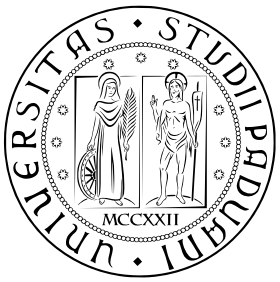

Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Triennale in Ingegneria Informatica

Esperimenti mip per una classe di problemi di assegnamento quadratico

Relatore
Prof. Domenico Salvagnin

Laureando
Mattia Toffolon

Anno Accademico 2022/2023

Padova, XX luglio 2023

A mamma, papà e Cristian

Indice

Introduzione

La ricerca operativa, in inglese *operational research*, è una branca della matematica applicata che si occupa dell'analisi e risoluzione di complessi problemi decisionali mediante modelli matematici e metodi quantitativi. Lo scopo di questa disciplina è dunque quello di fornire supporto nell'attività decisionale in cui occorre gestire e coordinare risorse limitate sottoforma di soluzione al problema ottima, quando possibile, o ammissibile. Essa rappresenta un approccio scientifico alla risoluzione di problemi complessi che ha trovato largo successo in molti settori come l'economia, l'informatica e l'ingegneria.

Una nota branca della ricerca operativa è l'ottimizzazione. Essa si fonda sulla risoluzione di problemi di massimizzazione o minimizzazione di una data funzione, detta funzione obbiettivo, le cui variabili sono legate fra loro tramite un dato insieme di vincoli.

Questo studio verte su una particolare classe di problemi di ottimizzazione detta di assegnamento quadratico, in inglese *quadratic assignment problem* (QAP). In particolare, è stato analizzato l'andamento della complessità di risoluzione delle istanze del problema al variare della sua dimensione e di altri parametri utili alla sua generazione.

La stesura di questo elaborato è stata articolata in quattro capitoli. Inizialmente verranno brevemente esposti alcuni concetti teorici utili alla comprensione degli esperimenti svolti. Successivamente si proseguirà con le metodologie adottate per la realizzazione e risoluzione delle istanze e la presentazione dei risultati. Infine, verrà riportata una breve trattazione relativa alle conclusioni del lavoro svolto.

Tutto il codice sviluppato relativamente a questo studio è stato scritto in Python [1], linguaggio di programmazione *general-purpose* di alto livello che ha permesso di gestire ognuna delle diverse fasi in cui si è articolato lo svolgimento degli esperimenti, dalla generazione delle istanze all'elaborazione grafica dei risultati. Il codice sorgente, così come tutti i risultati delle elaborazioni, sono liberamente consultabili online nella *repository* utilizzata per gestire il controllo di versione del progetto [2].

Capitolo 1

Nozioni preliminari

In questo primo capitolo sono stati riportati alcuni concetti che rappresentano le fondamenta teoriche del lavoro svolto. Come prima cosa è stato definito formalmente il concetto di problema di ottimizzazione. Successivamente si è proseguito con una coincisa trattazione relativa a due importanti algoritmi usati nella risoluzione di problemi di ottimizzazione: *branch-and-bound* e *branch-and-cut*. Infine, è stata esposta la definizione formale di problema di assegnamento quadratico sul quale questo lavoro è stato basato. Molte delle definizioni in questo capitolo sono ispirate alle dispense del corso di *Modelli e Software per l'Ottimizzazione Discreta*, tenuto dal professore Domenico Salvagnin [3][4].

1.1 Problema di ottimizzazione

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 $x_j \in D_j$.

In generale, un problema di ottimizzazione nella forma (1.1) è intrattabile, nel senso che non esistono algoritmi efficienti (o addirittura algoritmi) per la sua risoluzione. Si rende pertanto necessario considerare casi particolari di questa formulazione che presentano struttura e proprietà particolari da poter sfruttare.

1.2 Programmazione lineare intera

Uno dei casi particolari della formulazione generale di problema di ottimizzazione (1.1) è quello della 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:

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

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 la tipologia di vincoli a disposizione nel processo di formalizzazione matematica del problema, determinando una maggior difficoltà in fase di modellazione. Tuttavia, nel caso della MIP, questa restrizione non comporta un'eccessiva limitazione sul tipo di problemi formulabili secondo questo paradigma. Alcuni esempi classici di problemi risolvibili mediante MIP sono *knapsack*, problemi di *scheduling*, *minimum vertex cover* e, naturalmente, *quadratic assignment problem*. Inoltre, l'introduzione dei vincoli di linearità ed interezza comporta notevoli vantaggi nella definizione ed implementazione di algoritmi di risoluzione, due dei quali sono stati descritti più nel dettaglio nelle seguenti sezioni.

1.2.1 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 [5].

Questo algoritmo permette di gestire il problema dell'esplosione combinatoria scartando intere porzioni dello spazio delle soluzioni attraverso operazioni di *pruning*. Ciò risulta possibile quando si riesce a dimostrare che questi sottospazi non possono contenere soluzioni migliori di quelle note. Branch-and-bound implementa inoltre una strategia *divide and conquer*, che permette di ottenere la soluzione al problema ricombinando quelle

relative a partizioni del problema stesso. Viene riportata di seguito una breve descrizione dell'algoritmo.

Sia F l'insieme delle soluzioni ammissibili di un problema di minimizzazione (oppure di 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. Il costo di tale soluzione nota $z = f(\bar{x})$, detto *incumbent*, rappresenta per sua natura un limite superiore al valore della soluzione ottima.

L'algoritmo branch-and-bound prevede una fase iniziale 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*, l'algoritmo termina in quanto si è trovata una soluzione ottima del problema. Se il rilassamento dovesse risultare impossibile, è possibile anche in questo caso terminare la ricerca di una soluzione in quanto si può concludere che anche il problema di partenza è impossibile.

Invece, nel caso in cui 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 tale che:

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

Tale fase, detta di *branching*, è giustificata dal fatto che la soluzione ottima del problema è data dalla minima tra le soluzioni delle varie separazioni $F_i \in F^*$ dette figli di F.

$$min\left\{c\left(x\right) \mid x \in F\right\} = min\left\{min\left\{c\left(x\right) \mid x \in F_i\right\} \mid F_i \in F^*\right\}$$

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 sottoproblemi da processare.

L'algoritmo procede quindi con il selezionare un sottoproblema P_i dalla coda e risolverne un rilassamento. Si possono presentare quattro casi differenti:

- Se si trova una soluzione $\in F$ migliore dell'attuale incumbent, quest'ultimo viene sostituito dalla soluzione trovata e si procede con lo studio di un altro sottoproblema.
- Se il rilassamento del sottoproblema non ammette soluzione, allora si smette di esplorare l'intero sottoalbero a lui associato nello spazio di ricerca (*pruning by infeasibility*).

- Altrimenti, si confronta la soluzione trovata con il valore corrente dell'*upper-bound* dato dall'incumbent; se quest'ultimo è minore della soluzione del rilassamento trovata, è possibile anche in questo caso smettere di esplorare il sottoalbero associato al sottoproblema corrente, in quanto non può portare ad una soluzione migliore di quella già nota (*pruning by optimality*).
- Infine, se non è stato in alcun modo possibile scartare o concludere l'esplorazione del sottoalbero associato a P_i, è necessario eseguire nuovamente il *branching*, aggiungendo i nuovi sottoproblemi alla lista dei sottoproblemi da processare.

L'algoritmo prosegue processando sottoproblemi finché la lista di questi 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 B&B. Questa formulazione può essere tuttavia specializzata nella risoluzione di problemi MIP in maniera immediata, agendo sulle condizioni che regolano *bounding* e *branching*. Per quanto riguarda il primo, la scelta più diffusa consiste nel considerare il rilassamento lineare dei sottoproblemi, rilassando dunque il vincolo di interezza. Se la soluzione del rilassamento non è intera, una possibile separazione in sottoproblemi può essere effettuata considerando la partizione:

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

Infine, è importante notare come, per costruzione, ogni soluzione trovata dall'algoritmo è migliore dell'incumbent e, di conseguenza, l'andamento dell'upper bound del problema decresce fino al raggiungimento della soluzione ottima. A differenza degli upper bound, i lower bound dei singoli sottoproblemi non hanno invece valenza globale. Nonostante ciò, è comunque possibile derivare un lower bound globale considerando il minimo tra tutti i lower bound dei sottoproblemi ancora aperti. Avere a disposizione in qualsiasi momento entrambi i bound del problema, permetta quindi di valutare la bontà della soluzione provvisoria in ogni momento.

1.2.2 Algoritmo Branch and Cut

L'algoritmo *branch-and-cut* (B&C) rappresenta una versione migliorata dell'algoritmo branch-and-bound, introdotta nel 1987 da M. Padberg e G. Rinaldi [6] e ideata appositamente per la risoluzione di problemi MIP.

L'algoritmo branch-and-cut è un ibrido tra branch-and-bound, trattato nella sezione precedente, e un algoritmo a piani di taglio puro, in cui la soluzione è viene ottenuta mediante raffinazioni progressive dello spazio delle soluzioni attraverso la progressiva aggiunti di vincoli. Queste due tecniche si rafforzano a vicenda, contribuendo al raggiungimento di prestazioni complessive superiori a quelle che otterrebbe ciascuna di esse singolarmente.

L'idea alla base di questo algoritmo è quella di "rafforzare" la formulazione associata al rilassamento lineare di ogni sottoproblema mediante la generazione di piani di taglio. I vantaggi a livello risolutivo sono molteplici, tra cui una maggior probabilità di ottenere soluzioni intere del rilassamento lineare o, in alternativa, di ottenere lower bound più stretti e pertanto più efficienti in fase di pruning.

Nonostante l'idea alla base di questo approccio sia relativamente semplice, l'implementazione dell'algoritmo B&C è tutt'altro che banale e richiede l'esistenza di procedure efficienti per la risoluzione del seguente problema di *separazione*: data una soluzione frazionaria x^*, trovare una diseguaglianza valida $\alpha^T x \leq \alpha_0$, se esiste, violata da x^*, cioè tale che $\alpha^T x^* > \alpha_0$.

1.3 Quadratic assignment problem

I QAP (Quadratic assignment problem), conosciuti anche come *problemi di assegnamento quadratico*, si fondano sulla ricerca di un posizionamento ottimale di un dato insieme di unità anche dette *facilities*, in riferimento ai *facility location problems* di cui i QAP sono una branca, in un dato insieme di posizioni anche dette *locations*. Come provato da S. Sahni e T. Gonzalez nel 1976 [7], un problema di assegnamento quadratico è di tipo NP-difficile, ciò significa che non esistono algoritmi in grado di trovarne una soluzione in un tempo polinomiale. Inoltre, può essere risolto all'ottimo solo per instanze particolarmente piccole.

Nello specifico, il problema può essere presentato come segue: n unità devono essere assegnate ad n posizioni differenti, sapendo che a_{ij} è il flusso di informazioni che deve essere trasferito dall'unità i all'unità j e che la distanza tra le posizioni r ed s è pari a b_{rs}, si vuole trovare l'assegnamento delle unità nelle posizioni ottimo, ovvero quello che minimizza la somma dei prodotti flusso $\times$ distanza. Matematicamente, il problema si può formulare nel seguente modo:

$$\min_{\pi \in P(n)} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} b_{\pi_i \pi_j}$$

dove $A = (a_{ij})$ e $B = (b_{rs})$ sone due matrici $n \times n$, $P(n)$ è l'insieme di tutte le possibili permutazioni di $\{1, 2, ..., n\}$ e π_i indica la posizione dell'unità i nella permutazione $\pi \in P(n)$.

Capitolo 2

Approccio sperimentale

Prova

Bibliografia

[1] Guido Van Rossum e Fred L Drake Jr. *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

[2] `https://github.com/mattia-toffolon/QAP`.

[3] Domenico Salvagnin. *Cenni di Programmazione Lineare Intera*. Università degli Studi di Padova, 2020.

[4] Domenico Salvagnin. *Introduzione all'ottimizzazione discreta*. Università degli Studi di Padova, 2018.

[5] 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`.

[6] M. Padberg e G. Rinaldi. «Optimization of a 532-city symmetric traveling salesman problem by branch and cut». In: *Operations Research Letters* 6.1 (1987), pp. 1–7. ISSN: 0167-6377. DOI: `https://doi.org/10.1016/0167-6377(87)90002-2`. URL: `https://www.sciencedirect.com/science/article/pii/0167637787900022`.

[7] Sartaj Sahni e Teofilo Gonzalez. «P-complete approximation problems». In: *Journal of the ACM* 23.3 (1976), pp. 555–565. DOI: `10.1145/321958.321975`. URL: `https://www.researchgate.net/publication/220432228_P-complete_approximation_problems_J_ACM_233_555-565`.