Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Triennale in Ingegneria Informatica

Esperimenti di programmazione lineare intera per problemi di vertex cover

Relatore
Prof. Domenico Salvagnin

Laureando
Giacomo Camposampiero

Anno Accademico 2020/2021

Padova, 19 luglio 2021

Text.

Indice

Capitolo 1

Introduzione

La ricerca operativa, dall'inglese *operational research*, è una disciplina scientifica relativamente giovane, nata con lo scopo di fornire strumenti matematici di supporto ad attività decisionali in cui occorre gestire e coordinare attività e risorse limitate. La ricerca operativa permette infatti di trovare, mediante la formalizzazione matematica di un problema, una soluzione ottima o ammissibile (quando possibile) al problema stesso. Costituisce di fatto un approccio scientifico alla risoluzione di problemi complessi, che ha trovato grande applicazione in moltissimi ambiti, non ultimo quello industriale.

Una delle diverse branche che compongono la ricerca operativa è l'ottimizzazione. Quest'ultima si occupa principalmente di problemi che comportano la minimizzazione o la massimizzazione di una funzione, detta funzione obbiettivo, sottoposta a un dato insieme di vincoli. Un problema di ottimizzazione può essere formulato come

$$\begin{aligned} &\min(or\ \max) f(x) \\ &S \\ &x \in D \end{aligned} \tag{1.1}$$

dove $f(x)$ è una funzione a valori reali nelle variabili x, D è il dominio di x e S un insieme finito di vincoli. In generale, x è una tupla $(x_1, ..., x_n)$ e D è un prodotto cartesiano $D_1 \times ... \times D_n$, e vale $x_j \in D_j$.

Un problema nella forma (1.1) risulta essere intrattabile, ovvero non esistono algoritmi efficienti (o non esiste proprio alcun algoritmo) per la sua risoluzione. Si rende quindi necessario considerare dei casi particolari di questa formulazione, i quali possiedono determinate proprietà che possono essere sfruttate nella definizione di algoritmi ad-hoc.

1.1 Programmazione lineare intera

Uno dei casi particolari della formulazione generale (1.1) a cui si è precedentemente fatto riferimento viene trattato dalla programmazione lineare intera. Un problema di programmazione lineare intera consiste nella minimizzazione (o massimizzazione) di una funzione lineare soggetta ad un numero finito di vincoli lineari, con in aggiunta il vincolo che alcune delle variabili del problema debbano assumere valori interi. In generale, il problema può quindi essere riformulato come

$$\begin{aligned} &\min cx \\ &a_i x \sim b_i \qquad i = 1, ..., m \\ &l_j \leq x_j \leq u_j \qquad j = 1, ..., n = N \\ &x_j \in \mathbb{Z} \qquad \forall j \in J \subseteq N = 1, ..., n \end{aligned}$$

Se $J = N$ si parla di programmazione lineare intera pura, altrimenti di programmazione lineare intera mista (o MIP, dall'inglese *Mixed Integer Programming*).

La programmazione lineare intera restringe quindi notevolmente il tipo di vincoli a disposizione nel processo di formalizzazione matematica del problema, determinando una maggior difficoltà in fase di modellazione del problema. Tuttavia, questo non comporta eccessive restrizioni, almeno per la MIP, sui tipi di problemi che possono essere formulati secondo questo paradigma. Alcuni esempi di problemi risolvibili mediante MIP sono *knapsack*, problemi di *scheduling*, *facility location* e, naturalmente, *vertex covering* (di cui si discuterà più approfonditamente di seguito nella Sezione **TODO**).

Allo stesso tempo, l'introduzione dei vincoli di linearità ed interezza comporta notevoli vantaggi nella definizione ed implementazione di algoritmi risolutivi.

1.1.1 Algoritmo Branch and Bound

L'algoritmo branch-and-bound (B&B) è un algoritmo di ottimizzazione generica basato sull'enumerazione dell'insieme delle soluzioni ammissibili di un problema di ottimizzazione combinatoria, introdotto nel 1960 da A. H. Land e A. G. Doig [1].

Questo algoritmo permette di gestire il problema dell'esplosione combinatoria mediante il *pruning* di intere porzioni dello spazio delle soluzioni, che può essere effettuato quando si riesce a dimostrare che queste ultime non contengono soluzioni migliori di quelle note. Branch and bound implementa inoltre una strategia *divide and conquer*, che permette di partizionare lo spazio di ricerca e di risolvere ognuna di esse separatamente. Viene riportata di seguito una breve descrizione dell'algoritmo.

Sia F l'insieme delle soluzioni ammissibili di un problema di minimizzazione (simmetrico nel caso della massimizzazione, a meno di un cambio di segno della funzione obiettivo), $c : F \to \mathbb{R}$ la funzione obiettivo e $\bar{x} \in F$ una soluzione ammissibile nota, generata mediante euristiche o mediante assegnazioni casuali. Sia $z = f(\bar{x})$ il costo di tale soluzione nota, anche detto *incumbent*, che rappresenta per sua natura un limite superiore al valore della soluzione ottima.

L'algoritmo Branch and Bound esegue un'iniziale fase di *bounding* in cui uno o più vincoli del problema vengono rilassati, allargando di conseguenza l'insieme delle possibili soluzioni $G \supseteq F$. La soluzione di questo rilassamento, se esiste, rappresenta un *lower bound* alla soluzione ottima del problema iniziale. Se la soluzione di tale rilassamento appartiene a F o ha costo uguale all'attuale *incumbent*, allora l'algoritmo termina, in quanto si è trovata una soluzione ottima del problema. Se, al contrario, il rilassamento risulta essere impossibile, possiamo anche in questo caso terminare la ricerca di una soluzione e concludere che anche il problema di partenza è impossibile.

Nel caso in cui invece una soluzione al rilassamento esiste ma non è contenuta nell'insieme delle soluzioni ammissibili F, l'algoritmo procede con l'identificare una separazione F^* di F, ossia un insieme finito di sottoinsiemi tali che

$$\bigcup_{F_i \in F^*} F_i = F$$

Questa fase, detta di *branching*, è giustificata dal fatto che la soluzione ottima dell'intero problema è data dalla minima tra le soluzioni delle varie separazioni $F_i \in F^*$. F^* è spesso, anche se non necessariamente, una partizione dell'insieme iniziale F. A questo punto, tutti i figli di F vengono aggiunti alla coda dei sotto-problemi da processare.

L'algoritmo procede quindi con il selezionare un sotto-problema P_i dalla coda un rilassamento. A questo punto, ci sono quattro possibili casi

- Se si trova una soluzione ammissibile ($\in F$) migliore dell'attuale *incumbent* $\bar{x}$, allora quest'ultimo viene sostituito dalla soluzione trovata e si procede con la soluzione di un altro sotto-problema P_i.
- Se il rilassamento del sotto-problema non ammette soluzione, allora si smette di esplorare l'intero sotto-albero a lui associato nello spazio di ricerca (*pruning by infeasibility*).
- Altrimenti, si confronta la soluzione trovata con il valore corrente dell'*upper-bound* z; se quest'ultimo è minore della soluzione trovata, è possibile anche in questo caso

smettere di esplorare il sotto-albero associato al sotto-problema corrente, in quanto non può portare ad una soluzione migliore di quella che già si conosce (*pruning by optimality*, ricordando che la soluzione del rilassamento costituisce un *lower bound* alla soluzione del problema originale).

- Infine, se non è stato possibile scartare o concludere la visita del sotto-albero associato a P_i in alcun modo, è necessario eseguire nuovamente il *branching*, aggiungendo i nuovi sotto-problemi alla lista dei sotto-problemi da processare.

L'algoritmo prosegue nella selezione di sotto-problemi finché la lista di questi ultimi non si svuota. Quando ciò avviene, la soluzione rappresentata dall'attuale *incumbent* è la soluzione ottima al problema iniziale.

Quella appena descritta rappresenta una formulazione generica dell'algoritmo Branch and Bound. Questa formulazione può essere tuttavia specializzata nella risoluzione di problemi MIP con relativa semplicità, agendo sulle condizioni che regolano *bounding* e *branching*. Nel primo caso la scelta più diffusa consiste nel rilassamento del vincolo di interezza. Se la soluzione del rilassamento non è intera, una possibile separazione in sotto-problemi può essere fatta considerando la partizione

$$x_j \leq \left\lfloor x_j^* \right\rfloor \vee x_j \geq \left\lceil x_j^* \right\rceil$$

Per costruzione, ogni soluzione trovata dall'algoritmo è quindi migliore dell'*incumbent* e, di conseguenza, l'andamento dell' *upper bound* del problema è strettamente decrescente. I *lower bound* non hanno, al contrario degli *upper bound* , valenza globale. Derivare un *lower bound* globale è comunque possibile considerando il minimo tra tutti i *lower bound* dei sotto-problemi ancora aperti. Avendo a disposizione in ogni momento entrambi i *bound* del problema, è possibile valutare la bontà della soluzione provvisoria in qualsiasi momento.

1.1.2 Algoritmo Branch and Cut

Capitolo 2

Presentazione delle sperimentazioni

Capitolo 3

Risultati sperimentali

Capitolo 4

Conclusioni

Bibliografia

[1] A. H. Land e A. G. Doig. «An Automatic Method of Solving Discrete Programming Problems». In: *Econometrica* 28.3 (1960), pp. 497–520. ISSN: 00129682, 14680262. URL: http://www.jstor.org/stable/1910129.