Programmazione lineare intera
Introduzione

Giovanni Righini

6 agosto 2010

1 Introduzione

Dal punto di vista algoritmico la Programmazione Lineare Intera e la Programmazione Lineare 0-1, cioè con variabili binarie, sono spesso trattate insieme: in effetti hanno la stessa complessità e richiedono algoritmi simili. Tuttavia dal punto di vista modellistico esiste una profonda differenza tra i problemi in cui le variabili rappresentano *quantità* e quelli in cui le variabili rappresentano *scelte logiche* di tipo "vero/falso".

Poiché il taglio di questo testo per gli insegnanti delle scuole superiori è essenzialmente modellistico e non algoritmico, consideriamo qui tutti i problemi di PLI semplicemente come "varianti più difficili" di problemi di PL e ci concentriamo invece sui problemi con variabili binarie.

L'unica cosa che distingue dal punto di vista modellistico i problemi di PLI da quelli di PL è la risposta alla domanda se le quantità da scegliere possano assumere qualisiasi valore oppure debbano per forza essere rappresentate da numeri interi. Dal punto di vista della soluzione degli esercizi con il Solver di Excel, ciò che cambia è solo l'uso di una semplice opzione che permette di dichiarare le variabili come intere. L'effetto che è possibile osservare, rispetto alla soluzione dei modelli di PL, è un aumento del tempo di calcolo ed un peggioramento del valore ottimo. Queste osservazioni consentono eventualmente di introdurre in modo semplice ed intuitivo il concetto di *rilassamento continuo*.

I problemi di Programmazione Lineare 0-1 sono molto istruttivi dal punto di vista didattico poiché richiedono di esprimere condizioni logiche sotto forma di equazioni e disequazioni su variabili binarie, il che sulle prime non è banale né intuitivo. D'altra parte una volta appresa questa tecnica di modellizzazione, è possibile apprezzare la straordinaria potenza espressiva della Programmmazione Matematica.

A parità di numero di vincoli e di variabili un problema di PLI o PL0-1 può è in generale molto più difficile da risolvere per via algoritmica rispetto ad un problema di PL con variabili continue. È facile verificare con qualche piccolo esempio che la soluzione ottima di un problema di ottimizzazione discreta (intera o binaria) non sempre si ottiene arrotondando a valori interi la soluzione ottima dello stesso problema risolto nel continuo.

Per questo motivo esiste una vastissima area di ricerca alla frontiera tra l'Informatica e la Ricerca Operativa, riguardante la realizzazione di algoritmi *euristici* e *approssimanti*, che non garantiscono l'ottimalità della soluzione trovata ma proprio per questo hanno un tempo di esecuzione molto inferiore a quello di un algoritmo di ottimizzazione esatto. Questo può essere un fertilissimo terreno d'incontro tra la programmazione e la Ricerca Operativa, nel senso che i problemi di ottimizzazione si prestano benissimo a sperimentare diversi algoritmi e a confrontarli in termini di tempo di calcolo e qualità della soluzione trovata. Attività di grande valore didattico possono essere realizzate nei laboratori informatici delle scuole con pochissimi prerequisiti: non serve di più dei primi rudimenti di programmazione per realizzare innumerevoli algoritmi dalla complessità estremamente variabile, per problemi di ottimizzazione combinatoria (che hanno il vantaggio di consentire di lavorare sempre con numeri interi, evitando molti problemi di tipo numerico).

2 Richiami teorici

Per risolvere all'ottimo garantito problemi lineari con variabili discrete (intere o binarie), i solutori eseguono algoritmi di tipo *branch-and-bound*.

Questi algoritmi devono il loro nome alle operazioni di *branching* e *bounding* che vengono ripetutamente eseguite. Il *branching* consiste nella scomposizione di un problema in sottoproblemi più vincolati, ottenuti ad esempio fissando il valore di una variabile. Esistono molte diverse politiche di *branching* e i solutori più sofisticati consentono all'utente di scegliere quale usare. Una molto semplice per i problemi di programmazione 0-1 consiste nel generare due sotto-problemi, scegliendo una variabile x e fissandola in un caso a 0 e nell'altro a 1.

Iterando ricorsivamente questa operazione si genera un *albero di ricerca*, le cui foglie sono i sottoproblemi "banali". Un sottoproblema è banale quando si dimostra che non ammette soluzione, o quando è nota la sua soluzione ottima (magari perché è l'unica possibile).

Per evitare la generazione di un numero eccessivo di sottoproblemi, ad ogni sottoproblema viene associato un *bound*: si tratta di un *lower bound* se il problema è di minimizzazione, di un *upper bound* se è di massimizzazione. Tale *bound* è un limite al valore ottimo che quel sottoproblema può avere nel migliore dei casi. Se tale limite non è migliore del valore di una soluzione ammissible già nota $\overline{x}$, il sottoproblema può essere cancellato, poiché è garantito che la sua soluzione ottima non può essere migliore di $\overline{x}$.

Esistono diversi modi per calcolare un *bound* da associare ad ogni sottoproblema: nei solutori *general purpose* di solito il *bound* è dato dal valore ottimo del rilassamento continuo, che viene calcolato con l'algoritmo del simplesso.

L'albero di ricerca può essere esplorato con diverse strategie: anche questa scelta può influenzare pesantemente le prestazioni dell'algoritmo. I solutori più sofisticati consentono all'utente di scegliere tra alcune politiche di esplorazione predefinite.