

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA TRIENNALE IN INGEGNERIA INFORMATICA

TECNICHE MIP AVANZATE PER UNA CLASSE DI  
PROBLEMI DI ASSEGNAMENTO QUADRATICO

*Relatore*

Prof. Domenico Salvagnin

*Laureando*

Alessandro Dario

Anno Accademico 2023/2024

Padova, 17 luglio 2024



# Abstract

Nella ricerca operativa, il problema dell'assegnamento quadratico (QAP) è una classe di problemi difficile da risolvere all'ottimo sia in teoria che in pratica. Una sottoclasse di istanze, dette *TaiC*, sono strutturalmente più semplici ma richiedono comunque grande enumerazione.

Il problema è affrontato mediante la programmazione lineare intera: si presenta un modello alternativo per la linearizzazione del problema, si studiano tecniche di generazione di tagli e strategie di branching che ne migliorano le prestazioni. Si confrontano sperimentalmente i risultati dei due modelli e delle diverse migliori individuate.



# Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Programmazione lineare . . . . .	1
1.1.1	Simpleso . . . . .	1
1.2	Programmazione lineare intera . . . . .	2
1.2.1	Branch & Bound . . . . .	2
1.2.2	Branch & Cut . . . . .	3
1.3	Problema di assegnamento quadratico . . . . .	4
1.3.1	Istanze TaiC . . . . .	5
<b>2</b>	<b>Sviluppo del modello</b>	<b>7</b>
2.1	Formulazione del modello . . . . .	7
2.1.1	Modello quadratico generale . . . . .	7
2.1.2	Modello quadratico TaiC . . . . .	8
2.1.3	Linearizzazione intuitiva . . . . .	8
2.1.4	Linearizzazione KB . . . . .	9
2.1.5	Miglioramento del lowerbound . . . . .	10
2.2	Separazione dei tagli . . . . .	11
2.2.1	Costanti locali . . . . .	11
2.2.2	Tagli fra coppie libera-fissa . . . . .	12
2.2.3	Tagli fra coppie libera-libera . . . . .	13
2.2.4	Margine di violazione . . . . .	14
2.3	Priorità di branching . . . . .	14
2.3.1	Scarto tra i bound . . . . .	15
2.3.2	Scarto con il costo quadratico . . . . .	15
2.3.3	Ordine di esplorazione . . . . .	16
<b>3</b>	<b>Risultati sperimentali</b>	<b>17</b>
3.1	Metodologia . . . . .	17

---

3.1.1	Istanze . . . . .	17
3.1.2	Software . . . . .	18
3.1.3	Ambiente . . . . .	18
3.2	Confronti statistici . . . . .	19
3.2.1	Il modello KB base . . . . .	19
3.2.2	Varianti al modello KB . . . . .	20
3.2.3	Implementazione ad-hoc . . . . .	21
3.2.4	Dati complessivi . . . . .	23
<b>4</b>	<b>Conclusioni</b>	<b>25</b>
4.1	Risultati . . . . .	25
4.2	Possibili miglioramenti . . . . .	25
4.3	Possibili sviluppi . . . . .	26
	<b>Bibliografia</b>	<b>27</b>

# Capitolo 1

## Introduzione

Questo capitolo fornisce un'introduzione generale ai concetti di programmazione lineare (LP), lineare intera (MIP), e agli algoritmi utili alla loro risoluzione. Vengono inoltre introdotti i problemi di assegnamento quadratico trattati dal presente elaborato.

### 1.1 Programmazione lineare

Il paradigma della programmazione lineare consente la modellazione di un problema tramite un numero finito di variabili e vincoli. Le variabili hanno come dominio un intervallo di  $\mathbb{R}$ , i vincoli sono equazioni o disequazioni lineari sulle variabili. Risolvere il problema all'ottimo vuol dire trovare una soluzione che rispetti i vincoli e sia dimostrata minimo globale per una certa funzione lineare. In generale un LP può essere scritto come:

$$\begin{aligned} \min \quad & cx \\ a_i x & \sim b_i & i = 1, \dots, m \\ l_j \leq x_j & \leq u_j & j = 1, \dots, n \end{aligned} \tag{1.1}$$

dove  $\sim \in \{\leq, \geq, =\}$ , il *lowerbound*  $l_j \in \mathbb{R} \cup \{-\infty\}$ , l'*upperbound*  $u_j \in \mathbb{R} \cup \{+\infty\}$ . [1]

#### 1.1.1 Simplexso

Richiedere la linearità dei vincoli e della funzione obiettivo rende un LP poco espressivo, ma ne consente una risoluzione efficiente in teoria e in pratica. L'algoritmo del simplexso, ideato nel 1947 da G. Dantzig [2], è ad oggi largamente utilizzato nella risoluzione di problemi lineari.

Essendo i vincoli lineari e i domini intervalli continui, lo spazio delle soluzioni forma geometricamente un poliedro convesso. L'algoritmo parte da un vertice qualsiasi di questo poliedro e si muove iterativamente verso vertici di costo progressivamente minore: essendo la funzione di costo anch'essa lineare, il minimo trovato è così sempre globale. Un poliedro ha in generale un numero di vertici esponenziale: la loro esplorazione porta nei casi degeneri a complessità temporali pessime, tuttavia esistono nella pratica tecniche per mitigare queste eventualità.

Nonostante esistano algoritmi che garantiscono tempi polinomiali, il semplice è spesso scelto perché consente di risolvere velocemente un grande numero di LP strutturalmente simili fra loro. Questa abilità viene sfruttata dalle tecniche presentate in seguito.

## 1.2 Programmazione lineare intera

È possibile modellare una gamma più vasta di problemi ammettendo come ulteriore vincolo per un sottoinsieme di variabili la loro restrizione a un dominio intero. Scriviamo un generico problema MIP come:

$$\begin{aligned}
 & \min cx \\
 & a_i x \sim b_i \quad i = 1, \dots, m \\
 & l_j \leq x_j \leq u_j \quad j = 1, \dots, n = N \\
 & x_j \in \mathbb{Z} \quad \forall j \in J \subseteq N = 1, \dots, n
 \end{aligned} \tag{1.2}$$

dove  $J$  è il sottoinsieme di variabili a cui imponiamo vincolo di interezza.

Purtroppo un problema MIP è in generale NP-difficile, e la sua risoluzione può richiedere in pratica un grande sforzo computazionale.

### 1.2.1 Branch & Bound

Il *branch-and-bound* è un algoritmo di ricerca ottimo che impiega una strategia *divide-and-conquer*. Si tratta di un algoritmo generico per l'ottimizzazione, ma è qui trattata la sua specializzazione per problemi MIP [3][4].

Viene effettuata una ricerca ad albero dove il nodo radice è il problema originale e ogni nodo è a sua volta un problema MIP, più semplice del suo genitore. Quando un nodo alla frontiera dell'albero è selezionato per l'espansione, l'algoritmo procede così:

1. Risolve il *rilassamento* del problema, dato dalla rimozione di tutti i vincoli di interezza. Si ottiene così un problema LP, risolvibile velocemente con il semplice.

2. Se il rilassamento è impossibile, di certo il problema originale che prevede vincoli aggiuntivi è a sua volta impossibile. Il sottoalbero è potato in quanto *infeasible*.
3. Altrimenti, si considera la soluzione dell'LP. Se questa per caso rispetta i vincoli di interezza, è ammissibile anche per il problema originale e il sottoproblema del nodo attuale è quindi risolto: non è necessario espandere oltre questo nodo. Quando questo accade, l'algoritmo tiene traccia globalmente della migliore soluzione intera trovata finora, detta l'*incumbent*  $\bar{z}$ .
4. Altrimenti, si considera il costo dell'LP risolto; avendo ignorato dei vincoli, questo costo è sicuramente ottimistico rispetto al vero costo della miglior soluzione intera di questo nodo. Se il costo del rilassamento è peggiore dell'*incumbent*  $\bar{z}$ , allora saranno peggiori anche tutti i costi dei nodi del sottoalbero generato da questo nodo. Poiché nessuna soluzione di questo sottoalbero potrà mai migliorare l'*incumbent*, il sottoalbero è potato.
5. Altrimenti, è necessario espandere questo nodo generando nuovi nodi di frontiera. Questo è fatto, in genere, scegliendo una variabile  $x_j$  con vincolo di interezza non rispettato dalla sua soluzione  $x_j^*$  nell'LP. Lo spazio delle soluzioni è partizionato in due parti, associate a due nodi: ad uno si aggiunge il vincolo  $x_j \leq \lfloor x_j^* \rfloor$  mentre all'altro il vincolo  $x_j \geq \lceil x_j^* \rceil$  (figura 1.1). Risolti i nodi figlio, la soluzione al nodo genitore sarà la migliore fra le due dei figli.

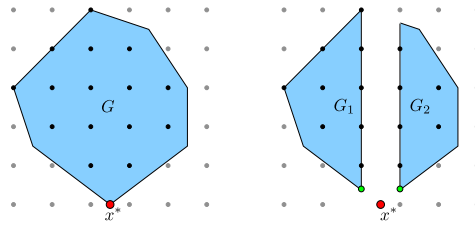
Durante l'esplorazione dell'albero, l'algoritmo migliora progressivamente l'*incumbent*  $\bar{z}$ : in un problema di minimo questo è un upperbound sull'ottimo  $z^*$ . Nel frattempo, il costo di rilassamento più basso fra tutti i nodi aperti costituisce un lowerbound sullo stesso. L'incrocio dei due bound porta alla terminazione dell'algoritmo, mentre in una interruzione forzata la differenza relativa dei due bound è detta *optimality gap*.

Per evitare una ricerca completa dello spazio delle soluzioni, il B&B spera di potare un alto numero di sottoalberi. Le prestazioni sono fortemente influenzate dai due gradi di libertà dell'implementazione:

- L'ordine in cui espandere i nodi di frontiera.
- La scelta della variabile di branching.

### 1.2.2 Branch & Cut

Il *branch-and-cut* è un miglioramento dell'algoritmo B&B, attualmente stato dell'arte per la risoluzione di problemi MIP generici.

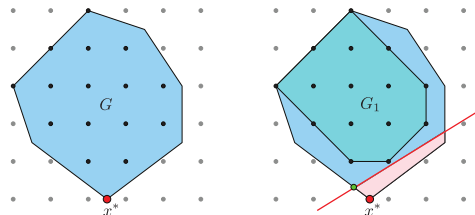


**Figura 1.1:** La procedura di branching partiziona lo spazio delle soluzioni continue  $G$  sulla base della soluzione al rilassamento  $x^*$ , portando a due nuove soluzioni continue da considerare.

Supponiamo che ad un certo nodo il rilassamento trovi la soluzione  $x^*$ , che non rispetta alcuni vincoli di interezza. Esistono delle tecniche in grado di generare dei *piani di taglio*, ossia delle disequazioni violate dalla soluzione  $x^*$  ma che non escludono nessuna soluzione intera; sono cioè valide e ridondanti per il problema MIP originale (figura 1.2). Le disequazioni trovate sono aggiunte al problema LP, che va risolto nuovamente. Questo sforzo è premiato dalla speranza che il nuovo rilassamento:

- Abbia una soluzione con meno variabili frazionarie, o addirittura soluzione intera.
- Abbia una soluzione con un costo migliore.

Partendo dalla nuova soluzione è possibile separare ulteriori piani di taglio. Tuttavia i tagli successivi tendono a essere sempre meno efficaci agli scopi elencati, e incorrono spesso in difficoltà di precisione numerica. È quindi un ulteriore grado di libertà dell'implementazione decidere se e quanti tagli separare, oltre a quali classi di tagli impiegare.



**Figura 1.2:** Un taglio esclude la soluzione al rilassamento  $x^*$ , avvicinando lo spazio delle soluzioni continue  $G$  allo spazio delle soluzioni intere  $G_1$  (*convex hull*) e producendo una nuova soluzione continua.

### 1.3 Problema di assegnamento quadratico

Il *quadratic assignment problem* (QAP) è un problema di *facility location*, cioè si suppone di avere un certo insieme di  $n$  unità (*facilities*) ognuna delle quali deve essere assegnata a una e una sola fra  $n$  posizioni (*locations*). Il problema è descritto da due matrici:

- $A$  è una matrice  $n \times n$ , dove  $a_{uv}$  indica il flusso di informazioni fra le unità  $u$  e  $v$ .

- $B$  è una matrice  $n \times n$ , dove  $b_{ij}$  indica la distanza fra le posizioni  $i$  e  $j$ .

Viene richiesto di trovare l'assegnamento che minimizzi la somma dei prodotti *flusso*  $\times$  *distanza* per tutte le coppie di unità. Matematicamente, si può esprimere il problema come la ricerca di una permutazione  $\pi$  che assegni ogni facility  $i$  alla location  $\pi_i$ ; detto cioè  $\Pi_n$  l'insieme di tutte le permutazioni di  $n$  elementi si cerca:

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

Il problema fu introdotto matematicamente nel 1957 [5] e successivamente provato essere NP-difficile [6]; in pratica risulta molto costoso computazionalmente risolvere grandi istanze di un QAP.

Un gran numero di istanze di problemi di assegnamento quadratico sono raccolte nella libreria *QAPLIB* [7], consultabile online.

### 1.3.1 Istanze TaiC

In un articolo del 1995 [8], E. D. Taillard definisce una classe di problemi di assegnamento quadratico noti come *Tai\*C*. La loro formulazione nasce per la generazione dei pattern di grigio (figura 1.3): nella stampa digitale è utile disporre di griglie di pixel bianchi o neri che quando ripetute su una grande superficie diano l'illusione di una certa tonalità di grigio. La formulazione può essere adeguata anche a problemi di *obnoxious facilities* dove per minimizzare i rischi per la salute i centri inquinanti di una città devono essere disposti lontani fra loro [9].

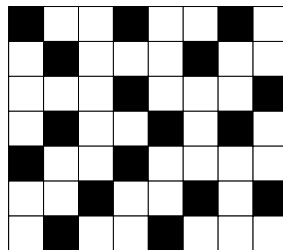
Questa classe di istanze prevede una struttura specifica per la matrice di flusso  $A$ . Le  $n$  facilities sono divise in due classi:  $m$  comunicano in ugual misura fra loro mentre le restanti  $n - m$  sono isolate. Si richiede quindi  $a_{uv} = 1$  se  $u, v \leq m$ ,  $a_{uv} = 0$  altrimenti. Si indicherà con  $d \in [0, 100]$  la densità del pattern richiesto, così che sia  $m = \lfloor \frac{nd}{100} \rfloor$ .

La matrice  $B$  è libera, ma in questo elaborato sono trattate istanze di pattern di grigio, dove tale matrice è generata ipotizzando una forza di repulsione elettrostatica fra i pixel neri, che li costringe a posizionarsi distanti fra loro. Si considera una griglia  $n_1 \times n_2 = n$  e si immagina di ripeterla un totale di nove volte nello spazio; si trova così che la forza subita da un pixel nero in posizione  $(r, s)$  nella griglia centrale a causa dei pixel neri nelle

posizioni  $(t, u)$  di tutte le nove griglie è:

$$f_{rstu} = \max_{v,w \in \{-1,0,1\}} \frac{1}{(r-t+n_1v)^2 + (s-u+n_2w)^2}$$

È indispensabile considerare anche i pixel delle griglie adiacenti per garantire che il pattern creato sia buono anche quando ripetuto nello spazio; se si considerasse la griglia singola, ad esempio, si pagherebbe un costo molto basso per accendere sia il pixel in alto a sinistra che in basso a destra, ma ripetendo il pattern questi risulterebbero invece adiacenti. Per praticità Taillard consiglia di utilizzare  $b_{ij} = \lfloor 10^5 \cdot f_{rstu} \rfloor$  ( $i = r \cdot n_1 + s$ ,  $j = t \cdot n_1 + u$ ).



**Figura 1.3:** Una soluzione ottima per l'istanza 7x8 di densità 30% ( $n = 56$ ,  $m = 17$ ).

# Capitolo 2

## Sviluppo del modello

Questo capitolo fornisce una analisi teorica dei modelli utilizzati per risolvere le istanze QAP interessate e delle tecniche utilizzate per migliorarne le prestazioni.

### 2.1 Formulazione del modello

#### 2.1.1 Modello quadratico generale

Descriviamo il generico problema di assegnamento quadratico tramite una formulazione algebrica. Siano:

- $U$  l'insieme delle unità (o facilities).
- $I$  l'insieme delle posizioni (o locations).
- $A = (a_{uv})$  la matrice dei flussi.
- $B = (b_{ij})$  la matrice delle distanze.

Utilizziamo una variabile per ogni coppia unità-posizione, con semantica:

$$x_{iu} = \begin{cases} 1 & \text{se l'unità } u \in U \text{ è stata assegnata alla posizione } i \in I \\ 0 & \text{altrimenti} \end{cases}$$

Ogni unità deve essere assegnata esattamente una volta, e ogni posizione deve ospitare

esattamente una unità; allora il problema può essere scritto come:

$$\begin{aligned}
 & \min \sum_{i \in I} \sum_{u \in U} \sum_{j \in I} \sum_{v \in U} a_{uv} \cdot b_{ij} \cdot x_{iu} \cdot x_{jv} \\
 & \sum_{i \in I} x_{iu} = 1 \quad \forall u \in U \\
 & \sum_{u \in U} x_{iu} = 1 \quad \forall i \in I \\
 & x_{iu} \in \{0, 1\} \quad \forall i \in I, u \in U
 \end{aligned} \tag{2.1}$$

### 2.1.2 Modello quadratico TaiC

Per quanto valida, la formulazione generica 2.1 può essere notevolmente semplificata nelle istanze di tipo TaiC. La struttura della matrice  $A$  porta a una fortissima simmetria: le unità vengono divise in due insiemi, e al loro interno le varie unità sono equivalenti fra loro. È noto come modellare queste simmetrie sia essenziale alla risoluzione dei problemi QAP [10]. Diciamo essere di categoria 1 le unità che comunicano fra loro e di categoria 0 le unità isolate, corrispondenti rispettivamente ai pixel neri e bianchi nei problemi di pattern di grigio. È allora sufficiente una sola variabile per ogni posizione, con significato:

$$x_i = \begin{cases} 1 & \text{se un'unità di categoria 1 è stata assegnata alla posizione } i \in I \\ 0 & \text{altrimenti} \end{cases}$$

Si ottiene in questo modo un modello con solamente  $n$  variabili e un vincolo:

$$\begin{aligned}
 & \min \sum_{i \in I} \sum_{j \in I} b_{ij} \cdot x_i \cdot x_j \\
 & \sum_{i \in I} x_i = m \\
 & x_i \in \{0, 1\} \quad \forall i \in I
 \end{aligned} \tag{2.2}$$

### 2.1.3 Linearizzazione intuitiva

Il modello 2.2 presenta un termine non lineare nella funzione obiettivo, cioè il prodotto  $x_i \cdot x_j$ . Per rendere il problema trattabile da un risolutore MIP è necessario rendere lineare il modello. Una tecnica nota per la linearizzazione di un prodotto di due variabili binarie è l'introduzione di un nuovo insieme di variabili binarie  $y$  con significato:

$$y_{ij} = x_i \cdot x_j \quad \forall i, j \in I \tag{2.3}$$

La coerenza fra le variabili  $x$  e  $y$  viene imposta dai vincoli lineari:

$$\begin{aligned} y_{ij} &\leq x_i && \forall i, j \in I \\ y_{ij} &\leq x_j && \forall i, j \in I \\ y_{ij} &\geq x_i + x_j - 1 && \forall i, j \in I \end{aligned} \quad (2.4)$$

Sostituendo 2.3 nella funzione obiettivo e aggiungendo i vincoli 2.4 al modello quadratico 2.2 otteniamo un modello lineare per i problemi TaiC:

$$\begin{aligned} \min & \sum_{i \in I} \sum_{j \in I} b_{ij} \cdot y_{ij} \\ & \sum_{i \in I} x_i = m \\ y_{ij} &\leq x_i && \forall i, j \in I \\ y_{ij} &\leq x_j && \forall i, j \in I \\ y_{ij} &\geq x_i + x_j - 1 && \forall i, j \in I \\ x_i, y_{ij} &\in \{0, 1\} && \forall i, j \in I \end{aligned} \quad (2.5)$$

Questo modello soffre di un debole rilassamento lineare: se una soluzione continua presenta  $x_i = \frac{m}{n} \leq \frac{1}{2} \forall i \in I$  è consentito porre  $y_{ij} = 0 \forall i, j \in I$ . Quindi, si soddisfa il vincolo  $\sum_{i \in I} x_i = m$ , senza pagare però nessun costo nella funzione obiettivo.

#### 2.1.4 Linearizzazione KB

Una linearizzazione alternativa del modello generale 2.1 è dovuta a Kaufman e Broeckx [11]. È presentata di seguito la sua semplificazione per i problemi TaiC come alternativa al modello 2.5. Partendo dal modello TaiC quadratico 2.2 si raccoglie a fattore comune la funzione obiettivo:

$$\sum_{i \in I} \sum_{j \in I} b_{ij} \cdot x_i \cdot x_j = \sum_{i \in I} \left( x_i \sum_{j \in I} b_{ij} \cdot x_j \right)$$

Si introduce un nuovo insieme di variabili continue positive  $w$ , dove  $w_i$  indica il costo totale che l'unità posizionata in  $i \in I$  paga a causa delle altre unità di categoria 1:

$$w_i := x_i \cdot \sum_{j \in I} b_{ij} \cdot x_j = \begin{cases} \sum_{j \in I} b_{ij} \cdot x_j & \text{se } x_i = 1 \\ 0 & \text{se } x_i = 0 \end{cases} \quad \forall i \in I \quad (2.6)$$

È possibile rendere lineare questa relazione quando  $w_i$  viene minimizzato imponendo:

$$w_i \geq \sum_{j \in I} (b_{ij} \cdot x_j) - M_i \cdot (1 - x_i) \quad \forall i \in I \quad (2.7)$$

dove  $M_i$  è una costante grande a sufficienza affinché il termine  $M_i \cdot (1 - x_i)$  possa annullare completamente il vincolo quando  $x_i = 0$ . In questo modo, se  $x_i = 0$  allora il vincolo 2.7 si riduce alla banale  $w_i \geq 0$ , mentre in caso contrario chiede  $w_i \geq \sum_{j \in I} b_{ij} \cdot x_j$ ; viene cioè rispecchiata la definizione 2.6. Quindi,  $M_i$  deve essere maggiore di qualsiasi valore possa raggiungere la sommatoria che deve poter annullare, e può essere trovato risolvendo:

$$\begin{aligned} M_i = \max \quad & \sum_{j \in I} b_{ij} \cdot x_j \\ & \sum_{j \in I} x_j = m \\ & x_i = 0 \\ & x_j \in \{0, 1\} \quad \forall j \in I \end{aligned} \quad (2.8)$$

Si scrive allora il modello di Kaufman e Broeckx (*KB* in breve) per i problemi TaiC:

$$\begin{aligned} \min \quad & \sum_{i \in I} w_i \\ & \sum_{i \in I} x_i = m \\ & w_i \geq \sum_{j \in I} (b_{ij} \cdot x_j) - M_i \cdot (1 - x_i) \quad \forall i \in I \\ & x_i \in \{0, 1\} \quad \forall i \in I \\ & w_i \geq 0 \quad \forall i \in I \end{aligned} \quad (2.9)$$

Rispetto al modello intuitivo 2.5 quest'ultimo richiede solamente  $n$  variabili addizionali invece che  $n^2$ . Anche il numero di vincoli è ridotto, per quanto il numero di coefficienti non-nulli rimanga  $\Theta(n^2)$ . Anche questo modello soffre di un rilassamento molto debole a causa del vincolo 2.7: dato che  $M_i$  deve essere sufficientemente grande da spegnere il vincolo completamente per  $x_i = 0$ , anche valori piccoli ma non nulli di  $x_i$  permettono di porre  $w_i = 0$  e non pagare nessun costo.

### 2.1.5 Miglioramento del lowerbound

Il modello 2.9 ha un rilassamento così debole che al nodo radice ha costo nullo, e si rivela poco utile alla risoluzione di problemi QAP. È noto un miglioramento a questo modello che consente di far pagare un costo anche per i valori di  $x_i$  così piccoli da non attivare i vincoli 2.7 [12]. Dato che ogni soluzione accettabile deve aver posto esattamente  $m$  unità nella categoria 1, l'accensione seppur parziale di una variabile  $x_i$  dovrà almeno pagare il

costo legato alle  $m$  unità di categoria 1 della soluzione, qualsiasi esse siano. Si calcola il costo minimo da pagare alla posizione  $i \in I$  quando vi è posta un'unità di categoria 1 in modo simile alla 2.8:

$$\begin{aligned} L_i = \min \quad & \sum_{j \in I} b_{ij} \cdot x_j \\ & \sum_{j \in I} x_j = m \\ & x_i = 1 \\ & x_j \in \{0, 1\} \quad \forall j \in I \end{aligned} \quad (2.10)$$

Calcolato  $L_i$ , si aggiungono al modello i tagli:

$$w_i \geq L_i \cdot x_i \quad \forall i \in I \quad (2.11)$$

Questi tagli non soffrono degli svantaggi dei vincoli 2.7, dato che non vengono disattivati per piccoli valori di  $x_i$ . La loro aggiunta al modello non modifica lo spazio delle soluzioni intere, ma migliora notevolmente il rilassamento lineare, rendendo il modello utilizzabile. Permettono inoltre di raggiungere, al nodo radice del B&B, il lowerbound di Gilmore-Lawler per il QAP [13].

## 2.2 Separazione dei tagli

È possibile migliorare notevolmente le prestazioni del modello separando ulteriori tagli ad ogni nodo dell'albero branch & bound, in stile branch & cut. Il risolutore è già in grado di separare diverse classi di tagli generiche a tutti i problemi MIP, a partire dal modello fornito. Tuttavia, il modello di per sé non contiene tutte le informazioni sul problema: il progettista che ne conosce la semantica può quindi fornire al risolutore classi di tagli (come i 2.11) altrimenti non derivabili solo dalla formulazione algebrica.

### 2.2.1 Costanti locali

Le classi di tagli più efficaci sono legate al miglioramento locale delle costanti  $L_i$  e  $M_i$  del modello. Queste costanti rappresentano il costo minimo e massimo associato all'attivazione della posizione  $i \in I$ , e il loro valore può essere rafforzato se ad un nodo il branching ha fissato variabili. Siano  $P_0, P_1 \subseteq I$  le variabili  $x$  fissate a 0 e 1 rispettivamente; allora è

possibile rafforzare le costanti locali secondo:

$$\begin{aligned}
 L_i = \quad & \min \sum_{j \in I} b_{ij} \cdot x_j \\
 & \sum_{j \in I} x_j = m \\
 & x_j = 0 \quad \forall j \in P_0 \\
 & x_j = 1 \quad \forall j \in P_1 \cup \{i\} \\
 & x_j \in \{0, 1\} \quad \forall j \in I
 \end{aligned} \tag{2.12}$$

$$\begin{aligned}
 M_i = \quad & \max \sum_{j \in I} b_{ij} \cdot x_j \\
 & \sum_{j \in I} x_j = m \\
 & x_j = 0 \quad \forall j \in P_0 \cup \{i\} \\
 & x_j = 1 \quad \forall j \in P_1 \\
 & x_j \in \{0, 1\} \quad \forall j \in I
 \end{aligned} \tag{2.13}$$

Risolvere i due problemi di minimo 2.12 e massimo 2.13 ad ogni nodo è relativamente economico: preordinando le righe di  $B$ , si selezionano le unità meno/più costose della riga tenendo conto delle variabili fisse a 0 e 1. Complessivamente è richiesto tempo  $\Theta(n)$ . I vincoli 2.11 e 2.7 sono aggiunti come tagli al modello<sup>1</sup>, con unica modifica il valore rafforzato delle costanti.

### 2.2.2 Tagli fra coppie libera-fissa

I tagli 2.11 tengono in considerazione una sola posizione ed il solo costo ad essa associato. Allo stesso modo il valore di  $L_i$  tiene conto solamente di una singola unità: nel risolvere 2.12 verranno sempre scelte per contribuire alla somma le posizioni più distanti all'unità  $i$ , anche quando queste risultassero molto vicine ad altre unità di categoria 1. Si cerca di punire queste scelte con una classe di tagli che generalizzino i 2.11 tendono conte delle variabili già fissate a 1. Nella loro forma più generale, si sceglie un qualsiasi  $P^* \subseteq P_1$  e si separa  $\forall k \notin (P_0 \cup P_1)$  il taglio:

$$w_k + \sum_{j \in P^*} w_j \geq F_{0,k} + (F_{1,k} - F_{0,k}) \cdot x_k$$

<sup>1</sup>Per ragioni di efficienza, è opportuno assicurarsi prima di aggiungere qualsiasi taglio al modello che questo sia violato dalla soluzione del rilassamento al nodo locale.

$$\begin{array}{ll}
F_{0,k} = \min \sum_{i \in I} \sum_{j \in P^*} b_{ij} y_i & F_{1,k} = \min \sum_{i \in I} \sum_{j \in P^* \cup \{k\}} b_{ij} y_i \\
\sum_{i \in I} y_i = m & \sum_{i \in I} y_i = m \\
y_i = 1 \quad \forall i \in P_1 & y_i = 1 \quad \forall i \in P_1 \cup \{k\} \\
y_i = 0 \quad \forall i \in P_0 \cup \{k\} & y_i = 0 \quad \forall i \in P_0 \\
y_i \in \{0, 1\} \quad \forall i \in I & y_i \in \{0, 1\} \quad \forall i \in I
\end{array}$$

Semanticamente,  $F_{0,k}$  è il costo minimo da pagare per le posizioni in  $P^*$  indipendentemente dalla scelta di  $x_k$ .  $F_{1,k}$  è invece il costo minimo da pagare per le posizioni in  $P^* \cup \{k\}$  se si sceglie di porre  $x_k = 1$ . Nel calcolo delle costanti la scelta delle  $y_i$  deve minimizzare *contemporaneamente* il costo per tutte le unità coinvolte, portando a un valore più alto rispetto alla somma delle singole  $L_j$ . Questa classe di tagli spesso non è violata dato che  $F_{0,k}$  è un lowerbound poco stretto per  $\sum_{j \in P^*} w_j$ , che invece compare per intero al lato sinistro della disequazione.

Essendo separabili  $O(n)$  tagli per ognuno degli  $\Theta(2^{|P^*|})$   $P^* \subseteq P_1$  sono state implementate due sottoclassi ritenute interessanti, utilizzando per  $P^*$ :

- Gli  $O(n)$  insiemi con  $|P^*| = 1$ ; in totale  $O(n^2)$  possibili tagli.
- $P^* = P_1$ ; in totale  $O(n)$  possibili tagli.

### 2.2.3 Tagli fra coppie libera-libera

Un'ultima classe di tagli mette in relazione i costi di coppie di variabili libere al nodo considerato. Si vuole nuovamente aggirare il difetto dei lowerbound  $L_i$ , calcolati su posizioni favorevoli alla singola unità ma costosi per l'intera soluzione. Si separa  $\forall j, k \in I \setminus (P_0 \cup P_1)$ ,  $j \neq k$ :

$$w_j + w_k \geq L_j \cdot x_j + L_k \cdot x_k + \Delta F_{jk} \cdot (x_j + x_k - 1)$$

$$\begin{array}{l}
\Delta F_{jk} = F_{1,jk} - L_j - L_k \geq 0 \\
F_{1,jk} = \min \sum_{i \in I} (b_{ij} + b_{ik}) \cdot y_i \\
\sum_{j \in I} y_i = m \\
y_i = 0 \quad \forall i \in P_0 \\
y_i = 1 \quad \forall i \in P_1 \cup \{j, k\} \\
y_i \in \{0, 1\} \quad \forall i \in I
\end{array}$$

$F_{1,jk}$  rappresenta il costo minimo da pagare per porre unità di categoria 1 sia nella posizione  $j$  che nella posizione  $k$ .  $\Delta F_{jk}$  rappresenta il costo aggiuntivo dovuto solo se  $j$  e  $k$  sono attivate insieme, e non individualmente. La classe di tagli può essere generalizzata per considerare più di due unità alla volta, ma questo non ha portato risultati utili; in effetti già con due unità quando il termine  $x_j + x_k - 1$  scende sotto lo zero il taglio diviene ridondante rispetto alla somma dei due rispettivi tagli 2.11: con ulteriori unità un ipotetico termine  $\sum_{i \in P^*} x_i - |P^*| + 1$  sarebbe di rado positivo.

Per evitare calcoli dispendiosi delle  $O(n^2)$  costanti  $F_{1,jk}$  e annesse disequazioni, vengono considerate solo le coppie  $j, k$  sopra la soglia empirica  $x_j + x_k - 1 \geq 0.4$ .

### 2.2.4 Margine di violazione

Quando vengono separati tagli a un nodo, è necessario ricalcolare la soluzione del rilassamento. Essendo questa un'operazione costosa, è opportuno escludere i tagli violati con un margine ridotto poiché si riveleranno certamente poco utili.

Da un'analisi statistica dei margini di violazione condotta su alcune istanze difficili, si osserva che un buon discriminante valido indipendentemente dalla dimensione dell'istanza è il rapporto

$$\frac{\Delta}{\sqrt{obj}} \tag{2.14}$$

dove  $\Delta$  è il margine di violazione della disequazione associata al taglio, e  $obj$  è il valore dalla funzione obiettivo del rilassamento considerato. È individuata per ogni classe di tagli una soglia  $T$  che scarti all'incirca metà dei tagli: i candidati con rapporto 2.14 sotto tale soglia non sono aggiunti al modello.

## 2.3 Priorità di branching

Un grado di libertà dell'algoritmo branch & bound è dato dalla scelta della variabile su cui effettuare branching quando la soluzione del rilassamento si rivela frazionaria. Un branching efficace può avere un forte impatto sulle prestazioni dell'algoritmo [14, §5].

Certamente l'algoritmo di branching deve preferire le variabili che nella soluzione risultano più lontane da valori interi. Per le variabili binarie  $x_i$  questa distanza è misurabile

secondo un valore di *infeasibility*:

$$u_i = \frac{1}{2} - \left| x_i - \frac{1}{2} \right| \quad \forall i \in I$$

Utilizzare esclusivamente  $u_i$  come priorità di branching in pratica non è migliore di una selezione casuale, ma è il punto di partenza per molte strategie note in letteratura. Si propone quindi una strategia che pesi tali valori di distanza per un certo punteggio  $s_i$ ; verrà scelta per il branching la variabile con il più alto prodotto  $u_i \cdot s_i$ . Sono riportate di seguito due di queste strategie.

### 2.3.1 Scarto tra i bound

Un peso delle variabili rivelatosi efficace per altre classi di problemi QAP [10] è:

$$s_i = M_i - L_i \geq 0 \quad \forall i \in I$$

Ad un certo nodo  $M_i$  rappresenta il costo  $w_i$  massimo che ci si potrà ottenere alla posizione  $i$ , mentre  $L_i$  il costo minimo. L'idea è che quando la loro differenza è ampia, questo indica che si ha poca informazione sulla variabile  $x_i$ . Inoltre, il rilassamento lineare è più forte per valori bassi di  $M_i$  e per valori alti di  $L_i$ . Il branching su  $x_i$  fisserà la variabile o a 0 o a 1, risolvendo entrambe le criticità nei futuri nodi del sottoalbero.

### 2.3.2 Scarto con il costo quadratico

In una soluzione frazionaria di un QAP, il vero costo associato ad ogni posizione è:

$$\bar{w}_i = \sum_{j \in I} b_{ij} \cdot x_i \cdot x_j \quad \forall i \in I$$

Nel modello KB (2.9) questo costo è modellato attraverso le variabili  $w_i$ . Tuttavia, a causa della debolezza del rilassamento dei vincoli 2.7, spesso si osserva una forte discrepanza fra il costo lineare  $w_i$  e il costo quadratico  $\bar{w}_i$ . Si utilizza allora come peso per il branching tale differenza:

$$s_i = \bar{w}_i - w_i \geq 0 \quad \forall i \in I$$

In questo modo, si tenta di eliminare dal sottoalbero le variabili a cui è associato un costo più basso di quello reale. Infatti, dopo il branching la variabile selezionata assumerà valore intero e sotto tale ipotesi è sempre  $w_i = \bar{w}_i$ .

### 2.3.3 Ordine di esplorazione

Un ulteriore grado di libertà dell'implementazione branch & bound è l'ordine in cui esplorare i nodi di frontiera generati dal branching. Generalmente vengono selezionati per primi i nodi con costo del rilassamento più basso: dato che il più basso rilassamento fra i nodi aperti è lowerbound globale per la soluzione ottima, questa strategia porta ad un progressivo miglioramento del gap di ottimalità durante la risoluzione. Si produce inoltre una ricerca più uniforme che trova in fretta buoni incumbent.

I problemi TaiC del QAP richiedono però molta enumerazione di nodi, dato che a basse profondità il costo del rilassamento è in genere molto più basso dell'incumbent. Questo rende plausibile anche un'esplorazione in profondità (DFS, *depth first search*) dell'albero delle soluzioni. Si tratta di una strategia che per un MIP generale è spesso miope e rischia di bloccarsi in sottoalberi subottimi, ma porta due vantaggi:

- Un ridotto uso della memoria, dato che la dimensione della coda di nodi aperti è limitata dall'altezza dell'albero e non più dal numero totale di nodi.
- Durante la risoluzione si esplorano in sequenza sottoproblemi molto simili, portando a migliori *warm-start* del simplesso.

Un notevole lato negativo dell'esplorazione DFS è che durante la risoluzione non porta ad un graduale miglioramento del lowerbound globale. Se l'algoritmo di ricerca è interrotto prima che termini (ad esempio perché si è raggiunto un limite di tempo) ci si può attendere un gap di ottimalità ancora molto alto.

# Capitolo 3

## Risultati sperimentali

Questo capitolo ripercorre le tecniche teoriche già descritte e ne riporta i risultati sperimentali. Vengono offerti confronti fra diversi modelli, varianti ed implementazioni.

### 3.1 Metodologia

#### 3.1.1 Istanze

Le istanze utilizzate durante la sperimentazione sono in totale 70. Vengono ottenute come descritto alla sezione 1.3.1 combinando:

- Dimensioni crescenti:  $4 \times 4$ ,  $5 \times 5$ ,  $4 \times 8$ ,  $6 \times 6$ ,  $5 \times 8$ ,  $6 \times 7$ ,  $5 \times 9$ ,  $7 \times 7$ ,  $7 \times 8$ ,  $8 \times 8$ .
- Densità crescenti: 20%, 30%, 40%, 50%, 60%, 70%, 80%.

Tuttavia, 7 di queste combinazioni non sono state risolte da nessuna implementazione e sono per questo escluse dai confronti statistici.

Per ottenere risultati consistenti, tutte le istanze sono state ripetute 3 volte utilizzando semi differenti per i generatori pseudocasuali [15]. I valori numerici riportati per una certa istanza sono dati dalla media aritmetica delle esecuzioni con semi diversi. Tutte le esecuzioni sono soggette a un limite di tempo di 4 ore, e a un limite di memoria di 14GB.

Si osserva che per una certa istanza TaiC con parametri  $(n, m)$ , il numero di soluzioni ammissibili è dato dal coefficiente binomiale  $B(n, m)$ . Come lo spazio delle soluzioni, anche il tempo impiegato a risolvere l'istanza e il numero di nodi esplorati crescono esponenzialmente e spaziano diversi ordini di grandezza. Per questa ragione le medie che coinvolgono tali quantità su più istanze adottano una media geometrica traslata, di 1

secondo per i tempi e di 1000 per i nodi. Per confrontare fra loro più varianti del modello, sono riportati sia il tempo medio sulle istanze risolte da tutte le versioni considerate che il tempo medio su tutte le 63 istanze; in quest'ultimo caso un'istanza non risolta è contata al time limit di 4 ore, anche in caso l'esecuzione termini per esaurimento della memoria.

### 3.1.2 Software

L'algoritmo risolutivo è stato implementato in C++, e si appoggia per la risoluzione dei problemi LP e MIP al risolutore commerciale CPLEX di IBM [16]. L'aggregazione statistica dei dati è stata implementata in Python, e si appoggia a librerie scientifiche (*pandas*, *matplotlib*, ...). Il codice sorgente sviluppato è reso disponibile online [17].

In particolare, il risolutore CPLEX è stato così impostato:

- Massima simmetria: `CPXPARAM_Preprocessing_Symmetry = 5`.
- Enfasi sulla memoria: `CPXPARAM_Emphasis_Memory = 1`.
- Nodi compressi e salvati su disco: `CPXPARAM_MIP_Strategy_File = 3`.
- Tagli disabilitati: `CPXPARAM_MIP_Limits_CutsFactor = 0`. Si è osservato infatti che le classi di tagli separate automaticamente da CPLEX non sono mai efficaci, e il loro calcolo rallenta la ricerca.

L'interazione fra il risolutore CPLEX e le routine C++ implementate avviene tramite callback sia per la separazione dei tagli (`CPX_CALLBACKCONTEXT_RELAXATION`) che per la strategia di branching (`CPX_CALLBACKCONTEXT_BRANCHING`).

### 3.1.3 Ambiente

L'hardware utilizzato è stato messo a disposizione dal cluster del Dipartimento di Ingegneria dell'Informazione. I tempi di esecuzione sono stati ottenuti su macchine con le seguenti specifiche:

- Intel(R) Xeon(R) CPU E5-2623 v3 @ 3.00GHz quad-core
- 16GB RAM
- Linux Fedora 37
- Gnu C Compiler 12.3
- IBM ILOG CPLEX Optimization Studio versione 22.1

## 3.2 Confronti statistici

### 3.2.1 Il modello KB base

Il modello KB (2.9) è il principale interessato da questa analisi. Si riportano i risultati che ne portano progressivi miglioramenti fino a raggiungere quello che è la linea base per i confronti successivi:

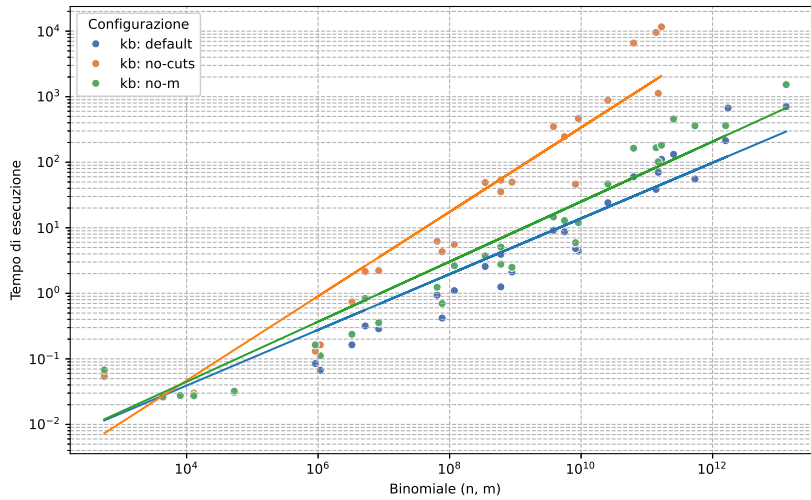
- **no-cuts**: configurazione che non prevede nessun taglio generato ai nodi. I tagli 2.11 sono aggiunti solo al modello iniziale e mai rafforzati.
- **no-m**: configurazione che prevede la separazione di tagli 2.11 ai nodi che migliorino le costanti locali  $L_i$  (sezione 2.2.1). I vincoli 2.7 non sono mai rafforzati.
- **default**: configurazione di base per il modello KB. Prevede la separazione di tagli ai nodi che migliorino sia le costanti locali  $L_i$  che  $M_i$  (sezione 2.2.1).

Sono riportati nella tabella 3.1 e nella figura 3.1 i risultati ottenuti da queste varianti. Si osserva dai dati ottenuti che il rafforzamento delle costanti locali tramite tagli porta a un notevole miglioramento delle prestazioni di risoluzione. Per questo motivo, le varianti trattate in seguito saranno da intendersi come addizioni alla configurazione di base **default**, e includeranno tutte la separazione dei tagli volti a rafforzare le costanti  $L_i, M_i$  ai nodi.

Si nota inoltre che la principale limitazione di queste implementazioni non è il limite di tempo (4 ore) bensì la memoria (14GB). Fatta eccezione della prima configurazione che non separa tagli, negli altri casi questa tecnica porta ad un veloce esaurimento della memoria disponibile. La causa è legata al risolutore CPLEX, che mantiene in memoria una *cut pool* dove salvare i tagli separati ai un nodi nel caso in cui tornassero utili in un secondo momento. Non si è trovata nessuna impostazione del software che prevenisse questo consumo di memoria.

Variante	Tempo <small>(63 i.)</small>	Tempo <small>(44 i.)</small>	Nodi <small>(44 i.)</small>	Istanze risolte
no-cuts	215.71	34.39	303380	44
no-m	70.20	8.68	55646	48
<b>default</b>	<b>30.30</b>	<b>3.78</b>	<b>14339</b>	<b>53</b>

**Tabella 3.1:** Valori medi (media geometrica traslata) e istanze risolte dal modello base e le sue varianti più semplici. Tempi (secondi) sia per le 44 istanze risolte da tutte le 3 configurazioni che per tutte le 63 istanze.



**Figura 3.1:** Tempi di esecuzione in funzione di  $B(n, m)$ . Scala logaritmica. Solo densità  $\leq 50\%$ .

### 3.2.2 Varianti al modello KB

È stato misurato l'impatto che ognuna delle tecniche descritte nelle sezioni 2.2 e 2.3 ha avuto sul modello KB di base. In particolare sono state studiate:

- **p:** configurazione che prevede la separazione di tagli che coinvolgono una variabile libera e una sola variabile fissa a 1 (sezione 2.2.2, caso  $|P^*| = 1$ ). Questa classe di tagli si è rivelata poco performante, soprattutto su istanze grandi.
- **a:** configurazione che prevede la separazione di tagli che coinvolgono una variabile libera e tutte le variabili fisse a 1 (sezione 2.2.2, caso  $P^* = P_1$ ). Questa classe di tagli si è rivelata poco performante, soprattutto su istanze grandi.
- **f:** configurazione che prevede la separazione di tagli che coinvolgono due variabili libere (sezione 2.2.3). Questa classe di tagli ottiene risultati molto variabili a seconda dell'istanza, con tendenza a migliorare le prestazioni per problemi grandi e per problemi a bassa densità.
- **r1:** configurazione che prevede una separazione meno aggressiva dei tagli, escludendone circa metà (sezione 2.2.4). Sarà necessario esplorare più nodi, ma il numero ridotto di tagli consente di risparmiare tempo e memoria. Si utilizzano le seguenti soglie:  $T_l = 1.0$  (tagli 1, 2.11),  $T_m = 2.1$  (tagli m, 2.7).
- **r3:** configurazione che prevede una separazione molto meno aggressiva dei tagli, escludendone circa il 90% (sezione 2.2.4). Le soglie  $T$  sono triple rispetto a **r\_1**. Il risparmio di memoria consente di risolvere 5 istanze in più rispetto a **default**.

- **b1**: configurazione che prevede strategia di branching pesata su  $M_i - L_i$  (sezione 2.3.1). I risultati variano a seconda dell'istanza, sono migliori per problemi di taglia piccola e su istanze poco dense.
- **b2**: configurazione che prevede strategia di branching pesata su  $\bar{w}_i - w_i$  (sezione 2.3.2). Porta a miglioramenti significativi rispetto alla strategia di branching default di CPLEX.
- **dfs**: configurazione che prevede un'esplorazione in profondità dell'albero di ricerca (sezione 2.3.3). Inizialmente inclusa per il forte risparmio di memoria, si è rivelata mediamente più veloce, specialmente su istanze grandi.
- **f+r1+b2**: configurazione che unisce le varianti più performanti. I tagli **f** (sezione 2.2.3) sono inclusi sopra la soglia  $T_f = 4.9$ .

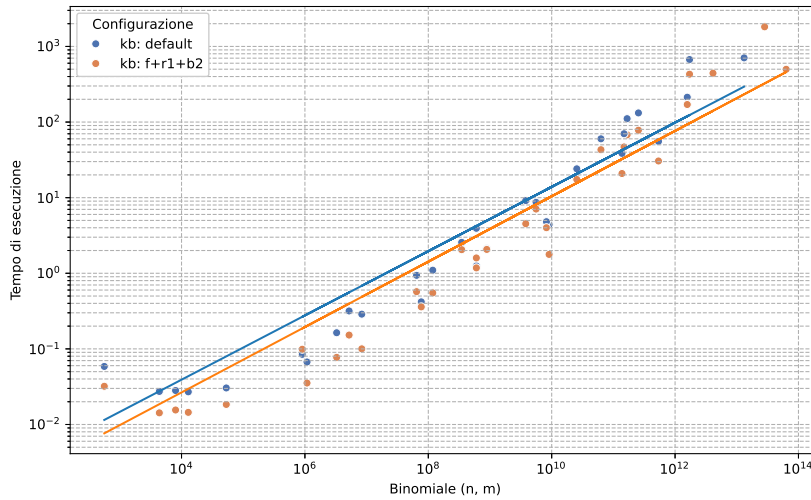
Sono riportati nella tabella 3.2 e nella figura 3.2 i risultati ottenuti da queste varianti. Si nota come non tutte le tecniche trattate abbiano avuto un impatto positivo sulle prestazioni del modello. Se una classe di tagli è poco efficace, il tempo speso a separare il taglio e ricalcolare la soluzione del rilassamento non compensa i benefici ottenuti.

Variante	Tempo <small>(63 i.)</small>	Tempo <small>(52 i.)</small>	Nodi <small>(52 i.)</small>	Istanze risolte
default	30.30	8.07	29780	53
p	33.82	9.29	29423	53
a	36.19	9.94	29898	53
f	30.43	8.16	<b>25690</b>	53
r1	23.77	7.22	32193	57
r3	23.52	7.45	40625	<b>58</b>
b1	27.98	7.73	30844	54
b2	26.86	7.38	29616	54
dfs	24.11	6.80	29324	55
<b>f+r1+b2</b>	<b>21.03</b>	<b>6.27</b>	27969	57

**Tabella 3.2:** Valori medi (media geometrica traslata) e istanze risolte dal modello base e le sue varianti successive. Tempi (secondi) sia per le 52 istanze risolte da tutte le 10 configurazioni che per tutte le 63 istanze.

### 3.2.3 Implementazione ad-hoc

Dal momento che le limitazioni di memoria hanno impedito la risoluzione delle istanze più grandi, si è abbandonato il risolutore MIP di CPLEX in favore di una implementazione *ad-hoc* che sfruttasse il software di IBM solamente per risolvere i rilassamenti LP. Le principali funzionalità implementate sono:



**Figura 3.2:** Tempi di esecuzione in funzione di  $B(n, m)$ . Confronto modello base e migliore variante. Scala logaritmica. Solo densità  $\leq 50\%$ .

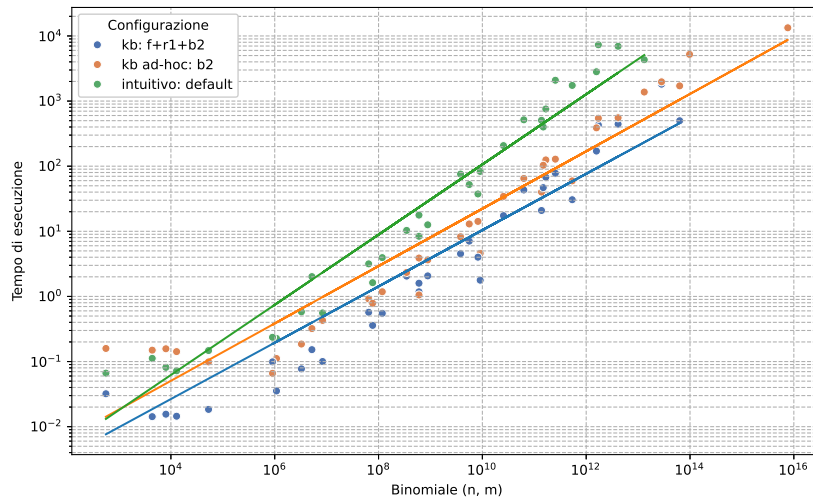
- Sfruttare la simmetria del problema per porre  $x_1 = 1$  senza perdita di generalità.
- Miglioramento delle costanti locali modificando i coefficienti dell'LP (sezione 2.2.1).
- Branching pesato su  $M_i - L_i$  (variante **b1**, sezione 2.3.1) oppure su  $\bar{w}_i - w_i$  (variante **b2**, sezione 2.3.2).
- Multithreading.
- Selezione dei nodi in ordine di bound, con *diving*. Ovvero, dei due nodi prodotti dal branching uno è messo in coda e l'altro è espanso subito, e ripeterà il diving ricorsivamente. Questa parziale ricerca in profondità (sezione 2.3.3) migliora i *warm-start* del simplesso e permette di trovare buone soluzioni già all'inizio della ricerca.

Si confronta inoltre il modello KB (2.9) con il modello intuitivo (2.5). I risultati del modello 2.5 sono già stati trattati in un precedente elaborato scritto da Toffolon M. [18]. Tuttavia, per un confronto coerente il modello è stato re-implementato in C++ in modo simile al modello KB.

Sono riportati nella tabella 3.3 e nella figura 3.3 i risultati ottenuti da queste implementazioni. Si osserva come la migliore variante del modello KB sia molto più performante del modello intuitivo, anche di un'ordine di grandezza. L'implementazione ad-hoc, per quanto più lenta, è efficiente rispetto alla memoria e riesce così a risolvere un alto numero di istanze.

Implementazione	Variante	Tempo <sub>(63 i.)</sub>	Tempo <sub>(54 i.)</sub>	Nodi <sub>(54 i.)</sub>	Istanze risolte
intuitivo	default	88.54	38.26	80537	55
kb	f+r1+b2	<b>21.03</b>	<b>7.58</b>	<b>33696</b>	57
kb ad-hoc	b1	24.39	9.80	40529	62
kb ad-hoc	b2	21.79	8.82	35887	<b>63</b>

**Tabella 3.3:** Valori medi (media geometrica traslata) e istanze risolte dalle diverse implementazioni proposte. Tempi (secondi) sia per le 54 istanze risolte da tutte le 4 configurazioni che per tutte le 63 istanze.



**Figura 3.3:** Tempi di esecuzione in funzione di  $B(n, m)$ . Scala logaritmica. Solo densità  $\leq 50\%$ .

### 3.2.4 Dati complessivi

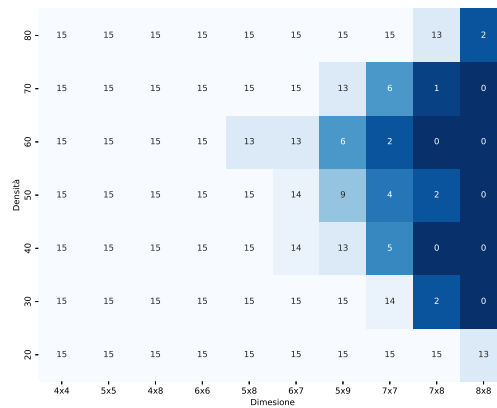
Si osserva in generale come la complessità del problema cresca in modo estremamente veloce:

- Le istanze 6x6 sono facilmente risolte da tutte le configurazioni.
- Le istanze 7x7 sono tutte risolte solamente dall'algoritmo *ad-hoc*.
- Le istanze 8x8 sono risolubili solo alle densità più semplici.

Come già osservato in precedenza [18] si nota come le densità intermedie siano le più difficili da trattare. Questo non dovrebbe sorprendere dato che hanno il più ampio spazio delle soluzioni ammissibili. Fa eccezione la densità 50%, che nonostante massimizzi la binomiale  $B(n, m)$  fissato  $n$ , presenta tempi di risoluzione più bassi delle densità adiacenti; questo è probabilmente dovuto alle maggiori simmetrie e regolarità che si presentano in questi casi.

Si osserva che con le tecniche proposte da questo elaborato le densità alte ( $\geq 60\%$ ) risultano più complesse rispetto alle loro corrispettive più basse, nonostante un simile spazio delle soluzioni. Tuttavia, è possibile ottenere una soluzione ottima per il problema  $(n, m)$  nota la soluzione al suo speculare  $(n, n - m)$  semplicemente invertendo la soluzione ( $x'_i = 1 - x_i$ ). Nei casi pratici questo sarebbe il modo migliore per risolvere problemi TaiC ad alta densità con tecniche MIP.

Si ha riscontro di queste osservazioni nelle figure 3.4 e 3.5.



**Figura 3.4:** Il numero di configurazioni che hanno risolto una certa istanza. I problemi più difficili sono stati risolti solo dall'implementazione *ad-hoc*.



**Figura 3.5:** Tempo di risoluzione migliore fra tutte le configurazioni, per ogni istanza.

# Capitolo 4

## Conclusioni

### 4.1 Risultati

Dalle prove sperimentali dettagliate al capitolo precedente, è evidente come la linearizzazione KB proposta in questo elaborato possa ottenere risultati migliori rispetto alla formulazione più intuitiva. Certamente questo richiede un notevole aumento nella complessità delle tecniche utilizzate: senza separazione di tagli ai nodi il semplice modello si è infatti dimostrato poco pratico.

I metodi descritti e le implementazioni sviluppate permettono di risolvere più velocemente istanze di dimensioni prima ritenute fuori portata. In particolare, l'implementazione basata su CPLEX ottiene ottimi risultati in termini di tempo, mentre l'implementazione Branch & Bound *ad-hoc* mantiene una bassa impronta sul consumo di memoria.

### 4.2 Possibili miglioramenti

Le limitazioni di memoria sono state il principale ostacolo alla risoluzione di grandi istanze. L'implementazione *ad-hoc* nasce per ovviare al problema, ma costringe ad abbandonare i decenni di ingegnerizzazione portati dai principali risolutori. Tecniche che consentano di separare tagli ai nodi senza un grave impatto sulla memoria sarebbero di gran beneficio alle implementazioni proposte.

Di ulteriore ingegnerizzazione potrebbero beneficiare anche le implementazioni riportate, attraverso uno studio più sistematico dell'effetto che i metaparametri hanno sulle prestazioni dell'algoritmo. In particolare, le tecniche di questo elaborato sono state af-

finite sulle istanze a bassa densità, dal momento che la soluzione a un'istanza  $(n, m)$  è l'opposto della soluzione a densità inversa  $(n, n - m)$ .

### 4.3 Possibili sviluppi

Si ricorda che il modello 2.9 trattato da questo elaborato è solamente un caso specifico adattato alle istanze TaiC. Il modello originariamente proposto da Kaufman e Broeckx [11] si presta invece alla risoluzione di un qualsiasi problema di assegnamento quadratico. È possibile che alcune idee illustrate in questo elaborato possano essere generalizzate, e si rivelino utili anche per implementazioni MIP valide su qualsiasi QAP.

# Bibliografia

- [1] Domenico Salvagnin. *Introduzione all'ottimizzazione discreta*. Università degli Studi di Padova, 2018. URL: <https://www.dei.unipd.it/~salvagni/didattica/intro.pdf>.
- [2] G. B. Dantzig. «Origins of the simplex method». In: (1987). URL: <https://www.osti.gov/biblio/6340929>.
- [3] 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>.
- [4] Domenico Salvagnin. *Cenni di Programmazione Lineare Intera*. Università degli Studi di Padova, 2011. URL: <https://www.dei.unipd.it/~salvagni/didattica/mip>.
- [5] Tjalling C. Koopmans e Martin J. Beckmann. «Assignment Problems and the Location of Economic Activities». In: *Econometrica* 25 (1957), p. 53. URL: <https://api.semanticscholar.org/CorpusID:154940014>.
- [6] 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](https://www.researchgate.net/publication/220432228_P-complete_approximation_problems_J_ACM_233_555-565).
- [7] R.E. Burkard et al. *QAPLIB - A Quadratic Assignment Problem Library*. <https://www.opt.math.tugraz.at/qaplib/inst.html>. 2002.
- [8] Éric D. Taillard. «Comparison of iterative searches for the quadratic assignment problem». In: *Location Science* 3.2 (1995), pp. 87–105. DOI: [https://doi.org/10.1016/0966-8349\(95\)00008-6](https://doi.org/10.1016/0966-8349(95)00008-6). URL: <https://www.sciencedirect.com/science/article/pii/0966834995000086>.

- [9] Richard L. Church e Zvi Drezner. «Review of obnoxious facilities location problems». In: *Computers & Operations Research* 138 (2022), p. 105468. ISSN: 0305-0548. DOI: <https://doi.org/10.1016/j.cor.2021.105468>. URL: <https://www.sciencedirect.com/science/article/pii/S0305054821002197>.
- [10] Matteo Fischetti, Michele Monaci e Domenico Salvagnin. «Three Ideas for the Quadratic Assignment Problem». In: *Operations Research* 60 (ago. 2012), pp. 954–964. DOI: 10.2307/23260287.
- [11] L. Kaufman e F. Broeckx. «An algorithm for the quadratic assignment problem using Bender’s decomposition». In: *European Journal of Operational Research* 2.3 (1978), pp. 207–211. ISSN: 0377-2217. DOI: [https://doi.org/10.1016/0377-2217\(78\)90095-4](https://doi.org/10.1016/0377-2217(78)90095-4). URL: <https://www.sciencedirect.com/science/article/pii/0377221778900954>.
- [12] Yong Xia e Ya-xiang Yuan. «A new linearization method for quadratic assignment problem». In: *Optimization Methods & Software - OPTIM METHOD SOFTW* 21 (ott. 2006). DOI: 10.1080/10556780500273077.
- [13] P. C. Gilmore. «Optimal and Suboptimal Algorithms for the Quadratic Assignment Problem». In: *Journal of the Society for Industrial and Applied Mathematics* 10.2 (1962), pp. 305–313. DOI: 10.1137/0110022.
- [14] Tobias Achterberg. «Constraint Integer Programming». Tesi di dott. Gen. 2007. DOI: 10.14279/depositonce-1634.
- [15] Andrea Lodi e Andrea Tramontani. «Performance Variability in Mixed-Integer Programming». In: *Theory Driven by Influential Applications*. Cap. Chapter 1, pp. 1–12. DOI: 10.1287/educ.2013.0112. URL: <https://pubsonline.informs.org/doi/abs/10.1287/educ.2013.0112>.
- [16] IBM, ILOG CPLEX Optimization Studio 22.1.0. <https://www.ibm.com/docs/en/icos/22.1.0>.
- [17] <https://github.com/adario7/QAP>.
- [18] Mattia Toffolon. «Esperimenti MIP per una classe di problemi di assegnamento quadratico». Università degli Studi di Padova, 2023. URL: <https://hdl.handle.net/20.500.12608/48628>.