	5626	0.64	**1**	**1**	0.2	**0**	**0**
11	**6120**	**0**	**1**	**1**	1	**0**	**0**
12	**5840**	**1**	**1**	**1**	0	**0**	**0**
13	6070	0	0	0.2	0.1	**0**	**1**
14	5768	0.32	0	0	0.2	**1**	0.1
15	5800	**0**	0	0	0.2	**1**	0.233
16	5820	**0**	0	0.2	0.45	**1**	**0**
17	5970	**0**	0	0.5	**0**	**1**	**0**
18	6020	**0**	0	0.2	**1**	**1**	**0**
19	**7810**	**0**	0	0	0	**1**	**1**
20	6160	**1**	0	0	0.2	**1**	0

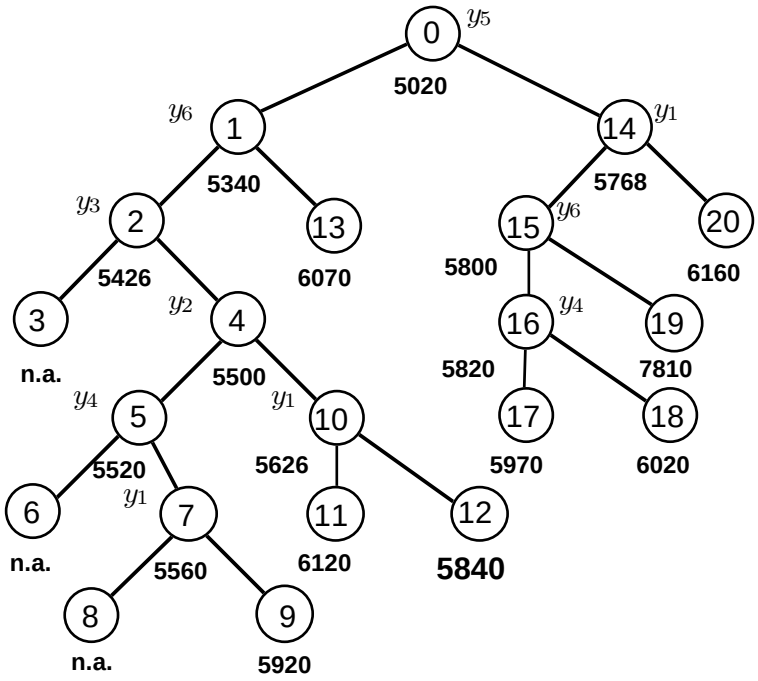

FIGURA 14.7

Si consideri un problema nella forma:

$$\begin{aligned} v := \min \quad & c\,x \\ & A^1\,x \geq b^1 \\ & A^2\,x \geq b^2 \\ & x \in \{0,1\}^n \end{aligned} \tag{14.11}$$

e si costruisca il duale come

$$L(u) = \min\left\{c\,x + u\,(b^1 - A^1\,x) \;:\; A^2\,x \geq b^2,\; x \in \{0,1\}^n\right\} \tag{14.12}$$

Indichiamo con d l'ottimo duale di (14.11). Sia v' il valore ottimo del rilassamento d'interezza di (14.11). La funzione duale del rilassamento d'interezza è definita da

$$\bar{L}(u) = \min\left\{c\,x + u\,(b^1 - A^1\,x) \;:\; A^2\,x \geq b^2,\; x \in [0,1]^n\right\}$$

L'ottimo duale del rilassamento d'interezza, trattandosi di programmazione lineare, vale v'. Ovviamente si ha $\bar{L}(u) \leq L(u)$ per ogni u, da cui si ottiene $v' \leq d \leq v$. Interessa allora capire quando $v' < d$ e quando $v' = d$. È stata introdotta da Geoffrion [1974] la seguente definizione:

14.12 Definizione. *Si dice che vale la* proprietà d'interezza *se* $L(u) = \bar{L}(u)$ *per ogni* u. ∎

È chiaro allora che, valendo la proprietà d'interezza, si ha $v' = d$ e in particolare $v' \geq L(u)$ per ogni $u \geq 0$. Quindi, a meno di non ottenere l'ottimo duale, il rilassamento d'interezza produce limitazioni inferiori migliori di quelle del rilassamento lagrangiano. Normalmente il valore ottimo del rilassamento d'interezza si ottiene facilmente risolvendo un problema di programmazione lineare, mentre trovare un ottimo duale richiede la scrittura di algoritmi ad hoc e non sempre elementari. Quindi, in tutti i casi in cui vale la proprietà d'interezza, non è consigliabile seguire un approccio lagrangiano. Questo avviene ad esempio se A^2 è una matrice totalmente unimodulare e deriva proprio da una separazione dei vincoli fatta apposta per togliere di mezzo i vincoli complicanti!

Va comunque detto che, anche in presenza della proprietà d'interezza, il passaggio attraverso il problema duale può decomporre il problema a dimensioni computazionalmente più vantaggiose, oppure può evidenziare strutture particolari e permettere quindi l'adozione di algoritmi particolarmente veloci. Si tratta di valutazioni da fare caso per caso.

Il rilassamento lagrangiano diventa in ogni caso pagante se $L(u) > \bar{L}(u)$. Purtroppo questa richiesta implica che, nel calcolo di $L(u)$, ci sia differenza qualitativa fra minimizzare rispetto a variabili intere oppure no, e spesso questa differenza si riflette in una differenza di complessità computazionale. Una minimizzazione potrebbe essere polinomiale e l'altra **NP**-difficile. Non ha senso una funzione duale il cui calcolo richieda di risolvere un problema **NP**-difficile, a meno che non si tratti di un problema di knapsack. In questo caso, anche se il problema è NP-completo, la complessità può rimanere contenuta in base ai dati del problema. Si riveda l'esempio 5.27.

14.13 Esempio. Uno dei più famosi esempi di applicazione del rilassamento lagrangiano riguarda il problema del TSP simmetrico. La formulazione duale è quella esposta nell'esempio 5.20. Il metodo suggerito per massimizzare la funzione duale è la quasi ascesa. La regola di suddivisione è simile a quella esposta nell'esempio 14.4. La scelta del circuito su cui operare la suddivisione è alquanto complessa e non viene qui riportata. Va menzionato

il fatto che con questa tecnica di enumerazione si poterono per la prima volta risolvere problemi con più di 500 nodi.

Si è già fatto notare alla fine della sezione 12.1 che l'ottimo duale ha lo stesso valore del rilassamento (12.15) e quindi si possono usare più vantaggiosamente le tecniche poliedrali. Ritorneremo su questo punto più avanti. ∎

14.14 Esempio. Si consideri il problema dell'assegnamento generalizzato esposto nell'esempio 5.27. Facendo riferimento alla notazione di quell'esempio, valutiamo le limitazioni inferiori che si ottengono con le formulazioni duali alternative $L_a(u)$ e $L_b(v)$. È immediato notare che per la funzione duale $L_a(u)$ vale la proprietà d'interezza e quindi l'ottimo duale deve coincidere con il rilassamento d'interezza. Per calcolare $L_b(v)$ invece bisogna risolvere tanti problemi di knapsack quante sono le macchine e quindi ci possiamo aspettare una limitazione inferiore più alta.

Consideriamo la seguente istanza (molto piccola, così i conti possono essere eseguiti manualmente): $m = 2$ macchine, $n = 2$ lavori, costi c_{ij} (i indice di lavoro, j di macchina) dati da $c_{11} = 3$, $c_{12} = 5$, $c_{21} = 5$, $c_{22} = 8$, e tempi di esecuzione a_{ij} dati da $a_{11} = 5$, $a_{12} = 8$, $a_{21} = 7$, $a_{22} = 5$, e infine capacità b_j di ogni macchina data da $b_1 = b_2 = 10$.

Esaminando i costi si vede che la macchina 1 è preferita da entrambi i lavori. Quindi ci sono essenzialmente due soluzioni da valutare, quealla in cui il primo lavoro viene assegnato integralmente alla macchina 1 e l'altra in cui il secondo lavoro viene assegnato integralmente alla macchina 1. Nel primo caso si ha $x_{11} := 1$ e $x_{12} := 0$. Fatta questa assegnazione non resta che assegnare più che si può del secondo lavoro alla prima macchina e la parte restante alla seconda. Quindi si ottiene $x_{21} := 5/7$ e $x_{22} := 2/7$. Il costo di questa soluzione è allora $3 + 5/7 \cdot 5 + 2/7 \cdot 8 = 62/7 = 9 - 1/7$. Per l'altra soluzione si ottiene $x_{21} := 1$ e $x_{22} := 1$, $x_{11} := 3/5$ e $x_{12} := 2/5$. Il costo di questa soluzione è allora $5 + 3/5 \cdot 3 + 2/5 \cdot 5 = 44/5 = 9 - 1/5 < 9 - 1/7$. Quindi la seconda soluzione è l'ottimo rilassato. Per calcolare le variabili duali ottime relative ai vincoli di assegnamento si tenga presente che se aumentiamo di ε la quantità da assegnare del lavoro 1, questa quantità non può che andare sulla macchina 2 con un aumento del costo di $5\,\varepsilon$, quindi la variabile duale v_1 vale -5. Mentre se aumentiamo di ε la quantità da assegnare del lavoro 2, questa quantità va sulla macchina 1 con un aumento del costo di $5\,\varepsilon$, ma contemporaneamente $7/5$ del lavoro 1 passano sulla macchina 2 con un aumento di costo di $14/5\,\varepsilon$. Quindi si ha $v_2 = -5 - 14/5 = -7.8$.

Il calcolo di $L_b(v)$ richiede la risoluzione dei due problemi di knapsack:

$$\begin{aligned} K_1 := \max\ & (-3 - v_1)\,x_{11} + (-5 - v_2)\,x_{21} \\ & 5\,x_{11} + \quad 7\,x_{21} \le 10 \\ & x_{ij} \in \{0,1\} \end{aligned}$$

$$\begin{aligned} K_2 := \max\ & (-5 - v_1)\,x_{12} + (-8 - v_2)\,x_{22} \\ & 8\,x_{12} + \quad 5\,x_{22} \le 10 \\ & x_{ij} \in \{0,1\} \end{aligned}$$

Possiamo valutarli per i valori ottimi duali del rilassamento d'interezza:

$$\begin{aligned} K_1 := \max\ & 2\,x_{11} + 2.8\,x_{21} \\ & 5\,x_{11} + \quad 7\,x_{21} \le 10 \\ & x_{ij} \in \{0,1\} \end{aligned} \qquad \begin{aligned} K_2 := \max\ & 0\,x_{12} - 0.2\,x_{22} \\ & 8\,x_{12} + \quad 5\,x_{22} \le 10 \\ & x_{ij} \in \{0,1\} \end{aligned}$$

L'ottimo del primo knapsack è unico ed è $\hat{x}_{11} := 0$, $\hat{x}_{21} := 1$, $K_1 = 2.8$, mentre vi sono due ottimi per il secondo knapsack, sia $x_{21} = 0$, $x_{22} = 0$ che $x_{21} = 1$, $x_{22} = 0$ con $K_2 = 0$.

Notiamo che il secondo ottimo, unitamente all'ottimo del primo knapsack soddisfa il vincolo di assegnamento e pertanto abbiamo la garanzia che questi valori corrispondono all'ottimo duale. Quindi l'ottimo duale vale $L(-5,-7.8) = 5 + 7.8 - 2.8 = 10$. Come si vede la limitazione inferiore che si ottiene è nettamente migliore del rilassamento d'interezza. In questo caso si ottiene addirittura l'ottimo primale, ma questo fatto non è così automatico in generale. Anche se si ottiene la dualità forte non è detto che, di fronte a soluzioni alternative, come nell'esempio, gli algoritmi di knapsack producano proprio l'ottimo primale.

Conviene scegliere come valore duale per il rilassamento lagrangiano proprio l'ottimo duale del rilassamento d'interezza, che, nella maggior parte dei casi, coincide con l'ottimo di $L_b(v)$. Ovviamente conviene risolvere i problemi di knapsack con la programmazione dinamica.

A titolo d'esempio con istanze più grandi (15 lavori su 4 macchine e dati casuali compresi fra 1 e 10) si ottiene un valore di rilassamento d'interezza mediamente del 5 % inferiore al valore ottimo, mentre il rilassamento lagrangiano produce proprio il valore ottimo. Quindi, non appena si trova l'ottimo in qualche nodo dell'albero di ricerca, la visita termina. ∎

14.5. Altri rilassamenti

Altri tipi di rilassamento possono dipendere dalla particolare struttura del problema. Ne presentiamo due esempi significativi:

14.15 Esempio. (vedi esempio 14.4) Si consideri nuovamente il TSP asimmetrico per il grafo orientato completo $G = (N, A)$ (sono cioè presenti tutte le coppie (i, j) con $i \in N$, $j \in N$ e $i \neq j$). Si associ a G il grafo bipartito $BG = (N_1, N_2, BA)$ con N_1 e N_2 due copie di V. Ogni arco $(i, j) \in A$ è associato ad un arco in BA che collega $i \in N_1$ a $j \in N_2$. Quindi, rispetto al grafo bipartito completo mancano solo gli archi (i, i). Ogni accoppiamento perfetto di BG corrisponde ad una permutazione *ammissibile* in $S_{|N|}$. I cicli della permutazione possono essere interpretati come circuiti orientati in G e circuiti hamiltoniani corrispondono a permutazioni *cicliche* ammissibili.

Un rilassamento del problema consiste nel considerare ammissibili tutte le permutazioni e non soltanto quelle cicliche. Quindi ad ogni nodo dell'albero di ricerca si risolve un problema di assegnamento pesato in BG. Se l'assegnamento non è una permutazione ciclica, si sceglie una dei circuiti forniti dall'assegnamento e si suddivide il problema come indicato nell'esempio 14.4.

Sia ad esempio $|N| = 6$ e sia data la seguente tabella di costi:

	1	*2*	*3*	*4*	*5*	*6*
1		2	3	5	6	7
2	4		3	7	8	6
3	3	3		9	5	7
4	5	7	1		3	2
5	5	7	6	4		2
6	4	9	7	3	2	

Prima di iniziare a sviluppare l'albero di ricerca è conveniente trovare una soluzione iniziale che sia abbastanza buona e che permetta di sfrondare l'albero, quanto meno nella fase iniziale. Consideriamo allora la seguente euristica: partendo dal nodo 1 si visiti il nodo più vicino e non ancora visitato (in caso di più alternative si scelga ad esempio il nodo di indice

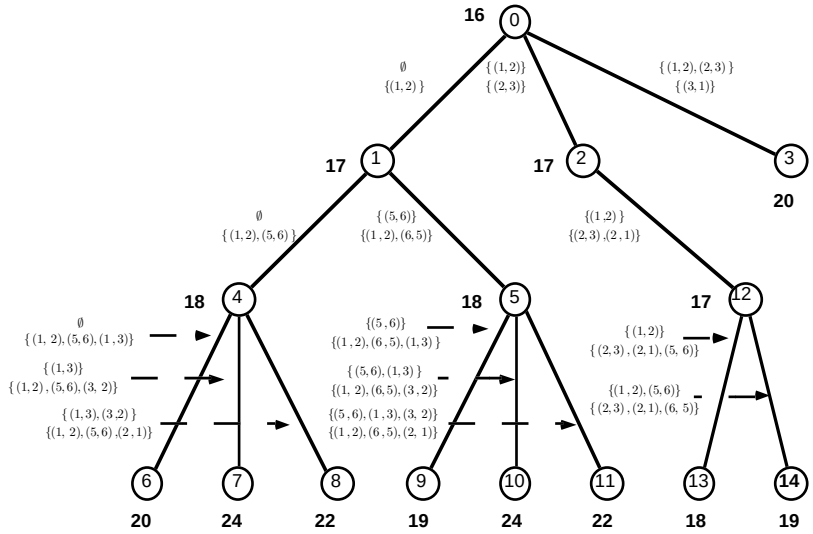

FIGURA 14.8

inferiore). Questa euristica produce il circuito $1 \to 2 \to 3 \to 5 \to 6 \to 4 \to 1$ con un costo di 20.

Risolvendo il problema dell'assegnamento con costi dati dalla tabella si ottiene la soluzione $x_{12} = x_{23} = x_{31} = x_{45} = x_{56} = x_{64} = 1$ (e ogni altra variabile nulla) con costo 16. Come si vede questa soluzione corrisponde ai due cicli $1 \to 2 \to 3 \to 1$ e $4 \to 5 \to 6 \to 1$. Per suddividere il problema scegliamo il circuito $1 \to 2 \to 3 \to 1$. Vengono allora generati tre sottoproblemi contraddistinti dagli insiemi di archi obbligati A^+ e di archi vietati A^-: $(A_1^+ := \emptyset,\ A_1^- := \{(1,2)\})$, $(A_2^+ := \{(1,2)\},\ A_2^- := \{(2,3)\})$, $(A_3^+ := \{(1,2),(2,3)\},\ A_3^- := \{(3,1)\})$.

L'albero di ricerca è rappresentato in figura 14.8. All'interno di ogni nodo è indicato il numero del sottoproblema corrispondente. Vicino ad ogni nodo è riportata la limitazione inferiore del sottoproblema rilassato corrispondente. Vicino ad ogni arco è evidenziato il sottoproblema indicando gli insiemi di archi obbligati (riga superiore) e archi vietati (riga inferiore).

Risolvendo il problema n. 1 si ottengono due cicli $1 \to 4 \to 3 \to 2 \to 1$ e $5 \to 6 \to 5$ di costo 17. Scegliamo il ciclo più corto per operare la successiva suddivisione. Quindi si aggiungono i sottoproblemi $(A_4^+ := \emptyset,\ A_4^- := \{(1,2),(5,6)\})$, $(A_5^+ := \{(5,6)\},\ A_5^- := \{(1,2),(6,5)\})$.

Conviene a questo punto operare una ricerca dell'albero in profondità in modo da ottenere al più presto una soluzione ammissibile. Quindi si passa a risolvere il problema n. 4, ottenendo ancora una soluzione con due cicli e di costo 18. I cicli sono $1 \to 3 \to 2 \to 1$ e $4 \to 6 \to 5 \to 4$. Scegliamo il primo dei due cicli per la suddivisione per cui vengono generati i sottoproblemi $(A_6^+ := \emptyset,\ A_6^- := \{(1,2),(5,6),(1,3)\})$,$(A_7^+ := \{(1,3)\},\ A_7^- := \{(1,2),(5,6),(3,2)\})$,$(A_8^+ := \{(1,3),(3,2)\},\ A_8^- := \{(1,2),(5,6),(2,1)\})$.

Il problema n. 6 genera una soluzione non ammissibile di costo 20 ($1 \to 4 \to 6 \to 5 \to 1$ e $2 \to 3 \to 2$). Tuttavia non è necessario generare ulteriori sottoproblemi, perché con i vincoli imposti non vi può essere nessuna soluzione migliore dell'incombente. Si passa quindi a risolvere il problema n. 7 che genera una soluzione non ammissibile di costo 24 ($1 \to 3 \to 5 \to 2 \to 1$ e $4 \to 6 \to 4$), e quindi non vengono generati sottoproblemi. Il

problema n. 8 produce una soluzione ammissibile di costo 22 ($1 \to 3 \to 2 \to 4 \to 6 \to 5 \to 1$) che viene scartata.

Ritornando indietro nell'albero di ricerca si risolve il problema n. 5. Si ottiene la soluzione non ammissibile $1 \to 3 \to 2 \to 1$ e $4 \to 5 \to 6 \to 4$ di costo 18. Questa volta bisogna generare altri sottoproblemi usando ad esempio il primo dei due circuiti. Si ha pertanto ($A_9^+ := \{(5,6)\}$, $A_9^- := \{(1,2),(6,5),(1,3)\}$), ($A_{10}^+ := \{(5,6),(1,3)\}$, $A_{10}^- := \{(1,2),(6,5),(3,2)\}$), ($A_{11}^+ := \{(5,6),(1,3),(3,2)\}$, $A_{11}^- := \{(1,2),(6,5),(2,1)\}$).

Risolvendo il problema n. 9 si ottiene una soluzione ammissibile ($1 \to 5 \to 6 \to 4 \to 3 \to 2 \to 1$) di costo 19. Questa soluzione diventa il nuovo incombente. Il problema n. 10 genera la soluzione ammissibile ($1 \to 3 \to 5 \to 6 \to 4 \to 2 \to 1$) di costo 24 e il problema n. 11 genera la soluzione ammissibile ($1 \to 3 \to 2 \to 4 \to 5 \to 6 \to 1$) di costo 22.

Si passa a risolvere il problema n. 2 che genera la soluzione non ammissibile $1 \to 2 \to 1$, $3 \to 5 \to 6 \to 4 \to 3$ di costo 17. Scegliendo il circuito $1 \to 2 \to 1$, si nota che l'alternativa che vieta l'arco $(1,2)$ è inconsistente con la precedente scelta di obbligare lo stesso arco. Pertanto l'unica possibilità è di vietare anche l'arco $(2,1)$. Si genera quindi il sottoproblema ($A_{12}^+ := \{(1,2)\}$, $A_{12}^- := \{(2,3),(2,1)\}$) con soluzione $1 \to 2 \to 4 \to 3 \to 1$, $5 \to 6 \to 5$ di costo 17.

Vengono allora generati i sottoproblemi ($A_{13}^+ := \{(1,2)\}$, $A_{13}^- := \{(2,3),(2,1),(5,6)\}$) e ($A_{14}^+ := \{(1,2),(5,6)\}$, $A_{14}^- := \{(2,3),(2,1),(6,5)\}$). Il problema n. 13 produce la soluzione ammissibile $1 \to 2 \to 6 \to 5 \to 4 \to 3 \to 1$ di costo 18, che diventa pertanto il nuovo incombente. Il problema n. 14 produce la soluzione ammissibile $1 \to 2 \to 5 \to 6 \to 4 \to 3 \to 1$ di costo 19.

Resta da considerare il problema n. 3 con soluzione $1 \to 2 \to 3 \to 5 \to 6 \to 4 \to 1$ di costo 20 (si tratta dell'incombente iniziale). La risoluzione del problema è pertanto terminata e l'ottimo è $1 \to 2 \to 6 \to 5 \to 4 \to 3 \to 1$ di costo 18. ∎

14.16 Esempio. Si consideri la seguente versione di problema a una macchina: sono assegnati n lavori e per ognuno di essi sono definite una data di rilascio r_j, una durata di esecuzione p_j e una data di scadenza d_j. I lavori possono essere eseguiti dalla macchina solo uno alla volta e non è ammessa l'interruzione del lavoro. L'inizio di ogni lavoro deve avvenire non prima della data di rilascio. Il completamento di un lavoro può avvenire anche oltre la data di scadenza, ma in questo caso si incorre in una penalità pari a $C - j - d_j$ (C_j istante di completamento del lavoro j). Si cerca la schedulazione che minimizza $\max L_j$ con $L_j := C_j - d_j$.

Se le date di rilascio fossero tutte uguali il problema sarebbe quello già esaminato nell'Esempio 9.35 e quindi risolvibile con la programmazione dinamica in tempo polinomiale. In questo caso particolare la schedulazione ottima si otterrebbe ordinando i lavori per date di scadenza decrescenti. In presenza di date di rilascio diverse potrebbe invece essere più utile lasciare la macchina inoperosa in attesa della disponibilità imminenente di un lavoro a priorità più elevata, che altrimenti verrebbe eseguito troppo tardi. A causa di queste scelte alternative il problema diventa **NP**-difficile in senso forte (trasformazione dal problema di tripartizione, vedi esempio 1.86).

Si noti che il problema in esame è diverso da quello considerato negli esempi 14.8 e 14.9 dove si minimizzava (come caso particolare) $\sum L_j$. Si definisca $q_j := \max_k d_k - d_j$. Quindi $L_j + \max_k d_k = C_j - d_j + \max_k d_k = C_j + q_j$ e minimizzare $\max L_j$ è equivalente a minimizzare $\max C_j + q_j$. D'ora in poi consideriamo questo come obiettivo. L'analisi che segue si deve a Carlier [1982].

Sia J un qualsiasi sottoinsieme di lavori. Nessun lavoro in J può iniziare prima di $\min_{j \in J} r_j$. Quindi il completamento di tutti i lavori in J non può avvenire prima di

```
Regola di Jackson
begin
    i lavori siano ordinati come r_j ≤ r_{j+1}; r_{n+1} := +∞
    S := ∅; j := 1; i := 1;
    while (j ≤ n) ∨ (S ≠ ∅) do
    begin
        if S = ∅ then τ := r_j;
        while τ ≥ r_j do
        begin
            S := S ∪ {j}; j := j + 1;
        end;
        sia k ∈ S : q_k ≥ q_h, ∀h ∈ S;
        t(i) := τ; i := i + 1;
        S := S \ {k}; τ := τ + p_k;
    end;
end.
```

$\min_{j\in J} r_j + \sum_{j\in J} p_j$ e allora il costo di una qualsiasi schedulazione non può essere inferiore a $\min_{j\in J} r_j + \sum_{j\in J} p_j + \min_{j\in J} q_j$. Possiamo pertanto definire la funzione

$$h(J) := \min_{j\in J} r_j + \sum_{j\in J} p_j + \min_{j\in J} q_j$$

e usarla come limitazione inferiore. In particolare $\max_J h(J)$ rappresenta la migliore limitazione inferiore. Il calcolo di $\max_J h(J)$ è meno difficile di quanto possa sembrare. Si supponga di rilassare la condizione che i lavori non debbano essere interrotti. La situazione è allora assimilabile al caso con date di rilascio tutte uguali. Pensando di simulare nel tempo l'esecuzione dei lavori, ogni volta in cui un nuovo lavoro risulta disponibile, il fatto di poter interrompere quello in esecuzione rende tutti i lavori ugualmente disponibili e quindi conviene schedulare quello a più alta priorità. Ci sono tempi morti allora solo se avviene che sia stato completato un insieme di lavori e non sia disponibile nessun altro lavoro. In questi casi il problema si decompone in sottoproblemi. Sia J uno di questi sottoinsiemi di lavori. Si vede immediatamente che $\max_{j\in J} C_j + q_j = h(J)$ e quindi $\max_J h(J)$ si ottiene schedulando i lavori con prelazione (*preemption*).

Carlier [1982] suggerisce di usare un algoritmo greedy che schedula i lavori in ordine di disponibilità e di priorità (detto regola di Jackson) che, anche se non calcola $\max_J h(J)$, fornisce una buona limitazione inferiore ed anche una soluzione ammissibile (di valore $t(n+1)$). Questa viene confrontata immediatamente con la limitazione inferiore per provarne eventualmente l'ottimalità e, fatto molto importante, si ricava un criterio di suddivisione estremamente pratico ed efficiente.

Il metodo di suddivisione verrà illustrato sui seguenti dati:

	1	2	3	4	5	6
r_j	0	2	5	5	8	9
p_j	3	1	2	3	4	3
d_j	14	16	15	17	14	15
q_j	3	1	2	0	3	2

La regola di Jackson schedula i lavori nell'ordine indicato. Sulla base di questo ordine si costruisce un grafo orientato (vedi figura 14.9) con nodi $\{0, 1, \ldots, n, n+1\}$, archi $(0, j)$,

$j := 1, \ldots, n$, $(j, j+1)$, $j := 1, \ldots, n-1$, $(j, n+1)$, $j := 1, \ldots, n$. Agli archi $(0, j)$ del primo gruppo si attribuiscono lunghezze r_j, agli archi $(j, j+1)$ del secondo gruppo lunghezze p_j e a quelli $(j, n+1)$ del terzo gruppo lunghezze $p_j + q_j$. Su questo grafo si identifica il cammino più lungo (evidenziato in grassetto in figura 14.9). Sia I l'insieme dei lavori sul cammino critico. Supponiamo a scopo illustrativo che I sia costituito dagli ultimi lavori (nell'esempio $I = \{3, 4, 5, 6\}$ e $t(n+1) = 19$).

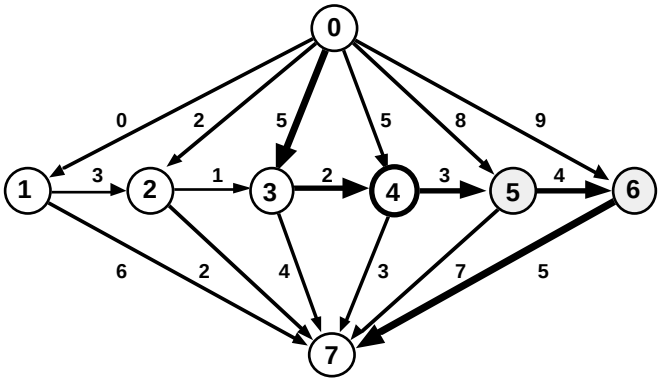

FIGURA 14.9

Se avviene che $q_n \leq q_j$, $\forall j \in I$, allora la soluzione ottenuta dalla regola di Jackson è ottima. Infatti il valore $t(n+1)$ di questa schedulazione è pari alla limitazione inferiore $h(I)$.

Se invece ciò non avviene sia c l'ultimo lavoro in I tale che $q_c < q_n$. Tale lavoro viene detto *lavoro critico* e l'insieme $J_c := \{c+1, \ldots, n\}$ viene detto *insieme critico*. Nell'esempio $c = 4$ e $J_c = \{5, 6\}$. L'osservazione cruciale è che quando la regola di Jackson schedula il lavoro critico, nessuno dei lavori dell'insieme critico è ancora disponibile, altrimenti, essendo questi a priorità più alta, ne verrebbe schedulato qualcuno. Allora si ha $\min_{j \in J_c} r_j > t(c) = \min_{j \in I} r_j + \sum_{j \in I \setminus J_c} p_j - p_c$. Quindi possiamo valutare

$$h(J_c) = \min_{j \in J_c} r_j + \sum_{j \in J_c} p_j + \min_{j \in J_c} q_j > \min_{j \in I} r_j + \sum_{j \in I \setminus J_c} p_j - p_c + \sum_{j \in J_c} p_j + q_n =$$

$$\min_{j \in I} r_j + \sum_{j \in I} p_j - p_c + q_n = t(n+1) - p_c$$

relazione che dà un'indicazione della bontà della limitazione inferiore. Si ha quindi, indicando con $\hat{t}$ il valore ottimo del problema,

$$h(J_c) \leq \hat{t} \leq t(n+1)$$

che nell'esempio diventa $17 \leq \hat{t} \leq 19$. Si noti che $h(J_c)$ non fornisce necessariamente il massimo di $h(J)$, però fornisce comunque una limitazione inferiore sufficientemente buona da non rendere pagante l'onere computazionale di schedulare i lavori anche rispetto alla opzione con prelazione.

Ciò che si ricava dall'evidenziare il lavoro critico e l'insieme critico è che in una schedulazione ottima il lavoro critico viene schedulato o prima dei lavori dell'insieme critico oppure dopo, ma mai in mezzo a loro. Si lascia la dimostrazione di questo fatto come esercizio non difficile, dopo tutte le considerazioni fatte. Quindi la regola di suddivisione si basa

sull'imposizione di precedenze, simili a quelle dell'esempio 14.5, ma un po' più stringenti. Bisogna sempre fare in modo che, nell'imporre regole di suddivisione, queste permettano poi di affrontare i sottoproblemi nei vari nodi dell'albero di ricerca usando sempre la stessa tecnica del nodo radice. Quindi le precedenze non possono essere imposte esplicitamente, ma vanno attuate implicitamente pensando a come opera la regola di Jackson. Quindi per imporre $c \prec J_c$ si ridefinisce $q_c := \max_{j \in J_c} q_j$ e per imporre $c \succ J_c$ si ridefinisce $r_c := \max_{j \in J_c} r_j$. Quindi nei nodi successori del nodo radice dell'esempio vengono creati due problemi con i seguenti dati:

	1	2	3	4	5	6
r_j	0	2	5	5	8	9
p_j	3	1	2	3	4	3
d_j	14	16	15	17	14	15
q_j	3	1	2	3	3	2

	1	2	3	4	5	6
r_j	0	2	5	9	8	9
p_j	3	1	2	3	4	3
d_j	14	16	15	17	14	15
q_j	3	1	2	0	3	2

Per il primo sottoproblema la regola di Jackson fornisce la medesima schedulazione del nodo radice, ma siccome il valori di q_j sono cambiati, si vede che questa schedulazione è ottima relativamente a tutte le schedulazioni tali che $c \prec J_c$. Per il secondo sottoproblema si ottiene $t(n+1) = 8 + 10 + 0 = h(4, 5, 6)$ e quindi anche questa schedulazione è ottima nel suo sottoalbero. Siccome è migliore della prima, è anche l'ottimo. Si vedano in figura 14.10 le due schedulazioni. Si noti come non sia comunque possibile completare tutti i lavori entro le scadenze. ∎

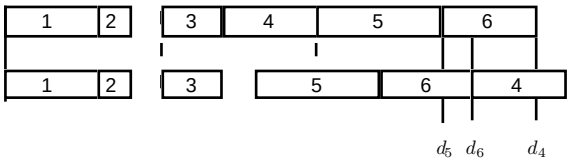

FIGURA 14.10

Il rilassamento lagrangiano può anche esser visto come un metodo per togliere di mezzo dei vincoli che complicano la risoluzione di un problema. Ovviamente vi sono più modi di eliminare vincoli fastidiosi. Un semplice metodo consiste nell'aggregare molte disequazioni in una sola.

14.17 DEFINIZIONE. *Sia dato il problema* $v = \min\{f(x) : g(x) \leq 0, x \in X\}$. *Sia* $u \geq 0$ *un vettore di dimensione opportuna. Il problema* $v' = \min\{f(x) : \hat{u}\, g(x) \leq 0, x \in X\}$ *viene detto* problema surrogato *del precedente.* ∎

Il problema surrogato ha un'unica diseguaglianza e ovviamente ogni soluzione ammissibile nel problema originario è ammissibile in quello surrogato, ed è oltrettanto ovvio che molte soluzioni ammissibili in quello surrogato non lo sono in quello originario. Quindi il problema surrogato costituisce un rilassamento.

Il valore ottimo surrogato dipende dal vettore di moltiplicatori $u \geq 0$ e c'è quindi il problema di determinare i moltiplicatori migliori (cioè quelli che producono la limitazione inferiore più alta). Si supponga che per il generico problema $v = \min\{f(x) : g(x) \leq 0, x \in X\}$ valga la dualità forte e quindi si abbia $L(\hat{u}) = v$, con $\hat{u}$ ottimo duale. Sia $L'(\mu)$ la funzione duale del problema surrogato. Quindi $L(\mu\,\hat{u}) = L'(\mu)$ e siccome, per l'ottimalità duale, $L(\mu\,\hat{u}) \leq L(\hat{u})$ per ogni $\mu \geq 0$, si ha che $\mu = 1$ è ottimo duale del problema surrogato.

Allora $v' \geq L'(1) = L(\hat{u}) = v \geq v'$, da cui $v' = v$, vale la dualità forte anche nel problema surrogato e l'ottimo del problema originario è ottimo del problema surrogato.

Se invece non vale la dualità forte, sia d il valore ottimo duale con $\hat{u}$ ottimo duale. Ragionando come nel caso precedente, si ha $v \geq v' \geq L'(1) = L(\hat{u}) = d$ e quindi il problema surrogato fornisce una limitazione inferiore non peggiore di quella del rilassamento lagrangiano e del rilassamento d'interezza. Questo miglioramento ha un costo computazionale, perché si deve comunque risolvere in ottimalità il problema duale, oppure il rilassamento d'interezza per problemi di programmazione lineare intera. Ci si può chiedere se, quando non vale la dualità forte, ci siano dei moltiplicatori migliori dell'ottimo duale da usare come problema surrogato. Questo può effettivamente succedere (vedi esercizio 14.20 più avanti). Purtroppo la dipendenza di v' da u non è semplice da studiare e quindi conviene usare direttamente l'ottimo duale.

14.18 Esempio. Si consideri ad esempio il seguente problema di programmazione lineare intera:

$$\begin{aligned} v = \max\ & c\,x \\ & A\,x \leq b \\ & x \in \mathbb{Z}^n_+ \end{aligned} \tag{14.13}$$

e sia $\hat{u}$ il suo ottimo duale. Allora il problema surrogato è il seguente problema di knapsack intero

$$\begin{aligned} v' = \max\ & c\,x \\ & (u\,A)\,x \leq u\,b \\ & x \in \mathbb{Z}^n_+ \end{aligned} \tag{14.14}$$

che ha una particolare struttura dei coefficienti se si sceglie come vettore di moltiplicatori l'ottimo duale $\hat{u}$ del rilassamento d'interezza di (14.13). Infatti $\hat{u}\,B = c_B$ con B matrice di base ottima e quindi il coefficiente di indice j dell'obiettivo è uguale al corrispettivo coefficiente del vincolo se j è nella base ottima, altrimenti è non inferiore. Conviene usare la programmazione dinamica per risolvere questo problema di knapsack e in particolare, siccome i coefficienti del vincolo possono essere non interi, conviene usare l'algoritmo di programmazione dinamica che opera sull'obiettivo.

Per capire esattamente la differenza fra il rilassamento lagrangiano e quello surrogato, si consideri il seguente esempio:

$$\begin{aligned} \max\ & 3\,x_1 + 2\,x_2 \\ & 4\,x_1 \leq 7 \\ & 4\,x_2 \leq 7 \\ & x_i \in \mathbb{Z}_+ \end{aligned} \tag{14.15}$$

È fin troppo evidente che, data l'interezza, i vincoli potrebbero essere ristretti a $x_i \leq 1$. Tuttavia non procediamo in questo modo perché in generale, come si è visto nel capitolo precedente, non è facile descrivere l'inviluppo convesso delle soluzioni intere. L'istanza dell'esempio è certamente banale, ma in questo modo si riesce a capire il significato dei vari rilassamenti. Il massimo del rilassamento d'interezza di (14.15) è naturalmente $\bar{x}_1 = \bar{x}_2 = 7/4$ con valore ottimo $35/4$ e variabili duali ottime $\hat{u}_1 = 3/4$ e $\hat{u}_2 = 1/2$, da cui si ottiene il rilassamento surrogato:

$$\begin{aligned} \max\ & 3\,x_1 + 2\,x_2 \\ & 12\,x_1 + 8\,x_2 \leq 35 \\ & x_i \in \mathbb{Z}_+ \end{aligned} \tag{14.16}$$

il cui valore ottimo è $8 < 35/4$ con ottimi $(2,1)$, oppure $(0,4)$. L'ottimo di (14.15) è invece $(1,1)$ con valore $5 < 8$. In figura 14.11 si vedono raffigurati l'insieme ammissibile del rilassamento d'interezza (quadrato più grande, tratteggiato, compresa la parte nascosta), l'insieme ammissibile di (14.15) (punti neri), il suo inviluppo convesso (quadrato più piccolo), l'insieme ammissibile di (14.16) (punti neri e punti bianchi), il suo inviluppo convesso (trapezio in grigio, compresa la parte nascosta). ∎

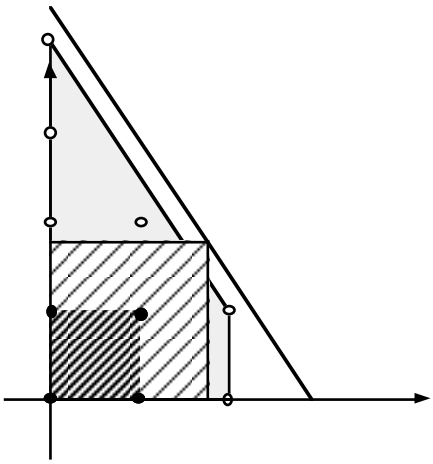

FIGURA 14.11

14.19 ESEMPIO. Vogliamo illustrare con un esempio la dipendenza di v' da u. Sia dato

$$\begin{aligned} \max \quad & x_1 + \quad x_2 \\ & 7\,x_1 + 5\,x_2 \leq 16 \\ & 2\,x_1 + 3\,x_2 \leq \quad 6 \\ & x_1, x_2 \geq 0 \end{aligned}$$

Possiamo senza perdita di generalità considerare due moltiplicatori della forma u e $(1-u)$ con $0 \leq u \leq 1$. Quindi otteniamo

$$\begin{aligned} \max \quad & x_1 + x_2 \\ & (2+5\,u)\,x_1 + (3+2\,u)\,x_2 \leq 6 + 10\,u \\ & x_1, x_2 \geq 0 \end{aligned}$$

L'ottimo è $\hat{x}_1 = (6+10\,u)/(2+5\,u)$, $\hat{x}_2 = 0$, se $(2+5\,u) \leq (3+2\,u)$, cioè se $u \leq 1/3$ e $\hat{x}_1 = 0$, $\hat{x}_2 = (6+10\,u)/(3+2\,u)$, se $u \geq 1/3$. Il grafico del valore ottimo in funzione di u è rappresentato in figura 14.12. Come si vede la funzione non è convessa (andrebbe minimizzata, perché?). ∎

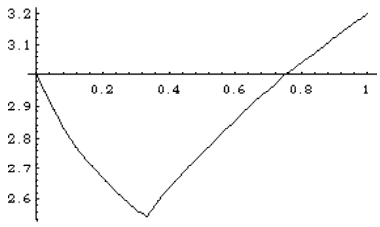

FIGURA 14.12

14.20 Esercizio. Si trovi l'andamento del valore ottimo surrogato in funzione dei moltiplicatori u e $(1-u)$ $(0 \le u \le 1)$ e si trovi un valore di u che produce una limitazione superiore (è un problema di massimo) migliore di quella data dall'ottimo duale $\hat{u}$ (normalizzato), per il problema

$$\max \{3\,x_1 + x_2 : 3\,x_1 \le 1,\ 3\,x_2 \le 1,\ x_i \in \mathbb{Z}_+\}$$

∎

14.6. Rilassamento gruppale

Consideriamo ora il problema di programmazione lineare intera:

$$\begin{aligned} \min \quad & c\,x \\ & A\,x = b \\ & x \in \mathbb{Z}_+^n \end{aligned} \tag{14.17}$$

Sia B la matrice di base ottima del rilassamento d'interezza di (14.17). Possiamo pensare di rilassare il vincolo di non negatività sulle variabili in base e formare il problema

$$\begin{aligned} \min \quad & c_B\,x_B + c_N\,x_N \\ & B\,x_B + N\,x_N = b \\ & x_B \in \mathbb{Z}^m,\ x_N \in \mathbb{Z}_+^{n-m} \end{aligned} \tag{14.18}$$

La motivazione di questa scelta è dovuta al fatto, di facile dimostrazione, che il rilassamento d'interezza di (14.18) produce lo stesso ottimo del rilassamento d'interezza di (14.17). Quindi si può sperare che la limitazione inferiore che si ottiene quando si considerano variabili intere sia abbastanza buona. Per risolvere (14.18) bisogna far ricorso alla cosiddetta forma normale di Smith di una matrice, che qui ricapitoliamo brevemente.

Data una matrice A di interi esistono sempre due matrici quadrate unimodulari Q e R e una matrice S tali che

$$Q\,A\,R = S \qquad \text{e} \qquad S = \begin{pmatrix} \Delta & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{pmatrix}$$

dove $\Delta > 0$ è diagonale e il termine δ_i della diagonale divide il termine δ_{i+1}. La matrice S prende il nome di *forma normale di Smith* di A e si può calcolare in tempo polinomiale (Kannan e Bachem [1979]).

Il calcolo di S si può vedere come una estensione dell'algoritmo di Euclide per il calcolo del MCD di due interi. Si tratta di operare una serie di trasformazioni elementari su A fino a portarla nella forma normale di Smith. Le operazioni elementari sono: cambio del segno di una riga (colonna), permutazione di due righe (colonne), addizione ad una riga (colonna) del multiplo intero di un'altra riga (colonna). Queste operazioni sono equivalenti a premoltiplicare (postmoltiplicare) la matrice per una matrice elementare ottenuta dall'identica eseguendo su questa la medesima operazione. Le matrici elementari sono quindi unimodulari. Per arrivare alla matrice S bisogna portare, tramite permutazioni, l'elemento non nullo e di minimo valore assoluto in posizione $(1,1)$. Sia a_{11} tale elemento. Dopodiché gli altri elementi non nulli a_{1j} della prima riga e a_{i1} della prima colonna vengono ridotti ai resti delle divisioni a_{1j}/a_{11} e a_{i1}/a_{11} con l'operazione elementare di addizione di riga o colonna. Ricorsivamente si sposta l'elemento non nullo minimo in posizione $(1,1)$ finché $a_{1j} = 0$, $j > 1$ e $a_{i1} = 0$, $i > 1$. Poi si procede analogamente con le righe e colonne da 2 in poi.

14.21 Esempio. Le operazioni sulla matrice

$$A = \begin{pmatrix} 3 & 0 & 1 \\ 0 & 2 & 6 \end{pmatrix}$$

sono le seguenti

$$A_1 := \begin{pmatrix} 1 & 0 & 3 \\ 6 & 2 & 0 \end{pmatrix} = A \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} =: A\,R_1 \qquad A_2 := \begin{pmatrix} 1 & 0 & 0 \\ 6 & 2 & -18 \end{pmatrix} = A_1 \begin{pmatrix} 1 & 0 & -3 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} =: A_1\,R_2$$

$$A_3 := \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & -18 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ -6 & 1 \end{pmatrix} A_2 =: Q_1\,A_2 \qquad A_4 := \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \end{pmatrix} = A_3 \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 9 \\ 0 & 0 & 1 \end{pmatrix} =: A_3\,R_3$$

$$S := A_4 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \end{pmatrix} \qquad R := R_1\,R_2\,R_3 = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 9 \\ 1 & 0 & -3 \end{pmatrix} \qquad Q := Q_1 = \begin{pmatrix} 1 & 0 \\ -6 & 1 \end{pmatrix}$$

∎

Sia ora Δ la forma normale di Smith della matrice di base B ottima per il rilassamento d'interezza di (14.17). Essendo B quadrata e non singolare S si riduce alla sola parte diagonale. In particolare $|\det B| = \prod_i \delta_i$ e, se $|\det B|$ è primo, $\delta_i = 1$, $i = 1 \ldots, m-1$, $\delta_m = |\det B|$. Sostituendo si ottiene

$$\Delta\,R^{-1}\,x_B + Q\,N\,x_N = Q\,b$$

Si definisca ora $y := R^{-1}\,x_B$, $H := Q\,N$, $d := Q\,b$, per cui il vincolo $A\,x = b$ è equivalentemente espresso come

$$\Delta\,y = d - H\,x_N \tag{14.19}$$

In (14.19) sono ammissibili solo quei valori x_N tali che $d_i - (H\,x_N)_i$ è multiplo di δ_i per ogni i. In modo equivalente possiamo, a partire da Δ, definire il gruppo abeliano $\mathcal{G} := \oplus\{Z_k : k = \delta_i > 1\}$ (quindi se $|\det B|$ è primo $\mathcal{G} = Z_{|detB|}$) e costruire un omomorfismo $\varphi : Z^m \to \mathcal{G}$ definito da

$$\varphi : z \mapsto (z_m \bmod \delta_m, \ldots z_p \bmod \delta_p)$$

e quindi x_N è ammissibile in (14.19) se e solo se

$$\sum_{j \in \eta} \varphi(H^j)\,x_j = \varphi(d) \tag{14.20}$$

La relazione (14.20) è un'equazione gruppale in cui i valori interi x_j corrispondono a somme ripetute di elementi di $\mathcal{G}$. Si può pensare di costruire un grafo orientato in cui i nodi corrispondono agli elementi di $\mathcal{G}$ e gli archi sono definiti da

$$\big\{(h,k) : k = h + \varphi(H^j),\ h \in \mathcal{G},\ j \in \eta\big\}$$

e gli archi $(h, h + \varphi(H^j))$, $h \in \mathcal{G}$, vengono detti archi di tipo j. Allora l'ammissibilità di (14.20) per valori $x_N \geq 0$ è equivalente all'esistenza di un cammino orientato dall'elemento neutro di $\mathcal{G}$ a $\varphi(d)$. Ogni cammino di questo tipo definisce univocamente il vettore x_N, che a sua volta determina univocamente y in (14.19) e da $R\,y = x_B$ si determina x_B. Tuttavia non c'è nessuna garanzia che x_B sia non negativo e a priori non si riesce a stabilire se un certo cammino produrrà un x_B ammissibile oppure no. In ogni caso vale il seguente teorema di facile dimostrazione:

14.22 Teorema. *Il problema (14.18) è ammissibile se e solo se esiste in $\mathcal{G}$ un cammino dall'elemento neutro a $\varphi(d)$.* ∎

Quindi fra i molti cammini in $\mathcal{G}$ ci sono anche quelli che producono valori $x_B \geq 0$ (sempreché (14.17) sia ammissibile) e fra questi siamo interessati a quello che produce il minimo valore della funzione obiettivo. Avendo già espresso le variabili in funzione degli indici fuori base, è naturale considerare come coefficienti di costo delle variabili fuori base i loro costi ridotti, che risultano essere non negativi dato che si considera la base ottima. Quindi il problema a cui ci siamo ricondotti è la ricerca di cammini minimi in un grafo orientato con costi non negativi e sono problemi che si risolvono facilmente con l'algoritmo di Dijkstra. La lunghezza di ogni cammino è pari al peggioramento della funzione obiettivo rispetto alla soluzione del rilassamento d'interezza.

Se il cammino minimo in assoluto non produce una soluzione ammissibile si tratta di trovare altri cammini con una procedura di suddivisione. Il metodo che si raccomanda in questo caso si basa sull'imposizione che un arco sia presente almeno un numero fissato p di volte nel cammino che si cerca. Per capire come questa regola di suddivisione possa essere compatibile con il funzionamento dell'algoritmo di Dijkstra serve il seguente lemma di facile dimostrazione:

14.23 Lemma. *Siano assegnati interi non negativi $p_1, \ldots, p_{n-m}$. Il cammino ottimo a $\varphi(d)$ che usa archi di tipo j almeno p_j volte è dato dal cammino minimo a $\varphi(d - H\,p)$ più il cammino dato dagli archi di tipo j ripetuti p_j volte.* ∎

Quindi basta 'dirottare' il cammino minimo sul nodo $\varphi(d - H\,p)$. L'albero di ricerca ha nodi contraddistinti da un particolare valore di p. I suoi successori sono $n-m$ nodi generati da p secondo la seguente regola: sia $\hat{x}_N$ corrispondente al cammino ottimo del nodo p, allora

$$p_i^j := \begin{cases} p_i^j & \text{se } i \neq j \\ \hat{x}_i + 1 & \text{se } i = j \end{cases}$$

14.24 Esempio. Si riprenda in esame il problema di knapsack già considerato precedentemente

$$\begin{aligned} \max \quad & 24\,x_1 + 5\,x_2 + 26\,x_3 + 28\,x_3 \\ & 8\,x_1 + 5\,x_2 + 13\,x_3 + 7\,x_4 \leq 26 \\ & x_i \in \{0,1\} \quad \forall i \end{aligned} \tag{14.21}$$

che riscriviamo, per adattarlo alla formulazione (14.17), come

$$\begin{aligned} \max \quad & 24\,x_1 + 5\,x_2 + 26\,x_3 + 28\,x_3 \\ & 8\,x_1 + 5\,x_2 + 13\,x_3 + 7\,x_4 + x_5 = 26 \\ & x_1 + x_6 = 1 \\ & x_2 + x_7 = 1 \\ & x_3 + x_8 = 1 \\ & x_4 + x_9 = 1 \\ & x_i \geq 0 \quad \forall i \end{aligned} \tag{14.22}$$

Non è difficile ottenere i seguenti dati per la base ottima

$$B = \begin{pmatrix} 8 & 13 & 7 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \qquad N = \begin{pmatrix} 5 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \bar{c}_N = (\,5 \quad 2 \quad 8 \quad 14\,)$$

Il calcolo della forma normale di Smith produce le seguenti matrici:

$$S = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 13 \end{pmatrix} \qquad Q = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & -8 & 0 & -13 & -7 \end{pmatrix} \qquad R = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 \end{pmatrix}$$

e quindi

$$H = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 5 & 1 & -8 & -7 \end{pmatrix} \qquad d = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ -2 \end{pmatrix}.$$

Inoltre l'omomorfismo è semplicemente dato da

$$\varphi(z) := z_m \bmod 13 \Longrightarrow \varphi(H^1) = 5,\ \varphi(H^2) = 1,\ \varphi(H^3) = 5,\ \varphi(H^4) = 6,\ \varphi(d) = 11$$

Il grafo ha 13 nodi e archi dei seguenti tipi

archi di tipo 1:	$h \to h + 5 \mod 13$,	di costo	5
archi di tipo 2:	$h \to h + 1 \mod 13$,	di costo	2
archi di tipo 3:	$h \to h + 5 \mod 13$,	di costo	8
archi di tipo 4:	$h \to h + 6 \mod 13$,	di costo	14

Il cammino più breve è $0 \to 5 \to 10 \to 11$ di costo 12, con due archi di tipo 1 e uno di tipo 2, che implica $x_N = (2, 1, 0, 0)$ da cui $y = (1, -1, 1, 1, -1)$ e $x_B = (1, 0, 1, -1, 1)$, purtroppo non ammissibile. Allora bisogna suddividere generando i quattro nodi $p^1 := (3, 0, 0, 0)$, $p^2 = (0, 2, 0, 0)$, $p^3 = (0, 0, 1, 0)$, $p^4 = (0, 0, 0, 1)$. I nodi su cui far arrivare i cammini minimi sono rispettivamente $\varphi(d - Hp^1) = 9$, $\varphi(d - Hp^2) = 9$, $\varphi(d - Hp^3) = 6$, $\varphi(d - Hp^4) = 5$.

Si ottengono i seguenti quattro cammini nei rispettivi nodi dell'albero di ricerca: per p^1: $x_N = (1, 4, 0, 0) + p^1$ di costo $13 + 15 = 28$ e non ammissibile; per p^2: $x_N = (1, 4, 0, 0) + p^2$ costo $13 + 4 = 17$ e ammissibile con variabili in base $x_B = (1, 0, 1, 0, 1)$, che corrisponde alla soluzione di knapsack $\hat{x}^1 = (1, 1, 0, 1)$; per p^3: $x_N = (1, 1, 0, 0) + p^3$ costo $7 + 8 = 15$ e ammissibile con variabili in base $x_B = (0, 1, 1, 0, 0)$, che corrisponde a $\hat{x}^2 = (0, 1, 1, 1)$; cammino per p^4: $x_N = (1, 0, 0, 0) + p^4$ costo $5 + 14 = 19$ e ammissibile con variabili in base $x_B = (1, 1, 0, 0, 0)$, che corrisponde a $\hat{x}^3 = (1, 1, 1, 0)$. Siccome il valore più basso è anche ammissibile, la ricerca è terminata con l'ottimo. ∎

14.25 Esercizio. Si sfruttino i risultati di questa sezione per ottenere dei piani di taglio del tutto simili a quelli di Gomory (sezione 13.3). ∎

14.7. Decomposizione di Benders

Finora si sono affrontati i problemi con variabili intere cercando di riformulare il problema con variabili continue in modo da poter usare direttamente la programmazione lineare. Si è visto come ciò sia possibile in linea teorica e in alcuni casi si ottengano anche praticamente dei risultati importanti. Può sembrare pertanto sorprendente un approccio ai problemi in cui compaiono variabili intere e non intere che cerchi invece di riformulare il problema usando solo variabili intere. Qualche volta si ottiene una formulazione strutturalmente interessante

dal punto di vista algoritmico. Tale approccio prende il nome di *decomposizione di Benders* (Benders [1962]). Sia dato:

$$\begin{aligned} v := \min \quad & c\,x + d\,z \\ & A\,x + D\,z = b \\ & x \geq 0, \quad z \geq 0, \text{ intero} \end{aligned} \tag{14.23}$$

Possiamo sempre spezzare la minimizzazione in due fasi, una rispetto a x per un fissato valore di z e poi una seconda rispetto a z, cioè:

$$\min \{v(z) : z \geq 0, \text{ intero}\}$$

con

$$\begin{aligned} v(z) := d\,z + \min \quad & c\,x \\ & A\,x = b - D\,z \\ & x \geq 0 \end{aligned} \tag{14.24}$$

o anche, sfruttando la dualità

$$\begin{aligned} v(z) = d\,z + \max \quad & u\,(b - D\,z) \\ & u\,A \ \leq c \end{aligned} \tag{14.25}$$

Se si indicano con $u^1, \ldots, u^p$ i vertici del poliedro duale e con $r^1, \ldots, r^q$ le direzioni estreme, (14.25) si può riscrivere come

$$v(z) = \begin{cases} d\,z + \max\left\{u^i\,(b - D\,z) : i = 1, \ldots, p\right\} & \text{se } r^j\,(b - D\,z) \leq 0, \forall j \\ +\infty & \text{altrimenti} \end{cases}$$

e quindi (14.23) è esprimibile come

$$\begin{aligned} v = \min \quad & t \\ & t \geq d\,z + u^i\,(b - D\,z) \qquad i = 1, \ldots, p \\ & 0 \geq r^j\,(b - D\,z) \qquad\quad j := 1, \ldots, q \end{aligned} \tag{14.26}$$

In (14.26) compare solo una variabile continua ed un numero esponenziale di vincoli. Nonostante questo aspetto critico la formulazione (14.26) può dare luogo a risultati interessanti.

Si può riformulare (14.26) eliminando anche la variabile continua e fissando una soglia per t. Quindi il seguente problema è ammissibile se e solo se esiste esiste una soluzione ammissibile di (14.26) di valore non peggiore di t^*:

$$\begin{aligned} (u^i\,D - d)\,z \ & \geq u^i\,b - t^* \qquad i = 1, \ldots, p \\ r^j\,D\,z \ & \geq r^j\,b \qquad\quad\ \ j := 1, \ldots, q \end{aligned} \tag{14.27}$$

Il problema di ammissibilità (14.27) è un problema a variabili intere con un numero molto elevato di vincoli. Naturalmente solo i vincoli che servono vengono generati. Si pone allora il problema, noto un valore $\bar{z}$ ammissibile per una parte dei vincoli di (14.9), di verificare la sua ammissibilità anche per gli altri vincoli e questo si ottiene semplicemente risolvendo il problema di programmazione lineare (14.24), oppure il suo duale (14.25). Se da questo calcolo risulta $v(\bar{z}) \leq t^*$, allora si è trovata una soluzione ammissibile di costo non peggiore di t^*, altrimenti bisogna generare il vincolo $(\hat{u}\,D - d)\,z \geq \hat{u}\,b - t^*$, se (14.25) ammette soluzione ottima $\hat{u}$, oppure il vincolo $\hat{r}\,D\,z \geq \hat{r}\,b$, se (14.25) è illimitato lungo la direzione di fuga $\hat{r}$.

14.26 Esempio. Si consideri nuovamente il problema di schedulazione dell'esempio 14.16. Sia t_j il tempo di completamento del lavoro j e sia z_{ij} una variabile 0-1 che vale 0 se i precede j e 1 altrimenti. Per simmetria e semplicità notazionale vengono definite sia z_{ij} che z_{ji}, obbligate dal vincolo $z_{ij} + z_{ji} = 1$. Adottando l'approccio big-M il problema può essere formulato come:

$$\begin{array}{lllll} \min & t_{n+1} & & & \\ & t_{n+1} - t_j & \geq -d_j & \forall j & \\ & t_j & \geq r_j + p_j & \forall j & \\ & t_j - t_i & \geq p_j - M\, z_{ij} & \forall i, j,\ i \neq j & \\ & z_{ij} + z_{ji} & = 1 & \forall i, j,\ i < j & \end{array} \tag{14.28}$$

Possiamo senza perdita di generalità assumere che $\min r_j \geq 0$ e aggiungere una variabile fittizia t_0, in modo da riscrivere (14.28) come

$$\begin{array}{lllll} \min & t_{n+1} - t_0 & & & \\ & t_{n+1} - t_j & \geq -d_j & \forall j & \\ & t_j - t_0 & \geq r_j + p_j & \forall j & \\ & t_j - t_i & \geq p_j - M\, z_{ij} & \forall i, j,\ i \neq j & \\ & z_{ij} + z_{ji} & = 1 & \forall i, j,\ i < j & \end{array} \tag{14.29}$$

Per un valore fissato di z, (14.29) è il duale di un problema di cammino massimo in un grafo orientato con nodi $\{0, 1, \ldots, n, n+1\}$. Gli archi sono definiti dal sottografo orientato completo sui nodi $\{1, \ldots, n\}$, più gli archi $(0, j)$ e $(j, n+1)$, $\forall j$. Le lunghezze sono definite da:

$$\delta_{ij} := \begin{cases} r_j + p_j & \text{se } i = 0,\ j \neq n+1 \\ p_j - M\, z_{ij} & \text{se } i \neq 0,\ j \neq n+1 \\ -d_i & \text{se } i \neq 0,\ j = n+1 \end{cases}$$

Il problema (14.29) è ammissibile se e solo se non esistono circuiti orientati di lunghezza positiva. Per costruzione del grafo orientato nessun circuito contiene i nodi 0 e $n+1$. Quindi la condizione si esprime come

$$\sum_{(ij) \in C} \delta_{ij} = \sum_{(ij) \in C} (p_j - M\, z_{ij}) \leq 0 \qquad \forall C$$

ovvero, indicando con $L(C)$ la somma dei p_j lungo il circuito C, e con $z(C)$ la somma delle variabili z lungo il circuito C

$$M\, z(C) \geq L(C) \quad \Longrightarrow \quad z(C) \geq \frac{L(C)}{M} \quad \Longrightarrow \quad z(C) \geq 1$$

dove l'ultima relazione vale in quanto M può essere preso arbitrariamente grande e $z(C)$ è intero. In parole povere la condizione $z(C) \geq 1$ impedisce di assumere precedenze cicliche. Detto ancora in altro modo, se si indica con $G(z)$ il grafo che risulta eliminando gli archi con valore $z_{ij} = 1$, $G(Z)$ deve essere aciclico.

Un cammino semplice $\tilde{P}$ da 0 a t_{n+1} è composto da un arco $(0, i)$, un cammino semplice P in $\{1, \ldots, n\}$ e un arco $(j, n+1)$. Indichiamo con $L(P)$ la somma dei p_j per $j \in P$, $r(P) := r_k$ dove k è il primo nodo di P e $d(P) := d(h)$ dove h è l'ultimo nodo di P. Allora la lunghezza di $\tilde{P}$ si esprime come

$$r(P) + L(P) - d(P) - M\, z(P)$$

e l'esistenza di una schedulazione di valore non peggiore di t^* si esprime dicendo che per ogni cammino P si deve avere

$$r(P) + L(P) - d(P) - M\, z(P) \leq t^*$$

cioè

$$\frac{r(P) + L(P) - d(P) - t^*}{M} \leq z(P)$$

Come nel caso precedente la condizione si semplifica in

$$\begin{aligned} z(P) &\geq 0 \qquad \text{se} \quad r(P) + L(P) - d(P) \leq t^* \\ z(P) &\geq 1 \qquad \text{se} \quad r(P) + L(P) - d(P) > t^* \end{aligned}$$

Tuttavia basta usare la seconda condizione. Infatti l'algoritmo può lavorare in questo modo: fissato un valore di z_{ij} per cui $G(z)$ è aciclico (da fare con una qualsiasi euristica), si calcola il cammino massimo. Sia $\hat{L}$ la sua lunghezza. Ovviamente questa è una limitazione superiore al valore ottimo e, se siamo interessati a trovare soluzioni migliori, dobbiamo fissare $t^* < \hat{L}$.

Allora si cerca una soluzione intera al problema

$$\begin{aligned} z(P) &\geq 1 \qquad \forall P \in \mathcal{P} \\ z(C) &\geq 1 \qquad \forall C \in \mathcal{C} \end{aligned} \tag{14.30}$$

dove $\mathcal{P}$ e $\mathcal{C}$ sono gli insiemi di cammini e circuiti generati dall'algoritmo. Inizialmente $\mathcal{C}$ è vuoto e $\mathcal{P}$ consiste del primo cammino massimo calcolato. Siccome il valore di soglia t^* sarà sempre inferiore al migliore dei cammini massimi generati, si vede che solo la diseguaglianza $z(P) \geq 1$ è necessaria.

Si noti che, aggiungere la diseguaglianza $z(P) \geq 1$ per il cammino P appena generato come cammino più lungo per il valore corrente di z, significa tagliare questa soluzione, od anche, ribaltare l'orientazione di almeno uno degli archi del cammino.

Applichiamo la procedura ai dati dell'esempio 14.16 e inizializziamo con i valori di z corrispondenti all'ordine lineare $1 \to 2 \to 3 \to 4 \to 5 \to 6$. Con questi valori l'algoritmo di programmazione dinamica che calcola il cammino più lungo produce il cammino $P^1 := \{0, 3, 4, 5, 6, 7\}$ (si noti incidentalmente che è lo stesso cammino della figura 14.9). Quindi si deve trovare una soluzione 0-1 della diseguaglianza (nel seguito, per semplicità notazionale, non viene indicato il vincolo $z_{ij} + z_{ji} = 1$ che ovviamente deve essere presente):

$$z_{34} + z_{45} + z_{56} \geq 1 \tag{14.31}$$

Per il calcolo di (14.31) e dei successivi sistemi di diseguaglianze è stato impiegato un pacchetto di programmazione lineare intera. La soluzione z fornita corrisponde all'ordine lineare $1 \to 2 \to 3 \to 4 \to 6 \to 5$. Con questi valori il cammino più lungo è $P^2 := \{0, 3, 4, 6, 5, 7\}$ di valore 3, quindi peggiore di quello iniziale. Si deve allora risolvere:

$$\begin{aligned} z_{34} + z_{45} + z_{56} &\geq 1 \\ z_{34} + z_{46} + z_{65} &\geq 1 \end{aligned}$$

che dà soluzione corrispondente all'ordine lineare $1 \to 2 \to 3 \to 5 \to 4 \to 6$. Il cammino più lungo corrispondente è $P^3 := \{0, 5, 4, 6, 7\}$ di valore 3. Adesso si deve risolvere

$$\begin{aligned} z_{34} + z_{45} + z_{56} &\geq 1 \\ z_{34} + z_{46} + z_{65} &\geq 1 \\ z_{54} + z_{46} &\geq 1 \end{aligned}$$

che dà soluzione corrispondente all'ordine lineare $1 \to 2 \to 3 \to 5 \to 6 \to 4$. Il cammino più lungo corrispondente è $P^4 := \{0, 5, 6, 4, 7\}$ di valore 1. Noi sappiamo già che si tratta dell'ottimo perché l'esercizio è già stato risolto. Dovendo invece continuare con il metodo, prima di scoprire che si tratta dell'ottimo, bisogna generare ancora 22 cammini e 4 circuiti. Il metodo tende quindi a generare molte diseguaglianze. Il pregio è che si producono subito soluzioni buone. Un altro vantaggio è che i problemi di ammissibilità (14.30) non sono difficili. ∎

14.8. Enumerazione e generazione di colonne

Modellare un problema facendo ricorso a matrici di vincoli, che possono essere definite implicitamente in base alla struttura del problema, le cui colonne vengono generate solo nel momento di entrare in base, dà normalmente buoni risultati, soprattutto per ciò che riguarda la limitazione inferiore. Problemi con una forte struttura combinatoria 'travasano', per così dire, questa struttura all'interno della matrice ed è per questo motivo che si ottengono buone limitazioni inferiori.

C'è un problema però non appena si richieda che le variabili siano intere. Bisogna fare in modo che la regola di suddivisione non sia incompatibile con il meccanismo di generazione delle colonne. A titolo esemplificativo si consideri il problema di bin-packing nella formulazione di sezione 7.12, esempio 7.40. Se una variabile x_j, corrispondente ad un certo schema di riempimento, è frazionaria e vale $\bar{x}_j$, sembrerebbe naturale suddividere come $x_j \leq \lfloor \bar{x}_j \rfloor$ e $x_j \geq \lceil \bar{x}_j \rceil$. Imporre la scelta $x_j \geq \lceil \bar{x}_j \rceil$ è facile. Basta preassegnare $\lceil \bar{x}_j \rceil$ volte lo schema di riempimento j e ridefinire i dati del problema. Se la colonna j dovesse venire generata producendo un valore positivo di x_j significa che lo schema di riempimento j deve essere impiegato ulteriormente. Però, come si può imporre la scelta $x_j \leq \lfloor \bar{x}_j \rfloor$? Si supponga di avere introdotto nel modello k vincoli aggiuntivi del tipo $x_j \leq b_j$. Quindi tutte le colonne contengono altri k elementi quasi tutti nulli, tranne qualche 1 per gli schemi di riempimento che sono stati limitati superiormente. Si tenga presente che per tutte le colonne implicite questi elementi aggiuntivi sono tutti nulli. Quindi, se lo schema j è in base ed ha una limitazione superiore, il suo costo ridotto è $\bar{c}_j = 1 - \sum_i u^i a_i^j - u^h = 0$, dove h è l'indice di uno dei k vincoli aggiunti e si ha $\sum_i u^i a_i^j \geq 1$, dalla diseguaglianza primale $u^h \leq 0$, e quindi la colonna potrebbe venire rigenerata producendo una situazione di stallo.

Un modo per aggirare l'ostacolo consiste nella possibilità di generare i primi $k+1$ ottimi di un problema di knapsack. Così, anche se i primi k ottimi corrispondono a schemi già limitati, il $k+1$-esimo, se di costo ridotto negativo può essere generato, altrimenti la soluzione è ottima.

Ottenere i primi k ottimi di un problema di knapsack è effettivamente possibile usando la programmazione dinamica in modo opportuno. Ci limitiamo a fornire le idee essenziali. Siano da calcolare i primi k cammini ottimi di un grafo orientato e aciclico. Si modifichi l'algoritmo di programmazione dinamica nel seguente modo: per ogni nodo viene mantenuta una tabella con k righe e 3 colonne. Nella prima colonna sono contenuti i valori dei migliori k cammini al nodo finora trovati (nell'ordine); nella seconda è contenuto, per ogni cammino, un puntatore al nodo predecessore e nella terza è indicato, per ogni cammino, a quale riga della tabella del nodo predecessore si riferisce il cammino. Quando un nodo ha valori definitivi per la sua tabella, le tabelle dei suoi nodi successori vengono aggiornate attraverso un'operazione di fusione delle due tabelle (quindi di $2\,k$ cammini vengono mantenuti solo i migliori k). La complessità di questo algoritmo è $O(m\,k)$.

14.9. Branch-and-cut

Il termine *branch-and-cut* è stato coniato da Padberg e Rinaldi [1987] per designare una procedura che aggiungeva al consueto metodo branch-and-bound la potenza computazionale dei metodi poliedrali di taglio. Tuttavia la prima applicazione del metodo si deve a Grötschel et al. [1984] nella risoluzione del problema dell'ordine lineare. Riducendo all'essenziale la procedura, in ogni nodo dell'albero di ricerca si generano piani di taglio fino ad ottenere, se possibile, una soluzione intera, altrimenti, delle limitazioni inferiori migliori di quelle disponibili all'origine. Si procede alla suddivisione come nel metodo normale. L'aspetto interessante e molto remunerativo dal punto di vista computazionale è che i piani di taglio generati in un nodo dell'albero di ricerca sono validi in ogni altro nodo.

Per dare un'idea dell'accresciuta potenza di calcolo, le più grandi istanze di TSP risolvibili con i metodi precedenti non superavano i 500 nodi. Con la tecnica branch-and-cut fu possibile risolvere istanze dieci volte più grandi (Padberg e Rinaldi [1991], Grötschel e Holland [1991]). Un codice branch-and-cut, oltre a incorporare procedure di separazione, fa anche normalmente uso di ingegnose tecniche di generazione di colonne o di righe per poter accelerare la computazione.

14.27 Esempio. Applichiamo la tecnica branch-and-cut ad una istanza di TSP relativamente piccola (15 nodi). Con una normale tecnica di enumerazione implicita l'albero di ricerca sarebbe di dimensioni troppo elevate per poter essere disegnato in una figura di testo. Vedremo invece che l'utilizzo sistematico di tecniche poliedrali abbatte la dimensione dell'albero a solo tre nodi.

Scegliamo un'istanza di TSP euclideo con punti generati a caso uniformemente su un quadrato di lato 100 e successivamente arrotondati. Il rilassamento d'interezza con il vincolo di grado ugaule a 2 in ogni nodo e le limitazioni $0 \leq x_e \leq 1$ per ogni arco, produce la soluzione riportata in figura 14.13. Si tratta di un insieme di 3 circuiti. Si identificano in successione le diseguaglianze di sottocircuito violate $x(S) \geq 2$, con $S := \{5, 8, 12\}$ e $S := \{4, 5, 8, 10, 11, 12, 13\}$ (la seconda si ottiene dopo aver introdotto la prima e ottenuto la nuova soluzione). La soluzione che si ottiene è riportata in figura 14.14, dove gli archi tratteggiati indicano valore 1/2.

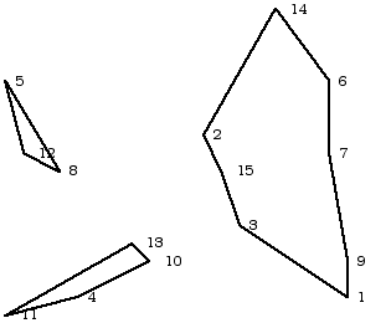

Figura 14.13

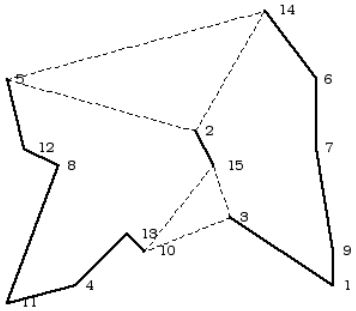

Figura 14.14

Non vi sono più diseguaglianze di sottocircuito violate. Si riconoscono invece due diseguaglianze a pettine violate (in questo caso due semplici germogli). Quindi si inseriscono le diseguaglianze di germoglio $x(H) + \sum_{i=1}^{2k+1} x(T_i) \leq |H| + k$ per i seguenti germogli (in successione):

H	T_1	T_2	T_3
3, 15, 10	10, 13	1, 3	2, 15
2, 5, 14	6, 14	5, 12	2, 15
3, 4, 10, 13, 15	1, 3	4, 11	2, 15
1, 3, 10	3, 15	10, 13	1, 9

Dopo l'inserzione delle prime due diseguaglianze di germoglio si ottiene la soluzione in figura 14.15. La soluzione che si ottiene alla fine dell'inserzione delle quattro diseguaglianze è riportata in figura 14.16. Gli archi tratteggiati indicano valori uguali a 1/4, 1/2 o 3/4.

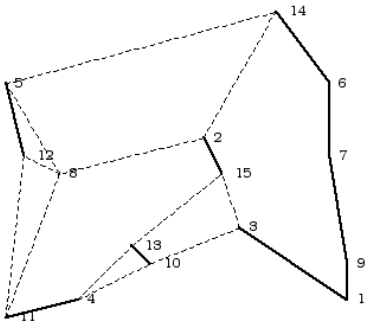

FIGURA 14.15

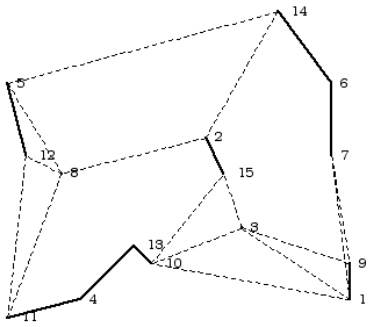

FIGURA 14.16

Non riconoscendo diseguaglianze di sottocircuito o di pettine violate si decide di operare la suddivisione. In questo caso si opera 'a vista' decidendo che l'arco $(10, 15)$ di valore $x_{10,15} = 0.25$ probabilmente non appartiene all'ottimo, data la sua posizione. Quindi si suddivide ponendo in un caso $x_{10,15} := 0$ e nell'altro $x_{10,15} := 1$. L'attuale soluzione frazionaria ha valore 380.75.

Ponendo $x_{10,15} := 0$ si ottiene la soluzione di figura 14.17, cioè un circuito di valore 382. Si noti che, se esistono soluzioni migliori, queste devono valere 381, e quindi la soluzione trovata ha un'alta probabilità di essere l'ottimo.

Ora non resta che provare l'altro ramo dell'albero di ricerca ponendo $x_{10,15} := 1$. La soluzione che si ottiene questa volta è un un insieme di circuiti di valore 385 (figura 14.18). Pertanto è una soluzione dominata dalla precedente, che risulta allora l'ottimo del problema.

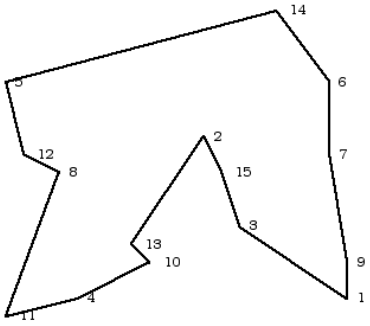

FIGURA 14.17

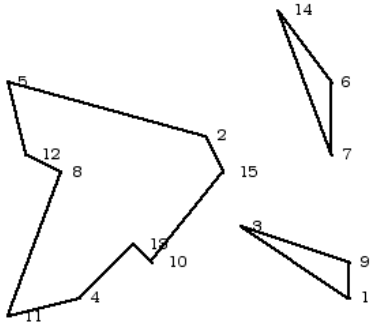

FIGURA 14.18

14.28 Esempio. Applichiamo la tecnica branch-and-cut al problema del massimo insieme stabile. Sia data l'istanza in figura 14.19. Risolvendo il duale del problema di programmazione lineare $\max\{\sum_i x_i : x_i + x_j \leq 1,\ (i,j) \in E,\ x_i \geq 0\}$ si ottiene la soluzione frazionaria $x_i = 1/2,\ \forall i$. Aggiungendo 16 diseguaglianze di cricca (da tre elementi) e le due diseguaglianze di buca dispari $\{1,2,3,8,7\}$ e $\{7,8,14,17,19,18,12\}$, si ottiene ancora una soluzione frazionaria di valore 8.2 (figura 14.20, dove i nodi, tranne i nodi 4, 11 e 15 di valore 0, sono disegnati di grandezza proporzionale al valore della soluzione: i valori sono 1/5, 2/5, 3/5 e 1). A questo punto non vengono più identificate buche violate e quindi bisogna suddividere. Una delle due soluzioni con il minimo valore frazionario è $x_6 = 1/5$. Suddividendo come $x_6 := 0$ si ottiene subito la soluzione intera in figura 14.21 di valore 8 (e quindi ottima a questo punto). L'albero di ricerca termina immediatamente anche con l'altro valore della suddivisione $x_6 := 1$ che porta alla soluzione intera in figura 14.22 (non ottima).

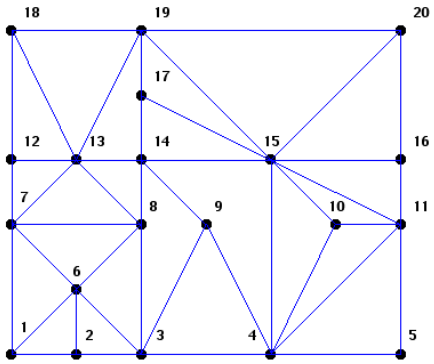

Figura 14.19

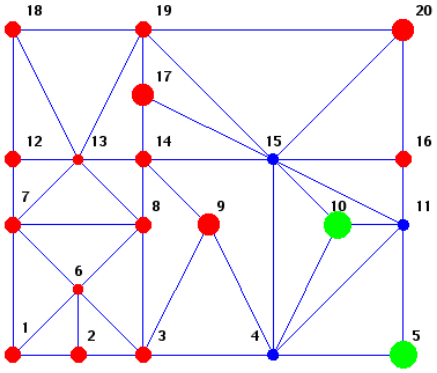

Figura 14.20

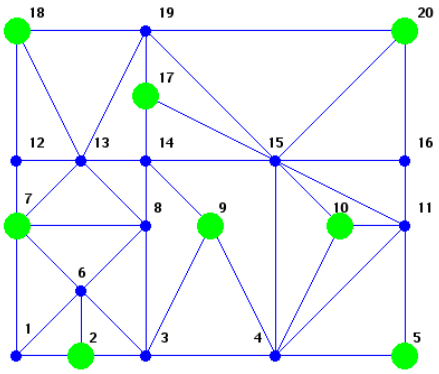

Figura 14.21

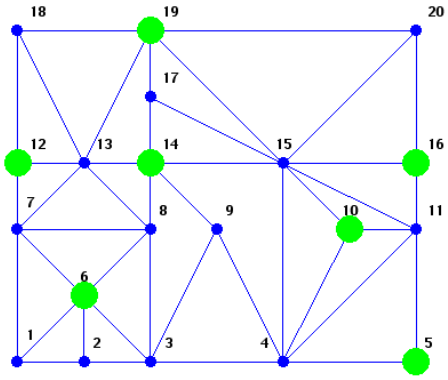

Figura 14.22