UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA TRIENNALE IN INGEGNERIA INFORMATICA

Esperimenti su reduced cost fixing in Programmazione Lineare Intera

Relatore

Prof. **Domenico Salvagnin**

Laureando

Christian Benetti

Anno accademico **2023 – 2024**

Data di laurea **19/03/2024**

Abstract

Il metodo di *Reduced Cost Fixing*, o più in generale *Reduced Cost Tightening*, rappresenta uno strumento migliorativo basilare impiegato in molti algoritmi risolutivi per problemi di programmazione lineare intera mista.

In questo studio, dopo aver opportunamente introdotto alcune nozioni fondamentali della programmazione lineare e il concetto di costo ridotto, sono stati svolti due esperimenti con lo scopo di verificare le performance della strategia in questione.

La maggior parte delle definizioni relative alla programmazione lineare derivano dalle dispense fornite dal prof. Domenico Salvagnin [12][13][14].

Gli esperimenti condotti hanno portato alla scrittura di uno script Python [11], tramite l'editor Visual Studio Code [8], attraverso il quale è stato possibile applicare l'algoritmo di fixing and tightening ad un pool di problemi di varie dimensioni e complessità.

Sono stati utilizzati due test set, *MIPLIB 2017* [9] e *setpart* che, grazie all'eterogeneità dei problemi in essi contenuti, hanno permesso di ricavare dati sperimentali il più possibile generali.

Attraverso il primo esperimento è stato possibile ottenere dei dati sull'efficienza dell'algoritmo in un *ambiente ideale*, mentre il secondo è stato utile per avere un'idea, sul possibile miglioramento delle performance, in caso di riformulazione della strategia di fixing tramite l'impiego di un *gap dimezzato*.

Attraverso i dati raccolti e i grafici riassuntivi, e comparativi, è stato infine possibile trarre alcune conclusioni ed evidenziare alcuni possibili sviluppi e impieghi futuri.

Indice

Introduzione

Questo capitolo ha l'obiettivo di porre le basi per la comprensione dei concetti che verranno discussi in seguito all'interno dell'elaborato.

I punti che verranno toccati rappresentano nozioni fondamentali della modellazione e dell'ottimizzazione di problemi di programmazione lineare.

1.1 Programmazione Lineare

Prima di passare alla definizione vera e propria di un problema di Programmazione Lineare è bene definire brevemente, al fine di avere la visione d'insieme, il concetto di "problema di ottimizzazione".

Un problema di ottimizzazione P consiste in: una funzione obiettivo a valori reali $f(x)$ da massimizzare o minimizzare (a seconda del tipo di problema), un insieme di variabili x definite all'interno del proprio dominio D e un insieme finito di vincoli S.

In generale x è una tupla di variabili $(x_1, \ldots, x_n)$ mentre il dominio D è un prodotto cartesiano $D_1 \times \ldots \times D_n$ tra i domini delle singole variabili, tali che $x_i \in D_i$.

Ogni $x \in D$ si dice *soluzione* di P. Una soluzione che soddisfi tutti i vincoli si dice *ammissibile*. Una soluzione ammissibile si dice *ottima* se rappresenta l'assegnamento di valori tale che la funzione obiettivo risulta massimizzata (o minimizzata).

L'insieme delle soluzioni ammissibili di un problema P si definisce con $F(P)$

Un problema di ottimizzazione si dice *risolto* quando viene trovata una soluzione ottima o quando si dimostra che esso è impossibile o illimitato.

Un problema di *Programmazione Lineare (LP)* consiste nella minimizzazione di una funzione lineare soggetta ad un numero finito di vincoli lineari, generalizzando si ottiene:

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

in cui:

- $\sim \in \{\leq, \geq, =\}$
- $l_j \in \mathbb{R} \cup \{-\infty\}, u_j \in \mathbb{R} \cup \{+\infty\}$

È bene notare come si definisca un problema di programmazione lineare unicamente come un problema di minimizzazione, viene usata questa notazione poiché di fatto è sempre possibile passare da una problema di max ad un problema di min, e viceversa, seguendo la semplice regola di trasformazione:

$$max(cx) = -min(-wx) \qquad (1.1)$$

D'ora in avanti verrà utilizzata questa notazione tenendo conto della proprietà appena enunciata.

1.1.1 Programmazione Lineare Intera

A partire dal concetto di programmazione lineare è possibile definire la Programmazione Lineare Intera.

Un problema di programmazione lineare intera (MIP) consiste nella minimizzazione di una funzione lineare soggetta ad un numero finito di vincoli lineari, con in più il vincolo che *alcune variabili* devono assumere valori interi.

In generale, quindi, si ha:

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

in cui:

$$\sim \in \{\leq, \geq, =\},\ l_j \in \mathbb{R} \cup \{-\infty\},\ u_j \in \mathbb{R} \cup \{+\infty\}$$

Esiste un'ulteriore classificazione dei problemi MIP in base alla struttura dell'insieme J.

Nel caso in cui $J = N$ si parla di *Programmazione Lineare Intera Pura*, altrimenti si parla di *Programmazione Lineare Intera Mista.*

Chiaramente quando $J = \emptyset$ si rientra nel caso della Programmazione Lineare.

Si può infatti notare come un problema di programmazione lineare intera sia di fatto un problema LP con l'aggiunta di un ulteriore vincolo, detto *vincolo di interezza*.

Questo concetto è fondamentale per la definizione di *Rilassamento Lineare* che verrà esposta in seguito.

1.2 Rilassamento

Il concetto di Rilassamento che verrà trattato nella seguente sottosezione è uno dei fondamenti comuni a tutte le strategie di ottimizzazione.

In sintesi la tecnica del rilassamento viene utilizzata, a partire da un problema iniziale, per ricavarne un problema più semplice, il problema così ottenuto, una volta risolto, potrà talvolta fornire informazioni utili alla risoluzione del problema principale.

Un rilassamento di un problema di ottimizzazione P è un nuovo problema di ottimizzazione R definito a partire da P apportando ad esso alcune delle seguenti modifiche:

1. rimozione di alcuni vincoli
2. sostituzione della funzione obiettivo $f(x)$ con un'approssimazione inferiore $g(x)$

Formalizzando, un rilassamento R avrà le seguenti caratteristiche:

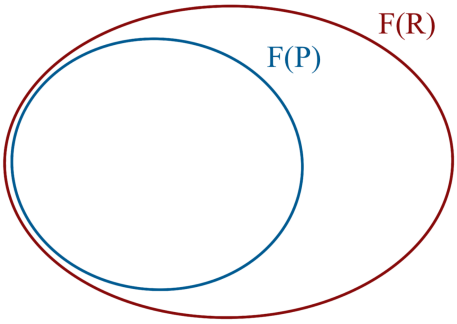
1. $F(R) \subseteq F(P)$
2. $g(x) \leq f(x) \quad \forall x \in F(P)$

Si noti che, nel caso di una problema di max le cose sarebbero leggermente diverse, in particolare la funzione obiettivo $g(x)$ costituirebbe un' approssimazione *superiore* a $f(x)$.

Come già detto, in questa tesi verrà usata quest' unica formulazione (P come problema di minimo) in virtù della regola (1.1).

Come detto in precedenza, la risoluzione di R permette di derivare alcune informazioni utili a seconda del caso in cui ci si trova.

La visualizzazione grafica degli insiemi delle soluzioni ammissibili dei due problemi rende più facile comprendere le intuizioni seguenti.



Tre sono i casi possibili:

1. $F(R) = \emptyset$, allora P è impossibile.
2. x^* è una soluzione ottima di R, $x^* \in F(P)$ e $g(x^*) = f(x^*)$, allora x^* è ottimo per P.
3. x^* è una soluzione ottima di R, $g(x^*)$ è un limite inferiore (lower bound) valido per il valore ottimo di P.

Proprio quest' ultima considerazione sarà fondamentale in seguito, formalizzando e ponendo x^*_R come soluzione ottima del rilassamento R otteniamo:

$$f(x^*) \geq g(x^*_R) \tag{1.2}$$

1.2.1 Rilassamento Lineare

Un particolare tipo di rilassamento, largamente impiegato per la risoluzione di problemi MIP, è il *rilassamento lineare.*

Il rilassamento lineare, detto anche rilassamento continuo, permette, a partire da un problema MIP, di ricavare un problema LP rimuovendo il vincolo di interezza sulle variabili.

Di solito viene utilizzato, anche all'interno di algoritmi di risoluzione più complessi, come segue:

1. Applicazione del rilassamento lineare.
2. Risoluzione all'ottimo del problema LP rilassato.
3. Ricerca, "nei dintorni" della soluzione ottima del rilassamento, della soluzione ottima intera.

1.3 Incumbent

Dato un problema di programmazione lineare P viene definita con il simbolo $\bar{x}$ una qualsiasi soluzione ammissibile nota, denominando F l'insieme di tutte le soluzioni ammissibili di P si avrà $\bar{x} \in F$.

A partire dalla definizione di $\bar{x}$ è possibile stabilire il concetto di *Incumbent.*

L'incumbent, o soluzione incombente, è il valore che corrisponde al costo della soluzione ammissibile nota $z = c(\bar{x})$, con z funzione obiettivo del problema di programmazione lineare preso in esame.

L'incumbent, quindi, rappresenta il valore che la funzione obiettivo z assume nel punto $\bar{x}$ di una soluzione ammissibile nota.

Si può facilmente dedurre come il valore di incumbent di un problema P permetta di definire un upper bound (problema di minimo) alla soluzione ottima del problema, infatti si ottiene:

$$f(x^*) \leq f(\bar{x}) \tag{1.3}$$

Dalla combinazione di (1.3) e di (1.2), che deriva dal concetto di rilassamento, si è in grado di definire formalmente un limite superiore (UB) e un limite inferiore (LB) alla soluzione ottima del problema:

$$g(x_R^*) \leq f(x^*) \leq f(\bar{x}) \tag{1.4}$$

Il concetto di incumbent, inoltre, trova largo impiego in alcuni algoritmi di ottimizzazione di problemi lineari interi, ad esempio, viene utilizzato nella strategia risolutiva denominata "Branch & Bound" (e nella sua variante "Branch & Cut").

L'utilizzo negli algoritmi citati in precedenza sfrutta sostanzialmente la proprietà (1.3), in sintesi un algoritmo $B\&B$ determina una prima soluzione ammissibile attraverso un euristica e utilizza il valore di incumbent corrispondente come valore di Upper Bound di riferimento.

Successivamente, attraverso un meccanismo di *divide-and-conquer*, suddivide il problema in sottoproblemi e ne risolve il rilassamento.

Un confronto tra questi valori e l'incumbent corrente permette di evitare l'esplorazione di regioni dello spazio delle soluzioni che non contengono soluzioni migliori di quelle note.

Il valore di incumbent viene via via aggiornato in modo da rappresentare sempre il costo della migliore soluzione ammissibile nota.

1.4 Costi ridotti

Il concetto di costo ridotto è alla base degli esperimenti e delle conclusioni riportate in questo elaborato, è quindi di fondamentale importanza fare un excursus sull'origine di tale strumento e sull'utilizzo all'interno delle varie tecniche di ottimizzazione.

Per arrivare alla definizione di costo ridotto è bene delineare correttamente alcuni punti, viene data una formulazione alternativa dei problemi di programmazione lineare:

$$min\ cx$$
$$Ax \geq b,\ x \geq 0$$

È importante notare come questo enunciato utilizzi il concetto di matrice, nello specifico:

- x corrisponde ad una tupla di variabili: $(x_1, ..., x_n)$ che può essere visualizzata come una matrice $n \times 1$.
- c è una matrice $1 \times n$, in cui ogni elemento c_i è il cosiddetto costo associato alla variabile x_i.
- A è una matrice $m \times n$ detta matrice dei vincoli.
- b è una matrice $m \times 1$.

Inoltre va evidenziato come il vincolo $x \geq 0$ non comporti alcuna perdita di generalità, infatti si dimostra che è sempre possibile sostituire una variabile x_j, inizialmente senza restrizioni di segno, con l'espressione $x_j^+ - x_j^-$, introducendo una nuova coppia di variabili:
$x_j^+ \geq 0,\ x_j^- \geq 0$

Viene poi effettuata un'ultima trasformazione del modello, anch'essa senza perdita di generalità, per fare ciò si tiene conto della proprietà che permette di convertire un qualsiasi problema di programmazione lineare con vincoli "$\geq$" in uno con vincoli "$=$".

Ovvero:

$$a_i x \geq b_i \iff \begin{cases} a_i x - s_i = b_i \\ s_i \geq 0 \end{cases}$$

La variabile s_i viene detta *variabile surplus.*

Otteniamo infine il modello:

$$\begin{aligned} & min\ cx \\ & Ax = b, x \geq 0 \end{aligned} \tag{1.5}$$

Si definisce rango di una matrice il numero massimo di colonne linearmente indipendenti della stessa.

Se la matrice A ha rango m ciò implica che ogni insieme di m colonne linearmente indipendenti forma una base per lo spazio vettoriale generato dalle colonne della matrice.

Da tale osservazione segue la definizione di *soluzione di base*, o *basic solution.*

Una *soluzione di base* x di $Ax = b$ è una soluzione che "usa" unicamente le colonne base (ovvero che formano la base) nella combinazione lineare Ax.

Per rendere più chiaro il concetto è bene visualizzare il prodotto matriciale in questione:

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \cdots & \cdots & \cdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \times \begin{bmatrix} x_1 \\ \cdots \\ x_n \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + \cdots + a_{1n}x_n \\ \cdots \\ a_{m1}x_1 + \cdots + a_{mn}x_n \end{bmatrix}$$

Essendo, il prodotto matriciale, un prodotto riga per colonna è evidente come la matrice contenente le variabili di sistema debba essere tale che $x_j = 0$ quando la colonna j di A non è parte della base.

Questo significa che ci saranno al più m variabili positive in una soluzione di base, mentre le restanti $n - m$ variabili saranno nulle.

Le m variabili non nulle sono dette *variabili di base*, o *in base*.

Le restanti $n - m$ variabili a valore zero sono dette *variabili fuori base*.

Per identificare la soluzione di base corrispondente ad una data base, occorre, a partire dalla matrice A, rimuovere le righe linearmente dipendenti e partizionare la matrice risultante in:

- Una matrice quadrata B, formata dalle sole colonne di A facenti parte della base.
- Una sottomatrice N formata dalle rimanenti $n - m$ colonne.

Verranno partizionate seguendo il medesimo comportamento anche le matrici x e c, nello specifico:

- x_B conterrà le variabili di base, e c_B i costi associati ad esse.
- x_N le variabili fuori base, e c_N i costi associati.

A questo punto è possibile riscrivere il modello (1.5) come segue:

$$\begin{aligned} min\ & c_B x_B + c_N x_N \\ & B x_B + N x_N = b,\ x_B, x_N \geq 0 \end{aligned} \tag{1.6}$$

Ora, a partire dal sistema precedente si pone x_B in funzione di x_N :

$$x_B = B^{-1}b - B^{-1}N x_N \tag{1.7}$$

Questo permette di dire che, nel caso in cui x_N sia uguale a 0, (sempre vero nelle soluzioni di base) la soluzione $(x_B, x_N) = (B^{-1}b, 0)$ sarà, appunto, di base, poiché al più m variabili risulteranno essere positive.

Inoltre, se $B^{-1}b \geq 0$, $(x_B, x_N) = (B^{-1}b, 0)$ rappresenta una *soluzione di base ammissibile.*

Attraverso i passaggi che verranno mostrati in seguito è possibile dimostrare come, a partire da un qualsiasi problema LP che ammetta soluzione ottima, almeno una delle soluzioni di base sia ottima.

Ciò significa che, per risolvere un problema LP, sarà sufficiente esaminare le soluzioni di base e verificare quali di queste siano ottimali, restringendo di molto lo spazio e i tempi di ricerca dell'ottimo.

La funzione obiettivo del modello (1.6) può essere riscritta in funzione delle variabili fuori base x_N, sfruttando (1.7), si otterrà:

$$z = c_B B^{-1} b + (c_N - c_B B^{-1} N) x_N$$

da cui

$$z = c_B B^{-1} b + r x_N \tag{1.8}$$

In quest'ultima formulazione si nota come $c_B B^{-1} b$ sia la funzione obiettivo nei punti della soluzione di base $(x_B, x_N) = (B^{-1} b, 0)$.

Inoltre si può finalmente definire il concetto di *vettore dei costi ridotti*:

$$r = (c_N - c_B B^{-1} N) \tag{1.9}$$

È bene specificare come spesso si utilizzi la terminologia *costo ridotto,* di una determinata variabile, per indicare l'elemento ad essa associato all'interno del vettore dei costi ridotti.

Dal punto di vista della nomenclatura è evidente il perché di tale nome, infatti, si nota come il costo ridotto di una variabile fuori base sia calcolato "riducendo" il costo di tale variabile di un determinato valore $c_B B^{-1} N$ dipendente dalla base scelta.

A partire da (1.8) è poi possibile fare una serie di importanti considerazioni:

- I costi ridotti di tutte le variabili in base sono, per definizione, nulli.
- Se $r \geq 0$, la soluzione di base ammissibile in questione è ottima.

  Infatti, modificando il valore di una variabile fuori base (si ricordi il vincolo $x \geq 0$ per cui la variabile posta a 0 può solo aumentare) la funzione obiettivo aumenterebbe di conseguenza, portando ad un peggioramento.

Teorema (1.0): Una soluzione di base $(x_B, x_N) = (B^{-1} b, 0)$, del problema LP (1.6), è ottima se il vettore dei costi ridotti $r \geq 0$, con r definito come (1.9).

1.4.1 Variante del modello LP

Infine si può considerare una variazione del modello LP (1.5), ponendo un lower e un upper bound alle variabili:

$$\begin{aligned} &min\ cx \\ &Ax = b,\ l \leq x \leq u \end{aligned} \tag{1.10}$$

Tale modello sarà di riferimento, nei capitoli successivi, per le applicazioni dei metodi di Reduced Cost Fixing e Tightening, in quanto può essere visto come una generalizzazione del modello (1.5).

Più precisamente si può vedere (1.5) come un problema (1.10) in cui non esiste upper bound e il valore '0' rappresenta il lower bound a tutte le variabili di sistema.

In questo modello:

- Una variabile viene considerata *fuori base* quando corrispondente ai valori di lower o upper bound da cui è limitata ($x_j = l_j$ o $x_j = u_j$), mentre *in base* in tutti gli altri casi.
- Una soluzione si può definire *di base* quando $n - m$ variabili sono ad uno dei propri bound.

Come in precedenza modificare (aumentare) il valore di una variabile fuori base che si trovi al proprio lower bound implica un peggioramento della funzione obiettivo nel caso in cui $r_j \geq 0$

Inoltre, rispetto a prima, modificare una variabile fuori base che si trovi già al suo upper bound (quindi diminuirne il valore, visto che la variabile è al suo limite superiore) porta ad un peggioramento nel caso di $r_j \leq 0$, formalizzando:

Corollario (1.0): Una soluzione di base (x_B, x_N) di un problema (1.10), è ottima se:

1. $r_j \geq 0$ per ogni x_j variabile fuori base di valore l_j
2. $r_j \leq 0$ per ogni x_j variabile fuori base di valore u_j

Con r definito come (1.9).

Grazie a quanto detto si nota come le soluzioni di base ammissibili siano *vertici* dell'insieme di ammissibilità del problema lineare.

La ricerca della soluzione ottima tra le sole soluzioni di base, corrisponde, quindi, alla restrizione della ricerca sui soli vertici del *poliedro* formato dai vincoli lineari.

Il metodo del *simplesso* utilizza proprio questi concetti per "saltare" da un vertice all'altro al fine di trovare la soluzione ottima. In particolare, esamina una sequenza di soluzioni di base ammissibili ottenute incrementando il valore di una variabile fuori base con *costo ridotto negativo*.

La soluzione ottima viene trovata, e l'esecuzione termina, quando tutti i costi ridotti risultano non negativi.

Reduced Cost Fixing and Tightening

Il focus di questo secondo capitolo è posto sul metodo di "Reduced Cost Fixing and Tightening", procedura su cui si basa l'intera sezione sperimentale dell'elaborato.

I punti principali saranno quindi:

- Citare le origini di tale strategia e come essa venga attualmente impiegata.
- Delineare gli obiettivi che il metodo si prefigge, nei vari casi, e in che contesti esso possa essere impiegato.
- Definire la procedura tramite cui il metodo viene portato a compimento e dimostrarne la correttezza da un punto di vista matematico.

2.1 Panoramica

Uno dei primi impieghi, se non esattamente il primo, del metodo di reduced cost fixing risale alla pubblicazione da parte di *George Dantzig, Ray Fulkerson, e Selmer Johnson* dell'articolo *"Solution of a large-scale traveling-salesman problem"* pubblicato nel 1954 sul "*Journal of the operations research society of America".*

In questo articolo di ricerca gli autori propongono una soluzione ad uno dei problemi più classici dell'informatica teorica, quello del *commesso viaggiatore*, o *travelling salesman*

Tale problema richiede, in breve, data una serie di città e la distanza tra ogni coppia delle stesse, di trovare il percorso più breve possibile che permetta di visitarle tutte una e una sola volta e ritornare, infine, a quella di partenza.

Dandone per la prima volta una formulazione come problema di programmazione lineare intera e sfruttando il concetto reduced cost fixing *Dantzig, Fulkerson e Johnson* sono stati in grado di risolvere un'istanza del problema con 49 città.

Attualmente, tale strategia è alla base di molti risolutori di problemi MIP.

È bene, prima di entrare maggiormente nello specifico, spiegare sommariamente il significato di *Reduced Cost Fixing and Tightening*.

I metodi di reduced cost fixing e reduced cost tightening (o strengthening), che per comodità è possibile comprimere, rispettivamente, in RCF e RCT, sono delle strategie attraverso cui è possibile, a partire da un problema di programmazione lineare intera, restringere il dominio di alcune variabili (RCT) o in alcuni casi fissarle ad un determinato valore (RCF).

È bene chiarire come, nonostante si faccia riferimento a RCF e RCT come a due strategie separate, la procedura che viene applicata sia in realtà la stessa, tale strategia ha infatti lo scopo di "irrobustire" il modello in questione (strengthening) rendendo meno complessa e dispendiosa la ricerca dell'ottimo.

L'applicazione della stessa strategia, infatti, permette di concentrare la ricerca del valore ottimo di una determinata variabile su un insieme di elementi più ristretto rispetto al dominio originale, talvolta tale restrizione porta a poter definire con certezza, quindi a fissare, il valore di una variabile.

2.2 Interpretazione geometrica dei costi ridotti

Per comprendere come si possa sfruttare il concetto di costo ridotto, spiegato nel capitolo precedente, per limitare lo spazio di ricerca, occorre introdurre quella che può essere definita *interpretazione geometrica* dei costi ridotti.

Finora, infatti, è stata discussa l'origine e la definizione formale di costo ridotto senza, però concentrare l'attenzione sul perché questo concetto sia tanto importante nell'ambito dell'ottimizzazione di problemi LP e MIP.

Per capire ciò è bene richiamare qui un paio di definizioni date in precedenza (1.9):

$$r = (c_N - c_B B^{-1} N)$$

definizione formale di vettore dei costi ridotti, e (1.8):

$$z = c_B B^{-1} b + r x_N$$

funzione obiettivo espressa in funzione di una determinata base e delle variabili fuori base x_N

Inoltre, per semplicità, verrà utilizzato (1.6) come modello di riferimento per le successive dimostrazioni in questa sottosezione, invece della sua generalizzazione (1.10).

È comunque dimostrabile come ciò che segue sia valido anche per il modello più generale.

Tramite (1.8) e il concetto di soluzione di base $(x_B, x_N) = (B^{-1}b, 0)$ è possibile notare come la funzione obiettivo ad essa associata sia $c_B B^{-1} b$.

Tale valore, che rappresenta la funzione obiettivo del problema nei punti della soluzione di base attuale, viene espresso come:

$$z_B = c_B B^{-1} b \tag{2.0}$$

si può quindi riscrivere (1.8) come:

$$z = z_B + r x_N \tag{2.1}$$

Questa riformulazione della funzione obiettivo del problema rende chiaro come essa sia formata, in effetti, da due elementi separati, z_B dipendente dalla base scelta e $r x_N$ dipendente dalle variabili fuori base e dai relativi costi ridotti.

È bene inoltre ricordare come, (1.6), anche le stesse variabili in base x_B siano dipendenti dalle variabili fuori base x_N oltre che dalla base stessa.

Inoltre, essendo $r x_N$ prodotto tra matrici questo può essere espanso mostrando i singoli elementi che compongono la somma, sempre riscrivendo (2.1) si ottiene:

$$z = z_B + (r_1 x_{N1} + \cdots + r_{n-m} x_{Nn-m})$$

Proprio da quest'ultima formulazione della funzione obiettivo viene l'intuizione fondamentale del metodo del reduced cost fixing, infatti, si nota facilmente come, a partire da una soluzione di base, la variazione unitaria di una qualsiasi variabile fuori base di indice *i* porti ad una variazione r_i del valore z.

Si può quindi definire il costo ridotto r_i come l'incremento marginale del costo complessivo (z) per ogni unità di variazione in aumento della variabile associata.

Questa nuova definizione rende evidente il teorema (1.0), infatti si nota come, nel caso in cui si avesse un costo ridotto $r_i < 0$ associato ad una variabile fuori base x_{Ni}, sarebbe sufficiente aumentare il valore di tale variabile per avere una nuova funzione obiettivo di valore $z' < z$, migliore della soluzione precedente (essendo $r_i x_{Ni} < 0$).

La soluzione di base non sarebbe quindi, per definizione, ottima.

Per estensione ciò dimostra anche il corollario (1.10) relativo al modello limitato da upper e lower bound, infatti, definendo con Δx_{Ni} la variazione di valore della variabile e x'_{Ni} il nuovo valore assunto dalla stessa, si avrà:

1. Data una variabile x_{Ni} fuori base a valore l_i con $r_i > 0$ aumentarne il valore porterebbe ad una variazione $r_i \Delta x_{Ni} > 0$, data da $\Delta x_{Ni} = (x'_{Ni} - l_i) > 0$, che peggiorerebbe la soluzione.
2. Data una variabile x_{Ni} fuori base a valore u_i con $r_i < 0$ diminuirne il valore porterebbe ad una variazione $r_i \Delta x_{Ni} > 0$, data da $\Delta x_{Ni} = (x'_{Ni} - u_i) < 0$, che peggiorerebbe la soluzione.

In entrambi i casi, quindi, la soluzione di base precedente si dimostrerebbe essere ottima.

2.3 Descrizione del metodo di Tightening

Dopo aver compreso tutti gli strumenti necessari si è finalmente in grado di illustrare la vera e propria procedura per il tightening delle variabili.

È bene ricordare come la procedura di reduced cost tightening venga applicata a partire da un problema di programmazione lineare intera denominato P.

In primo luogo, quindi, a partire da un problema MIP, è necessario trovare, attraverso un' euristica, una soluzione ammissibile a cui è associato il valore $\bar{z} = c\bar{x}$ (*incumbent)*.

Come visto in precedenza tale valore limita superiormente la funzione obiettivo del problema, da (1.3):

$$c^T x \leq \bar{z}$$

A questo punto a P viene applicato un rilassamento lineare, ottenendo il problema LP R.

Il nuovo problema così ottenuto viene risolto trovando una soluzione ottima di base, denominata x^*_R.

Si nota come tale soluzione, essendo di base, corrisponda ad un valore di funzione obiettivo pari a $c_B B^{-1} b$, che verrà chiamato z^*_R, da (2.1):

$$z_R = z^*_R + r x_N \qquad (2.2)$$

Il valore z_R rappresenta, quindi, il costo di ogni soluzione del rilassamento R.

Ora, utilizzando quanto dimostrato in (1.4) si può facilmente intuire come il valore di ogni funzione obiettivo del rilassamento R sia sempre limitato superiormente dall' incumbent del problema MIP originario, ottenendo così:

$$z_R^* + rx_N \leq \bar{z} \tag{2.3}$$

Da qui è bene considerare separatamente i due casi.

Variabili a Lower Bound

Nel caso di una variabile x_i, che si trovi fuori base al suo valore di lower bound, con l_i e $r_i \geq 0$ dato il nuovo valore x_i' si avrà, da (2.3) :

$$z_R^* + r(x_i' - l_i) \leq \bar{z}$$

mantenendo la sola variabile x_i sul lato sinistro dell'equazione si ottiene:

$$x_i \leq l_i + \frac{\bar{z} - z_R^*}{r_i} \tag{2.4}$$

È dunque possibile, a partire da una variabile fuori base a lower bound, applicare un nuovo upper bound.

Si nota, inoltre, come nel caso particolare in cui la variabile x_i, fuori base a l_i,fosse originariamente, nel problema MIP, sottoposta al vincolo di interezza, sarebbe possibile approssimare il nuovo upper bound alla sua parte intera, si avrebbe quindi:

$$x_i \leq \left\lfloor l_i + \frac{\bar{z} - z_R^*}{r_i} \right\rfloor \tag{2.5}$$

Variabili ad Upper Bound

Nel caso, invece, di una variabile x_i fuori base ad upper bound u_i, con $r \leq 0$ e nuovo valore x'_i, si ottiene, scambiando i segni e l'ordine delle operazioni per ottenere fattori sempre positivi:

$$z^*_R + r(u_i - x'_i) \leq \bar{z}$$

da cui:

$$x_i \geq u_i + \frac{\bar{z} - z^*_R}{r_i} \tag{2.6}$$

e in caso di variabile, originariamente, intera x_i, applicando questa volta la parte intera superiore si avrebbe:

$$x_i \geq \left\lceil u_i + \frac{\bar{z} - z^*_R}{r_i} \right\rceil \tag{2.7}$$

2.3.1 Fixing delle variabili

Dopo aver considerato il caso di tightening delle variabili è possibile prendere in esame il caso particolare, noto come *Reduced Cost Fixing* che permette di fissare le stesse ad un valore specifico.

È infatti possibile fissare il valore di una variabile nel caso in cui il bound generato tramite la regola con i costi ridotti, illustrata in precedenza, corrisponda al, già presente, bound a cui essa era limitata.
Definendo u'_i il bound generato da (2.4) o (2.5) e l'_i quello generato da (2.6) o (2.7), è possibile fissare un valore, nei 2 casi, quando:

$$u'_i = l_i \qquad \text{oppure} \qquad l'_i = u_i$$

Nel caso di una variabile x_i intera è sempre possibile, tenendo conto di (2.5) e (2.7), fissare una variabile ad uno dei suoi bound nel caso in cui:

$$|r_i| \geq \bar{z} - z^*_R$$

Esperimenti

Questo terzo capitolo ha l'obiettivo di presentare i due esperimenti, condotti nell'ambito del reduced cost fixing and tightening, e i relativi risultati sperimentali ottenuti corredati da grafici.

3.1 Descrizione

Entrambi gli esperimenti sono stati compiuti utilizzando il linguaggio di programmazione Python [11] nella sua versione 3.9.13 in un ambiente Windows 10 con l'ausilio di Visual Studio Code 1.84 [8], un software di editing di codice sorgente che permette l'integrazione, attraverso varie estensioni, di strumenti di debugging.

Sono stati utilizzati vari pacchetti Python, nello specifico: *numpy* [10] per la gestione di alcune strutture dati, *math* per le operazioni matematiche di parte intera e parte intera superiore e *matplotlib* [7] per la visualizzazione grafica di tutti i dati sperimentali ottenuti.

Inoltre per la risoluzione dei problemi di programmazione lineare è stata utilizzata l'interfaccia per Python del software IBM CPLEX [6] nella versione 22.1.1.

Come sarà reso evidente in seguito gli esperimenti non mirano a testare l'efficienza temporale del metodo di ottimizzazione in esame, motivo per cui le specifiche tecniche del computer su cui sono stati svolti sono irrilevanti, quindi omesse.

Lo scopo degli esperimenti condotti è quello di valutare la capacità, della strategia basata sui costi ridotti, di ridurre il dominio delle variabili e, soprattutto, di fissarne il valore.

A partire da due set di istanze di problemi MIP i test saranno in grado, tramite l'applicazione della procedura espressa nel capitolo precedente, di contare il numero di variabili che, per ogni problema, è stato possibile "limitare" o "fissare".

È stato scelto di testare tale strategia su due diversi test set in modo da avere una panoramica più generale sulla sua efficienza.

Ciò che differenzia tra loro i due esperimenti proposti in seguito, come sarà approfondito nelle sezioni specifiche, è sostanzialmente il valore di *gap* scelto per l'applicazione della strategia di strengthening.

Così, infine, sarà possibile valutare i miglioramenti apportati dalla modifica dell'incumbent in entrambi gli ambienti.

3.2 Strumento IBM Cplex

ILOG CPLEX Optimization Studio è un pacchetto di software di ottimizzazione di proprietà di IBM.

Nonostante sia disponibile, su licenza, per l'utilizzo in ambito aziendale, è possibile, da studente o docente universitario, usufruire di una versione gratuita illimitata attraverso l'IBM Academic Initiative.

In generale CPLEX permette di risolvere problemi di ottimizzazione con vincoli lineari o quadratici in cui la funzione obiettivo possa essere espressa in forma lineare o quadratica convessa.

Nel caso specifico, in questa tesi, è stato impiegato unicamente, tramite le API disponibili per Python, per la risoluzione di problemi LP.

È stato deciso di utilizzare CPLEX poiché è in grado di risolvere problemi LP e MIP, anche di grandi dimensioni, e a causa del suo elevato grado di flessibilità, è infatti possibile impostare una serie di parametri che modificano l'algoritmo di ottimizzazione.

Esistono, attraverso l'interfaccia di Python [2], vari modi per modificare tali settaggi, nello specifico si è scelto di caricare, attraverso la funzione *read_file('settings.prm'),* il file contenente i settings in formato prm.

I settaggi utilizzati nel caso in esame sono i seguenti.

3.2.1 Solution Type

CPXPARAM_SolutionType 1

Attraverso questo parametro è possibile decidere il tipo di soluzione ritornata dall'ottimizzatore.

In questo caso il valore di parametro '1' forza CPLEX a trovare una soluzione ottima di base.

3.2.2 Optimality tolerance

CPXPARAM_Simplex_Tolerances_Optimality *1e-9*

Come spiegato nella sezione 1.4, il metodo del simplesso, che viene impiegato da CPLEX per risolvere problemi LP, interrompe la propria esecuzione nel momento in cui i costi ridotti di tutte le variabili della soluzione di base trovata risultano non negativi.

La soluzione trovata in tale modo, tuttavia, è la soluzione ottimale da un punto di vista prettamente teorico, durante un approccio reale alla risoluzione di un problema CPLEX utilizza il valore del parametro *optimality tolerance* per determinare se un costo ridotto sia non negativo.

Nello specifico, CPLEX considera non negativo un costo ridotto *negativo* avente valore assoluto inferiore alla tolleranza di ottimalità, tale parametro rappresenta, quindi, una sorta di valore di "sensibilità", tanto più piccolo sarà il valore tanto più la soluzione trovata si avvicinerà a quella teoricamente ottimale.

Per gli esperimenti condotti è stato scelto il valore minimo di optimality tolerance: 1e-9.

3.3 Istanze di Benchmark

Allo scopo di testare l'algoritmo di reduced cost strengthening e di ricavare dei dati, quanto più possibile generali, sull'efficienza dello stesso, si è scelto di utilizzare due diversi test set.

Ogni set di dati di *benchmark*, quindi, è composto da una serie di problemi di programmazione lineare intera di diversa natura e di diversa difficoltà.

È bene specificare che, come si vedrà in seguito, gli esperimenti sono stati svolti in un ambiente "teorico" in cui è già noto il valore di costo ottimo di ogni problema MIP.

Nello specifico sono stati utilizzati i seguenti test set.

3.3.1 MIPLIB 2017

MIPLIB è una libreria di problemi di programmazione lineare intera creata nel 1992 in risposta al bisogno, da parte dei ricercatori di tutto il mondo, di avere accesso ad un largo database di problemi MIP.

Tale libreria ha subito nel corso del tempo varie modifiche ed è giunta alla sua sesta edizione, attualmente rappresenta lo standard per la comparazione delle performance degli ottimizzatori MIP.

Sono disponibili due test set: Collection Set e Benchmark Set.

Si è scelto di utilizzare benchmark set nella sua versione più recente (MIPLIB 2017) in quanto pensato appositamente per la valutazione delle performance degli algoritmi di ottimizzazione e composto da 240 problemi MIP di varia difficoltà.

L'intero set è stato filtrato rimuovendo 7 problemi inammissibili, ottenendo un nuovo test set di 233 elementi così composto:

- 199 problemi definiti, da MIPLIB, "easy", ovvero risolvibili in meno di un'ora utilizzando al più 16 threads, un risolutore e un hardware "standard".
- 34 problemi, "hard" di complessità maggiore.

I modelli del set sono lineari interi misti, o puri, e di grandi dimensioni, infatti sono, in media, composti da circa 52086 variabili di cui 31773 binarie e 1343 intere.

I file contenenti i problemi vengono forniti in formato *MPS*, standard per la rappresentazione di problemi LP e MIP, e corredati da un foglio di calcolo che descrive nel dettaglio le varie caratteristiche di ogni istanza, compreso il valore di costo della soluzione ottima migliore attualmente trovata, e i nomi dei ricercatori che le hanno create e condivise.

3.3.2 setpart

È stato poi scelto di ripetere ogni test su un ulteriore set di problemi MIP fornito dal professor. Domenico Salvagnin.

I problemi in esso contenuti sono istanze di set partitioning rilevanti in un approccio a generazione di colonne per problemi di routing

Tale test set è composto da 50 problemi di crescente dimensione e complessità, ogni problema è mediamente formato da 47639 variabili binarie.

L'archivio del set è composto da una serie di file, in formato *lp,* corredati da un documento che associa ad ogni istanza di problema il costo della soluzione ottima di riferimento.

3.4 Primo esperimento

In questa sezione viene descritto in dettaglio il primo dei due esperimenti condotti.

3.4.1 Descrizione

Come annunciato lo scopo degli esperimenti è quello di testare il *reduced cost fixing and tightening* su un largo insieme di problemi, per fare ciò si è scelto, in questo primo test, di utilizzare come valore di incumbent per il calcolo dei bound delle variabili, (2.4) o (2.6), il valore di costo ottimo del problema.

Tale esperimento, quindi, nonostante sia condotto in un ambiente "ideale", in cui è già noto fin dall'inizio il valore di costo della soluzione ottima, permette di calcolare un bound *matematicamente corretto*, essendo tale valore a tutti gli effetti un incumbent valido per il problema MIP in esame.

3.4.2 Codice

Il codice Python implementa le seguenti fasi:

1. Lettura, da un file di testo, dei nomi dei modelli del set e del loro costo ottimo,
2. Apertura di ogni singolo problema del test set, calcolo del bound su ogni variabile fuori base e conteggio del numero di variabili *tightened* e *fixed.*
3. Scrittura su un file di testo dei risultati ottenuti.
4. Tracciamento di grafici riassuntivi a partire dai file generati.

Il software utilizza una serie di variabili globali rappresentanti dei dizionari che associano ad ogni nome di problema del test set (chiave) un relativo valore numerico (valore).

Nello specifico: *fileName* associa i valori di costo ottimo forniti, *var* il numero di variabili totali del problema, *tightened* e *fixed* contano rispettivamente il numero di variabili ristrette e fissate alla fine dell'esecuzione dell'algoritmo.

Inoltre si usa una funzione lambda che codifica le formule (2.4)/(2.6) per il calcolo dei bound.

Inizialmente viene creata un'istanza di classe Cplex, che rappresenta un modello di programmazione matematica, e vengono letti i settings dell'ottimizzatore da file, in Python:

```
cpxRel = Cplex()
cpxRel.parameters.read_file('settings.prm')
```

Successivamente i nomi dei problemi MIP appartenenti al test set in esame vengono letti da file.

Esistono due file di testo, uno per ogni set di istanze, denominati "miplib2017.txt" e "setpart.txt", formattati in righe così composte:

nome_problema #costo_soluzione_ottima

Scorrendo le righe, vengono letti dal file i nomi dei modelli (corrispondenti ai nomi dei file da aprire), inizializzati di conseguenza i dizionari e impostati a valore iniziale '0' i contatori.
A questo punto è possibile visitare tutti gli elementi della struttura dati *fileName* e leggere, uno per volta e secondo il formato indicato come parametro, i modelli salvati su file tramite l'apposita funzione *read* di cplex.
Inoltre, tramite le funzioni dell'interfaccia Python

- `cpxRel.variables.get_types()`
- `cpxRel.set_problem_type(cpxRel.problem_type.LP)`

viene prima inizializzata una lista che salva il tipo di ogni variabile del problema associato al relativo indice e successivamente viene applicato il rilassamento lineare al modello.

Infine, viene chiamata l'apposita funzione per il calcolo dei bound.

Dopo aver risolto il rilassamento all'ottimo, tramite *cpxRel.solve()*, vengono impostati dei parametri utilizzando alcune funzioni messe a disposizione dall'interfaccia di Cplex, in Python:

- `fileName[cpxRel.get_problem_name()]` costo ottimo noto, del problema MIP, da utilizzare come incumbent
- `cpxRel.solution.get_objective_value()` costo soluzione ottima del rilassamento
- `cpxRel.solution.get_reduced_costs()` vettore dei costi ridotti associato alla soluzione
- `cpxRel.solution.basis.get_basis()` lista che fornisce lo status (lower bound, upper bound o in base) delle variabili, della soluzione corrente, rappresentato tramite valore intero.

Si utilizzano, quindi, altre due liste contenenti gli indici di tutte le variabili a lower e ad upper bound nella soluzione di base.

A questo punto è infine possibile scorrere, separatamente, tutte le variabili a lower e ad upper bound, tramite gli indici memorizzati nelle apposite liste, calcolare gli effettivi bound e contare le variabili fissate e ristrette.

A causa del concetto di optimality tolerance utilizzato da Cplex è necessario verificare che le variabili abbiano effettivamente il costo ridotto atteso (a lower bound: positivo se problema di *min*, negativo se problema di *max*).

In caso di variabili fuori base ad upper bound vale l'opposto (negativo se problema di *min*, positivo se problema di *max*).

Una volta fatto ciò il bound viene calcolato, tramite l'opportuna funzione lambda e, in caso di variabili intere o bianarie, arrotondato a parte intera (o parte intera superiore, a seconda del caso).

Al fine di contare le variabili *fixed* il limite precedentemente calcolato viene confrontato con il bound (opposto) preesistente o, nel caso di variabili binarie, con il bound *implicito* esistente per definizione (lb:0, ub:1).

Si noti come ogni variabile fissata venga contata anche come ristretta, avendo considerato il reduced cost fixing come un particolare caso di tightening.

3.4.3 Risultati e grafici sperimentali

I risultati ottenuti vengono scritti su file di testo formattato riga per riga come segue:

nome_problema #var_fissate #var_ristrette #var_totali

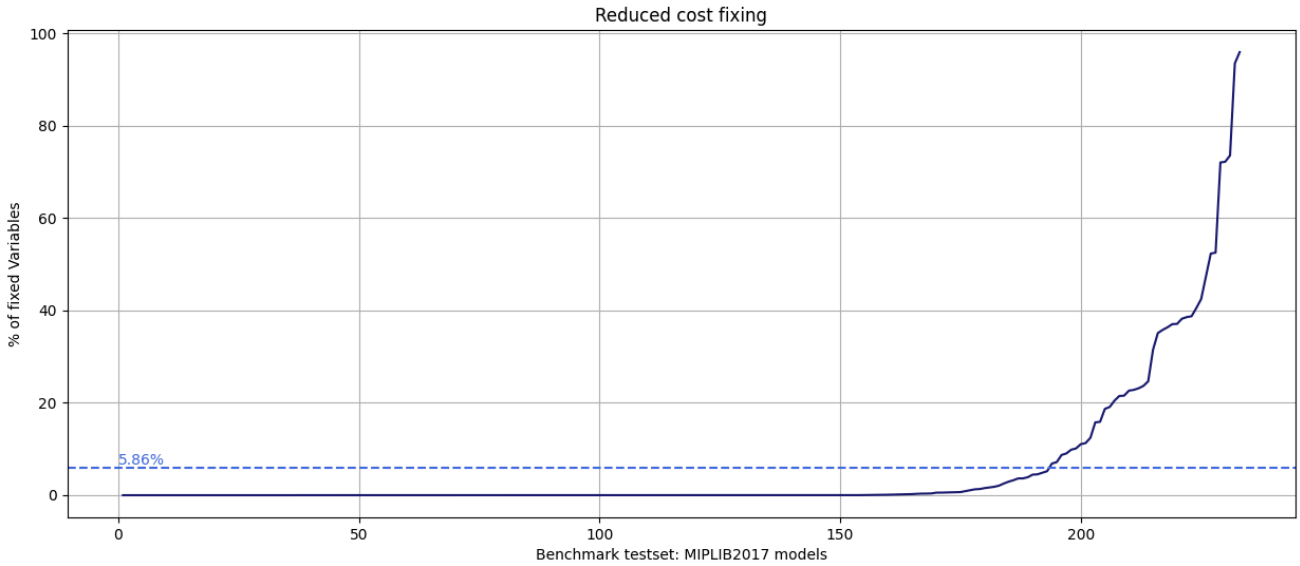
Data la grande mole di istanze non sarebbe possibile riportare i risultati sperimentali così acquisiti, che sono, tuttavia, messi a disposizione in doppio formato, *txt* e *csv*, all'interno della repository [1] nella sottocartella denominata *"Experiment_1"*.

A partire dai dati grezzi viene poi calcolata la percentuale di variabili ristrette e fissate per ogni problema, tali valori, una volta ordinati per ottenere delle funzioni monotone, vengono utilizzati per il tracciamento dei grafici riassuntivi.

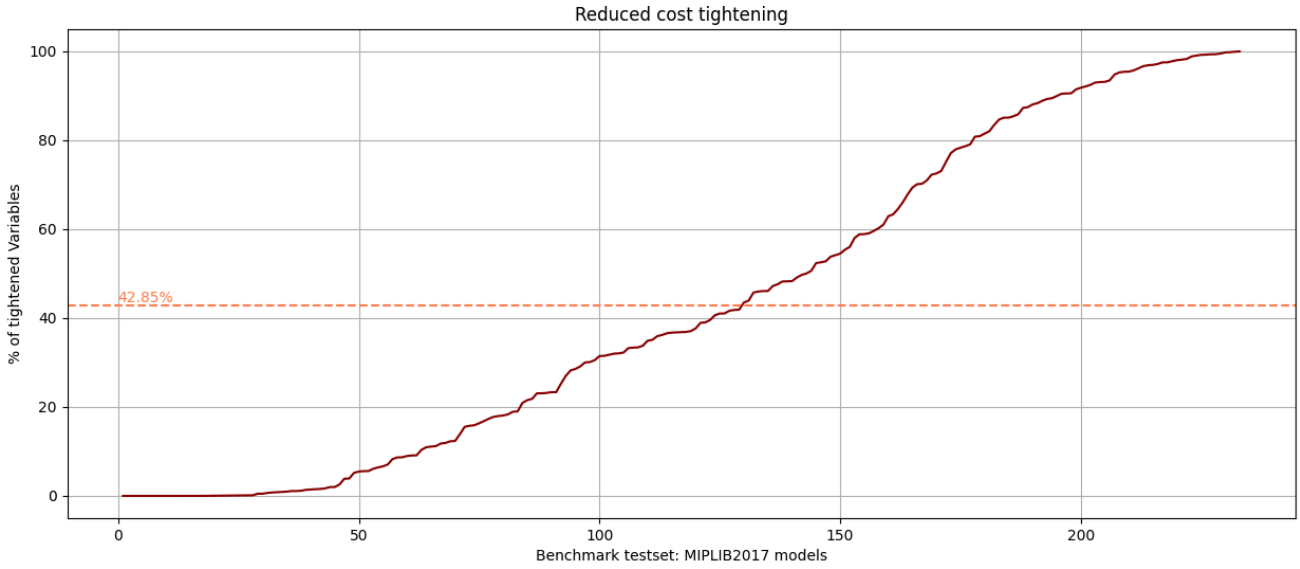
Sui suddetti grafici viene tratteggiata una retta orizzontale all'altezza del valore medio percentuale di variabili *fixed* o *tightened* con relativo valore numerico.

Aggregando i dati per test set si ottiene:

MIPLIB 2017

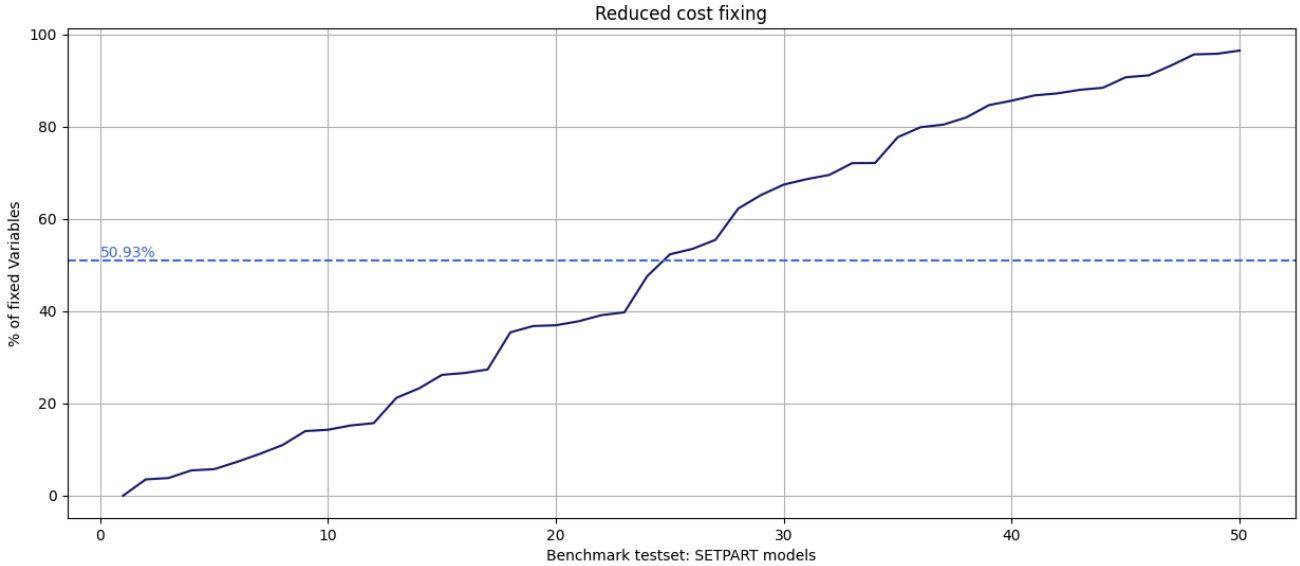


[variabili fissate su MIPLIB 2017]

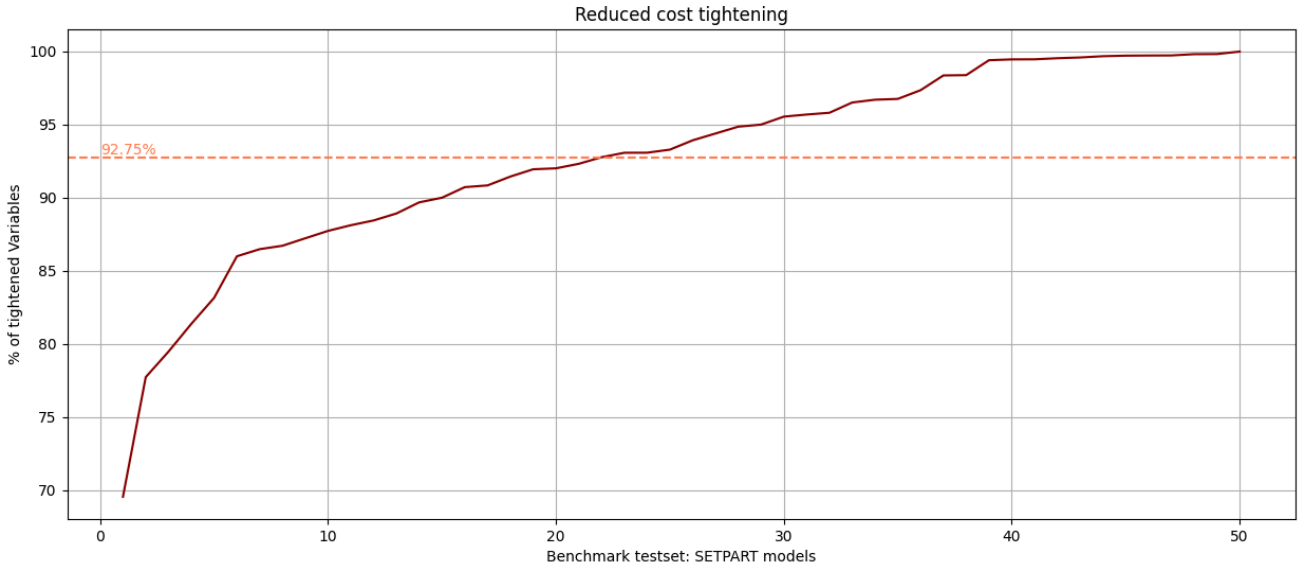


[variabili ristrette su MIPLIB 2017]

setpart



[variabili fissate su setpart]



[variabili ristrette su setpart]

3.5 Secondo esperimento

Come già accennato in precedenza la differenza sostanziale che intercorre tra il primo e il secondo esperimento sta nell'utilizzo di un diverso valore di *gap*.

Con *gap* si intende il valore di differenza che intercorre tra l'incumbent e il costo ottimo del rilassamento nella formula per il calcolo dei bound (2.4)/(2.6).

Nel primo esperimento tale gap era, appunto:

$$\bar{z} - z_R^*$$

nel secondo esperimento, invece, questo valore viene dimezzato, ottenendo un gap *artificiale*:

$$\frac{(\bar{z} - z_R^*)}{2}$$

Al di là di questa differenza i due test sono, da un punto di vista algoritmico, sovrapponibili, se non per il fatto che in questo secondo esperimento si è interessati alla raccolta di dati sul solo reduced cost fixing, senza considerare i casi di tightening.

I due esperimenti, quindi, sono concretamente molto simili, ma in realtà sussiste tra loro una grande differenza dal punto di vista concettuale.

Nel primo caso, come già detto, nonostante l'ambiente ideale in cui viene svolto l'esperimento, ciò permette comunque di calcolare un bound matematicamente corretto, in questo secondo caso, invece, questa proprietà viene meno.

Si nota, infatti, che dimezzando a priori il gap calcolato ciò potrebbe portare all'applicazione di un bound non corretto sulla variabile in esame.

Il secondo esperimento, quindi, ha lo scopo di valutare il miglioramento nelle performance di reduced cost fixing nel caso in cui fosse possibile trovare un gap sensibilmente più basso.

Da ciò deriva che i dati ottenuti hanno senso solo se messi in relazione al primo caso e non come dati a sé stanti sulla capacità di fixing sugli specifici problemi.

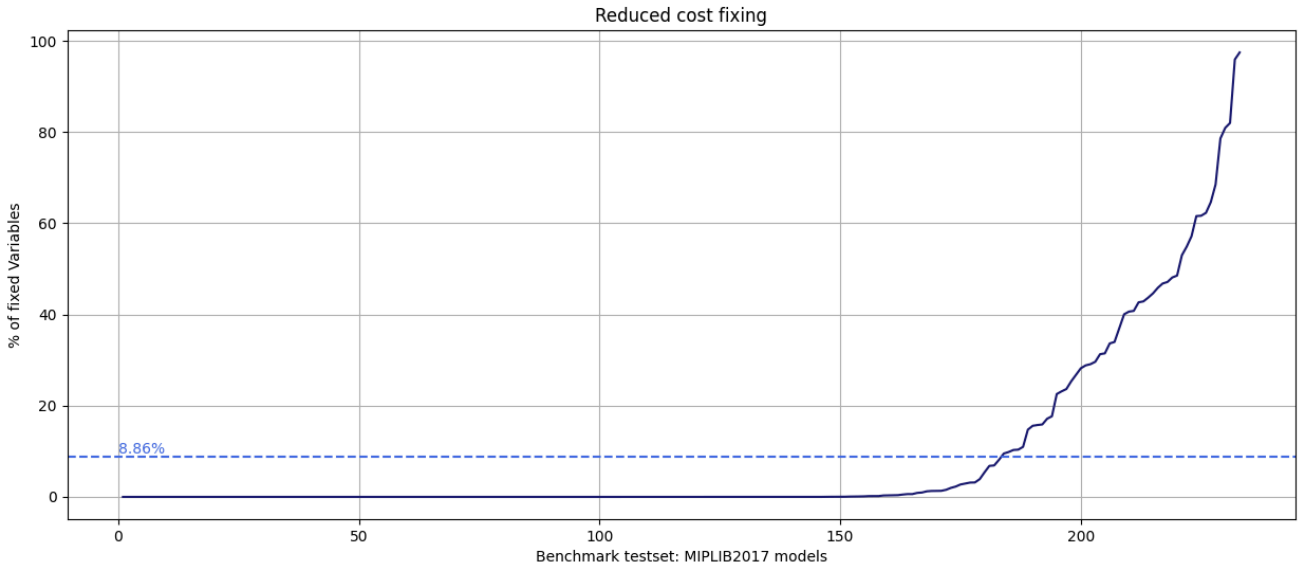
Da un punto di vista implementativo l'unica variazione è nella formulazione della variabile lambda utilizzata per calcolare i bound.

3.5.1 Risultati e grafici sperimentali

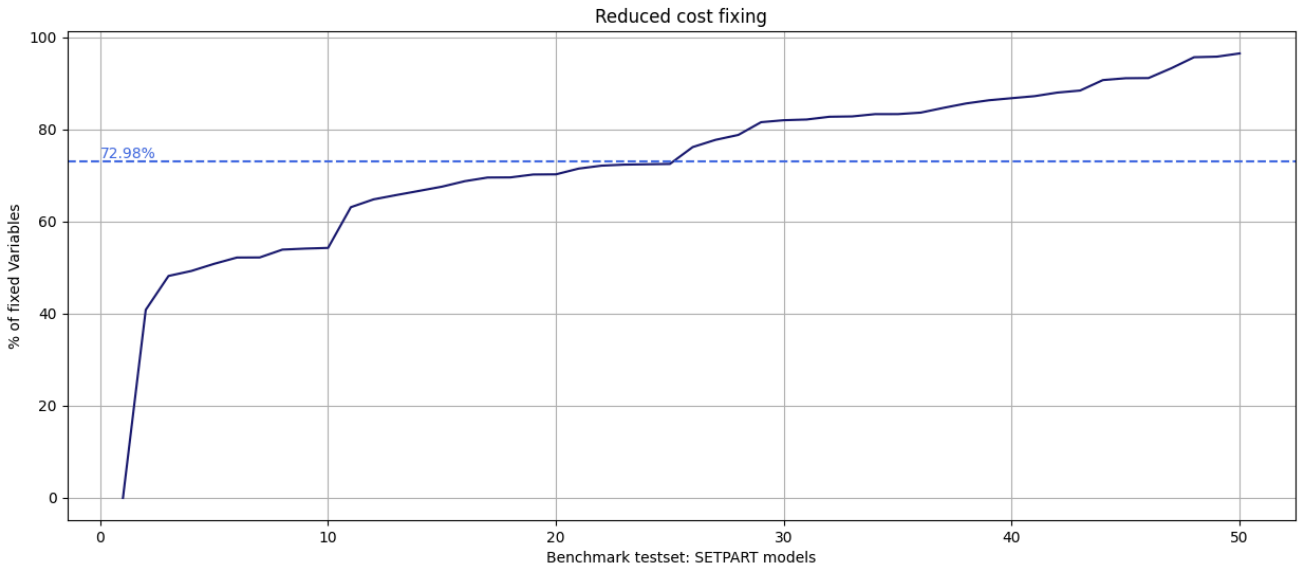
I risultati sperimentali grezzi, che analogamente al primo esperimento non è conveniente riportare qui, sono disponibili, in doppio formato, nella cartella *"Experiment_2"* all'interno della repository [1].

Vengono riportati i grafici, relativi al *Reduced Cost Fixing* applicato ai due test set, corredati, come in precedenza, dal valore di media percentuale.

MIPLIB 2017



setpart



Analisi dati sperimentali

Attraverso i dati, e i grafici riassuntivi, ricavati dalle prove sperimentali è possibile, infine, fare alcune considerazioni sulle performance della strategia di Reduced Cost Fixing and Tightening.

Grazie al primo esperimento è possibile, innanzitutto, notare una sensibile differenza tra la percentuale di variabili che è stato possibile restringere e quelle che è stato possibile fissare.

Sempre tenendo a mente che la media percentuale di variabili *tightened* comprende anche l'insieme delle variabili *fixed,* è comunque chiaro come sia possibile applicare un tightening, mediamente, su circa un 40% di variabili in più.

Ciò era già evidente considerando il fatto che la strategia di fixing richiede un vincolo più stringente per essere applicata.

Un'altra differenza, meno scontata, si riscontra, osservando i grafici, tra le performance dei metodi applicati al primo e al secondo test set.

Si nota infatti un considerevole incremento percentuale delle variabili fissate e ristrette nel test set *setpart.*

Ciò deriva dal fatto che *setpart* raggruppa una serie di problemi a variabili puramente binarie, tali variabili sono, per loro natura, limitate tra i valori 0 e 1, e questo rende più frequente la possibilità di fissare una di esse ad uno dei due valori soglia.

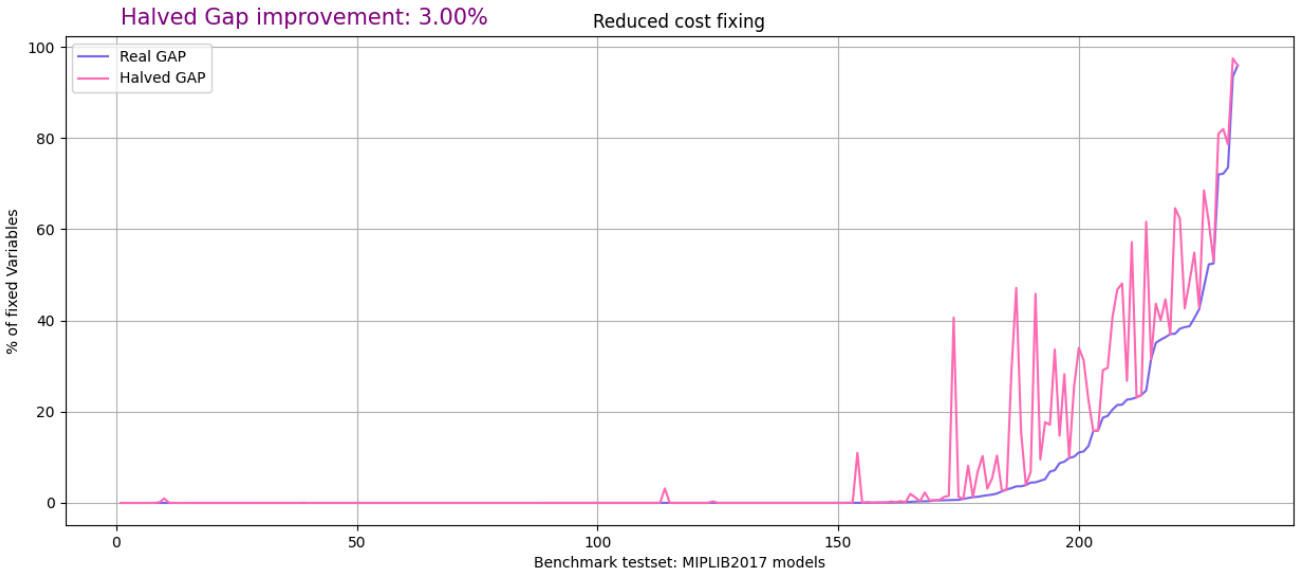
È possibile, quindi, affermare con certezza che le performance di tale algoritmo subiscono un netto miglioramento quando viene applicato a variabili intere, o ancor meglio, binarie, dimostrando una convenienza del suo impiego su problemi di programmazione lineare intera pura.

4.1 Analisi comparativa

Per quanto riguarda il secondo esperimento condotto si possono fare alcune valutazioni sull'effettivo miglioramento, in termini di efficienza, apportato dal gap dimezzato al fixing delle variabili.

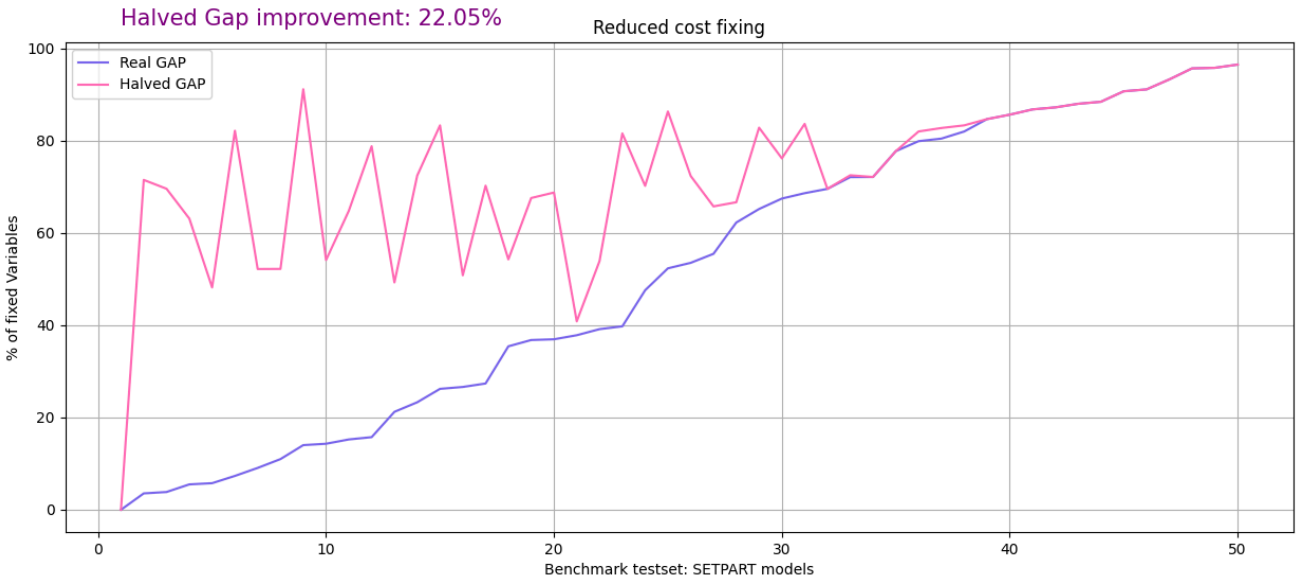
Nello specifico vengono tracciati i grafici comparativi sui due test set:

MIPLIB 2017



[grafico comparativo su MIPLIB 2017]

setpart



[grafico comparativo su setpart]

Come è possibile osservare dai due grafici l'introduzione di un gap dimezzato permette di fissare un buon numero di variabili in più rispetto al caso con gap matematicamente corretto.

Nel caso specifico del test set *setpart* si osserva un aumento medio del 22.05% di variabili fissate.

Nonostante tale esperimento abbia valore puramente teorico rende evidente come la ricerca di parametri strutturali che permettano di calcolare un gap, *matematicamente corretto*, più basso possibile comporti un sostanziale aumento delle performance del metodo.

Conclusioni

È infine possibile trarre alcune conclusioni sul lavoro svolto e sui risultati ottenuti.

Tramite gli esperimenti in esame è stato possibile valutare la capacità del metodo basato sui costi ridotti di fissare e restringere il valore di alcune variabili, ciò, chiaramente, impatta in senso positivo sui tempi di risoluzione di problemi di programmazione lineare intera, limitando la ricerca dei valori da assegnare alle variabili ad un insieme più ridotto.

Tale metodo può trovare diversi impieghi in vari algoritmi di ottimizzazione, sarebbe possibile, ad esempio, applicarlo ad ogni nodo di un algoritmo Branch&Bound in modo da velocizzarne l'esecuzione.

Il primo test quindi ha mostrato come sia possibile fissare e restringere un buon numero di variabili nel caso, non realistico, in cui si sia a conoscenza a priori del valore di costo ottimo del problema.

Se il primo esperimento ha dato un'idea iniziale sulla qualità, in caso ottimo, dell'algoritmo, il secondo permette di trarre delle considerazioni ben più importanti.

Tramite i dati raccolti, infatti, si nota come alcuni sviluppi futuri possano portare ad un sostanziale miglioramento nella tecnica di fixing delle variabili.

Nello specifico, sarebbe possibile concentrare gli sforzi sulla ricerca di una metodologia che permetta di calcolare dei valori di *gap ottimi* che, ottimizzando l'esecuzione dell'algoritmo, comporterebbero un incremento della percentuale di variabili fissate.

Bibliografia

[1] Benetti, Christian. “Esperimenti su reduced cost fixing in Programmazione Lineare Intera.” *GitHub repository*, https://github.com/ChristianBenetti/RCF_RCT_Experiments.

[2] “Cplex Python API Reference Manual.” https://www.ibm.com/docs/en/icos/22.1.1?topic=optimizers-cplex-python-api-reference-manual.

[3] Dantzig, George, et al. “Solution of a large-scale traveling-salesman problem.” *Journal of the operations research society of America*, 1954.

[4] De Giovanni, Luigi. *Note su Programmazione Lineare e Metodo del Simplesso (parte II)*. 2008, https://www.math.unipd.it/~luigi/courses/ricop0809/ro_05.m01.programmazionelineare.02.pdf.

[5] Hooker, John N. “Integrated Methods for Optimization.” *International Series in Operations Research & Management Science*, vol. 170, 2011.

[6] “IBM ILOG CPLEX Optimization Studio 22.1.1.0.” *IBM Documentation*, https://www.ibm.com/docs/en/icos/22.1.1.

[7] Matplotlib Python package. “matplotlib 3.8.0.” *Matplotlib — Visualization with Python*, https://matplotlib.org/.

[8] Microsoft. *Visual Studio Code 1.84*, https://code.visualstudio.com/.

[9] MIPLIB library. *MIPLIB 2017 -- The Mixed Integer Programming Library*, https://miplib.zib.de/index.html. Accessed 6 December 2023.

[10] Numpy Python package. *numpy 1.26.1*, https://numpy.org/.

[11] Python software foundation. “Python 3.9.13.” *Welcome to Python.org*, https://www.python.org/.

[12] Salvagnin, Domenico. *Cenni di programmazione Lineare*. 2020, Università degli studi di Padova.

[13] Salvagnin, Domenico. *Cenni di programmazione Lineare Intera*. 2020, Università degli studi di Padova.

[14] Salvagnin, Domenico. *Introduzione all’ottimizzazione discreta*. 2018, Università degli studi di Padova.

[15] Schurmann, Lukas, and Petra Mutzel. *A Reduced Cost-based Model Strengthening Method*. 2023, https://epubs.siam.org/doi/pdf/10.1137/1.9781611977714.7.