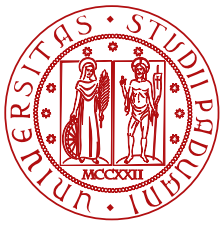

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

Implementazione dell'euristico Alternating Criteria Search per problemi di programmazione lineare intera

Relatore
Prof. Salvagnin Domenico

Laureando
Piai Luca

ANNO ACCADEMICO 2023-2024

Data di laurea 26/09/2024

Sommario

L'alternating Criteria Search è un algorimto euristico che permette di ottenere soluzioni di alta qualità per problemi generici di programmazione lineare intera mista. Ad oggi le implementazioni di ottimizzatori per problemi MIP allo stato dell'arte, sfruttano algoritmi euristici come strumento per migliorare l'efficienza nella ricerca di una soluzione ottima. L'ACS può essere utilizzato in congiunzione con un algoritmo di ricerca ottimale, oppure come risolutore per quella classe di problemi in cui il raggiungimento dell'ottimo risulta essere computazionalmente troppo oneroso.

L'algoritmo presenta molti parametri che devono essere impostati: i valori assegnati e la strategia con la quale questi vengono individuati influenzano le prestazioni dell'algoritmo. L'obiettivo è quello di ricavare, per via sperimentale, la combinazione di valori che fornisce le migliori prestazioni.

Indice

Capitolo 1

Introduzione

In questo capitolo vengono introdotti i concetti base dei problemi di ottimizzazione [1], i problemi di programmazione lineare (LP) [2], i problemi di programmazione lineare intera mista (MIP) e i rispettivi algoritmi risolutivi [3].

1.1 Definire un problema di ottimizzazione

Un problema di ottimizzazione P viene definito come segue:

$$P : \begin{cases} \min(or \max) f(x) \\ S \\ x \in D \end{cases} \tag{1.1}$$

dove x è una un vettore $[x_1, \ldots, x_n]$, D è un prodotto cartesiano $D_1 \times \ldots \times D_n$ tale per cui $x_j \in D_j \forall j = 1, \ldots, n$, $f(x)$ è una funzione a valori reali in x ed infine S è un insieme finito di vincoli.

Un vincolo $c \in S$ può essere una disuguaglianza o un'uguaglianza ed è associato ad un sottoinsieme di variabili x_c. Si parla di vincolo soddisfatto quando il suo valore è vero, in caso contrario si parla di vincolo violato.

Ogni $x \in D$ viene chiamata *soluzione* di P; se x soddisfa i tutti i vincoli del problema allora la soluzione viene detta *ammissibile*. L'insieme delle soluzioni ammissibili di un problema di ottimizzazione P si indica con $F(P)$.

Un problema di ottimizzazione viene detto *impossibile* (*infeasible*) se $F(P) = \varnothing$; *illimitato* (*unbounded*) se $f(x)$ non presenta un limite inferiore per una qualsiasi $x \in F(P)$. Infine si dice che un problema *ammette ottimo finito* se esiste almeno una *soluzione ottima* $x^* \in F(P)$. Una

soluzione $x^* \in F(P)$ è ottima se

$$f(x^*) \leq f(x) \quad \forall x \in F(P). \tag{1.2}$$

Risolvere un problema di ottimizzazione significa trovare una soluzione ottima, dimostrare che sia tale oppure dimostrare che il problema è impossibile o illimitato.

1.2 Concetti generali dell'ottimizzazione

1.2.1 Ricerca

Effettuare una ricerca consiste nel risolvere una sequenza di restrizioni $P_1, \ldots, P_m$ di un problema P. Una restrizione P_k è un problema di ottimizzazione ottenuto da P aggiungendo dei vincoli. Si osserva che:

- se $F(P_k)$ è illimitato allora lo è anche $F(P)$;
- se $F(P_k) = \varnothing$, ovvero se P_k è impossibile, allora non si può dire nulla di $F(P)$;
- se $F(P_k)$ presenta una soluzione ottima allora abbiamo trovato il limite superiore della soluzione ottima di P.

Notiamo come, nell'ultimo caso, abbiamo ricavato un'informazione sulla soluzione ottima di P risolvendo una sua restrizione.

Una ricerca viene detta *esaustiva* quando l'insieme delle restrizioni P_k ricopre l'intero spazio delle soluzioni ammissibili di P, ovvero $\bigcup_i F(P_i) = F(P)$. L'esempio più semplice di algoritmo di ricerca esaustivo è il *generate and test*. L'algoritmo ricava tutte le soluzioni $x \in D$, verifica quali di queste soddisfa i vincoli del problema ed infine sceglie la migliore. In questo caso le restrizioni di P si ottengono fissando x ad un certo valore appartenente al dominio D. Per come lo abbiamo descritto, l'algoritmo è esponenziale. Tuttavia può risultare efficace se utilizzato in casi specifici, per esempio quando le possibili soluzioni del problema non sono in numero elevato e sono computazionalmente poco costose.

Un algoritmo di ricerca esaustiva più avanzato è la *ricerca ad albero*. Lo spazio di ricerca di P viene diviso ricorsivamente fino ad ottenere delle restrizioni (corrispondenti alle foglie dell'albero) sufficientemente facili da risolvere. Il caso peggiore ha prestazioni esponenziali e si verifica quando tutte le foglie corrispondono a restrizioni di P in cui x è fissato ad un valore appartenente al dominio D. In altre parole l'algoritmo diventa identico al generate and test.

L'ultima tipologia di ricerca che introduciamo è quella *euristica*. La ricerca euristica permette di ottenere una soluzione del problema P più velocemente rispetto ad una ricerca esaustiva. Il vantaggio di questo approccio è proprio la velocità mentre lo svantaggio principale è che

in generale gli algoritmi euristici non sono ottimi, ovvero non trovano una soluzione ottima ma ne trovano una ammissibile.

1.2.2 Rilassamento

Un rilassamento di un problema di ottimizzazione P è un altro problema di ottimizzazione R ottenuto da P eliminando alcuni vincoli e sostituendo la funzione obiettivo $f(x)$ con una sua approssimazione inferiore $g(x)$. In altre parole: R è un rilassamento di P se $F(P) \subseteq F(R)$ e se $g(x) \leq f(x) \quad \forall x \in F(P)$. Valgono le seguenti proprietà:

- se $F(R) = \varnothing$ allora $F(P) = \varnothing$;
- se x^* è una soluzione ottima di R, allora $g(x^*)$ è un limite inferiore valido per il valore ottimo di P.
- se x^* è una soluzione ottima di R, $x^* \in F(P)$ e $g(x^*) = f(x^*)$, allora x^* è ottima anche per P.

Se ci troviamo nell'ultimo caso, allora abbiamo risolto P risolvendo una sua restrizione. Se R è più semplice da risolvere si ottiene un vantaggio in termini di costi computazionali.

1.3 Programmazione lineare

Un generico problema di programmazione lineare è definito come segue:

$$P : \begin{cases} \min c^t x \\ a_i x \sim b_i & i = 1, \ldots, m \\ l_j \leq x_j \leq u_j & j = 1, \ldots, n \end{cases} \tag{1.3}$$

dove $\sim \in \{\leq, \geq, =\}$, $l_j \in \mathbb{R} \cup \{-\infty\}$ e $u_j \in \mathbb{R} \cup \{+\infty\}$ e dove $c^t x$ è funzione lineare. Dato un problema di programmazione lineare, esiste un algoritmo che, al caso peggiore, riesce a risolverlo con complessità polinomiale nella taglia del problema, dove la taglia è definita dal numero delle variabili e di vincoli.

In generale la programmazione lineare è poco applicabile in quanto la maggior parte dei problemi reali non rientra a far parte di questa categoria. Sebbene non venga spesso applicata direttamente, risulta essere uno strumento di fondamentale importanza che viene utilizzato da algoritmi che risolvono problemi più complessi, come per esempio i problemi di programmazione lineare intera mista.

1.4 Programmazione lineare intera

Un problema di programmazione lineare intera (MIP, *Mixed Integer linear Programming*) è definito come segue:

$$P: \begin{cases} \min c^t x & \\ a_i x \sim b_i & i = 1, \ldots, m \\ l_j \leq x_j \leq u_j & j = 1, \ldots, n = N \\ x_j \in \mathbb{Z} & \forall j \in J \subseteq N = \{1, \ldots, n\} \end{cases} \tag{1.4}$$

dove $\sim \in \{\leq, \geq, =\}$. Rispetto alla 1.3 viene aggiunto il vincolo di interezza: una o più variabili devono assumere valori interi.

Si riescono a modellare molti più problemi con la programmazione lineare intera rispetto alla programmazione lineare.

1.4.1 Algoritmo Branch and Bound

Il branch-and-bound (B&B) è un algoritmo che si basa sul paradigma divide-and-conquer e viene utilizzato per risolvere problemi di ottimizzazione generica tra cui i problemi di programmazione lineare intera. Il B&B implementa una ricerca ad albero che permette di ricavare una soluzione ottima del problema considerato.

L'algoritmo inizia dal nodo radice il quale rappresenta il problema MIP che vogliamo risolvere. Indichiamo questo problema con P.

1. Si considera il rilassamento R di P ottenuto rimuovendo il vincolo di interezza alle variabili. R è un problema di programmazione lineare ed è quindi più semplice da risolvere. Una volta risolto il rilassamento, si possono presentare tre casi:

   (a) Se $F(R) = \varnothing$ allora $F(P) = \varnothing$ quindi l'algoritmo termina.

   (b) Se esiste una soluzione ottima $x^* \in F(R)$ tale per cui $x^* \in F(P)$, allora x^* è ottima per P e l'algoritmo termina.

   (c) Se invece $x^* \notin F(P)$ significa che la soluzione trovata non rispetta almeno un vincolo di interezza. In questo caso abbiamo trovato un *limite inferiore* sull'ottimo di P.

   Se ci ritroviamo nel caso (c) allora l'algoritmo prosegue.

2. Si effettua *branching* sul nodo radice. L'operazione di branching applicata ad un nodo permette di individuare i suoi nodi figli: questi sono sottoproblemi che vengono ottenuti mediante restrizioni del problema genitore. Almeno una delle variabili della soluzione x^*

non rispetta il vincolo di interezza, ovvero $\exists\, j \in J : x_j^* \notin \mathbb{Z}$. Di conseguenza due nodi figli possono essere creati imponendo a ciascuno uno dei due seguenti vincoli:

$$x_j \leq \lfloor x_j^* \rfloor, \quad x_j \geq \lceil x_j^* \rceil. \tag{1.5}$$

I nodi vengono aggiunti alla coda Q dei sottoproblemi da risolvere. La Figura 1.1 mostra il risultato di un'operazione di branching dal punto di vista della regione ammissibile del problema P.

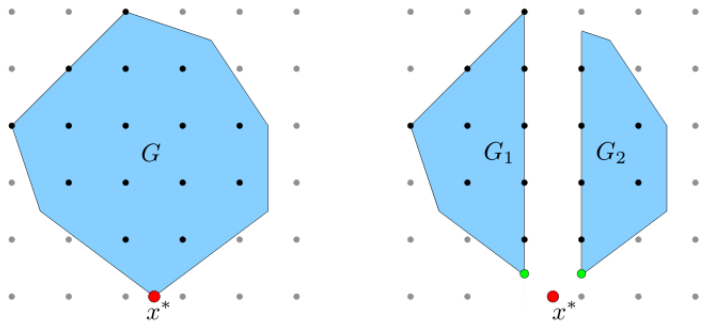

Figura 1.1: Operazione di branching dal punto di vista della regione ammissibile.

3. Si procede considerando un sottoproblema $P_i \in Q$ e si risolve un suo rilassamento R_i. Possono verificarsi quattro casi:

   (a) Se $F(R_i) = \varnothing$ allora il nodo può essere scartato (*prouning by infeasibility*).

   (b) Se esiste una soluzione ottima $\bar{x} \in F(R_i)$ tale che $\bar{x} \in F(P_i)$ allora abbiamo trovato un limite superiore $\bar{z} = c^t \bar{x}$ valido per la soluzione ottima di P. $\bar{z}$ viene chiamato *incumbent*: si tratta della migliore soluzione attuale di P.

   (c) Se $\bar{x} \notin F(P_i)$ si confronta l'incumbent con la soluzione trovata: se $c^t \bar{x} \geq \bar{z}$ allora il nodo può essere scartato (*prouning by optimality*) in quanto $c^t \bar{x}$ è il limite inferiore sull'ottimo di P_i, pertanto non può portare ad una soluzione migliore dell'attuale incumbent.

   (d) Altrimenti ($c^t \bar{x} < \bar{z}$) è necessario fare branching sul nodo ed aggiungere i suoi figli a Q.

4. Il più piccolo valore tornato dai rilassamenti dei nodi aperti è un limite inferiore valido per la soluzione ottima di P. Sfruttando questa proprietà viene aggiornato il valore del limite inferiore trovato nel primo punto.

5. L'algoritmo procede finché la coda dei sottoproblemi Q si svuota. In questa situazione il valore dell'incumbent coincide con quello del limite inferiore e pertanto $\overline{z}$ corrisponde alla soluzione ottima di P trovata dall'algoritmo.

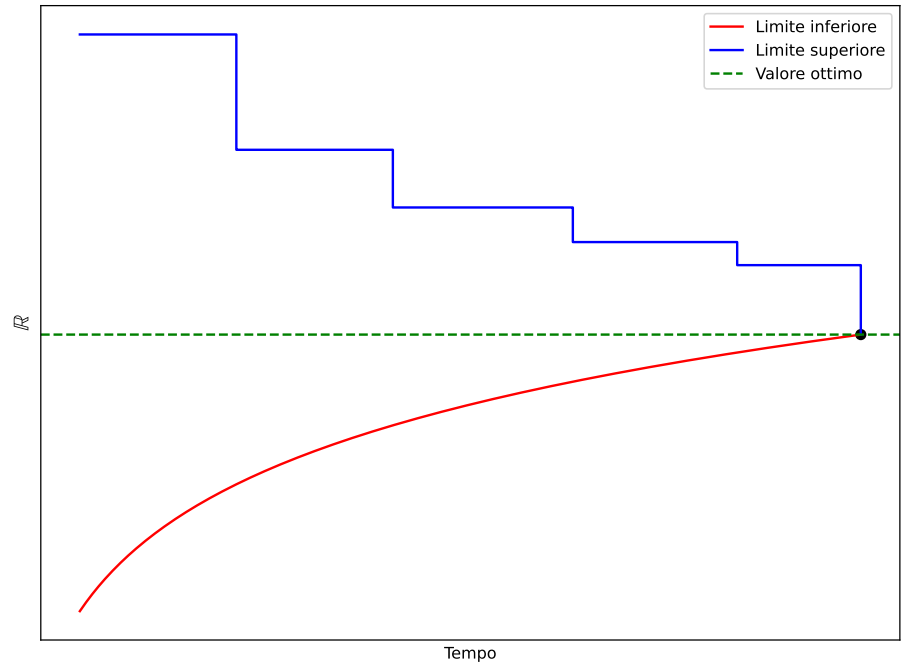

Figura 1.2: Andamento nel tempo del limite inferiore e superiore sul valore ottimo nel B&B.

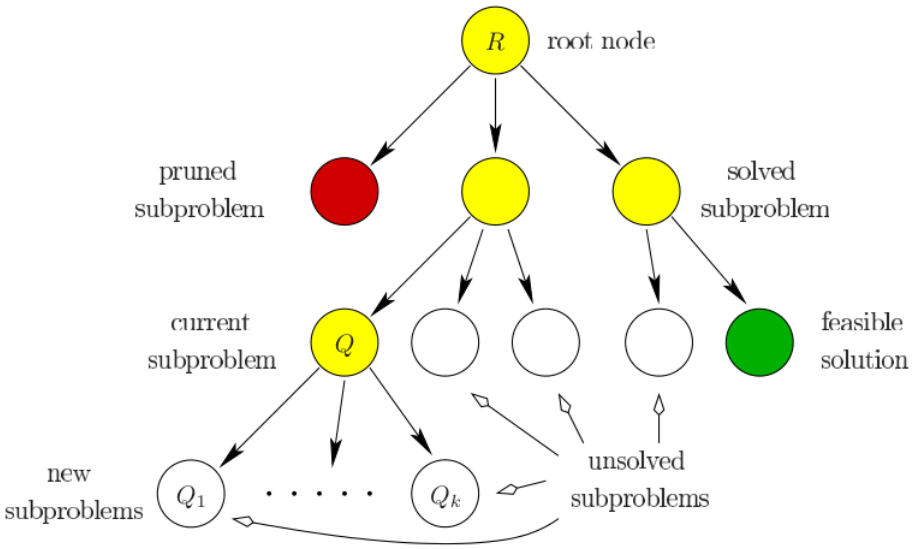

Figura 1.3: Ricerca ad albero di un algoritmo B&B.

1.4.2 Algoritmo Branch and Cut

L'algoritmo B&C è stato sviluppato appositamente per risolvere i problemi di programmazione lineare intera mista. Rispetto al B&B, nel B&C vengono ricavati dei piani di taglio per ogni rilassamento lineare di ogni sottoproblema dell'albero: si parla di rafforzamento del rilassamento lineare associato. L'applicazione dei questa tecnica aumenta la possibilità di ottenere una soluzione intera dal rilassamento lineare, allo stesso tempo permette di ottenere un migliore limite inferiore alla soluzione ottima del problema che stiamo risolvendo.

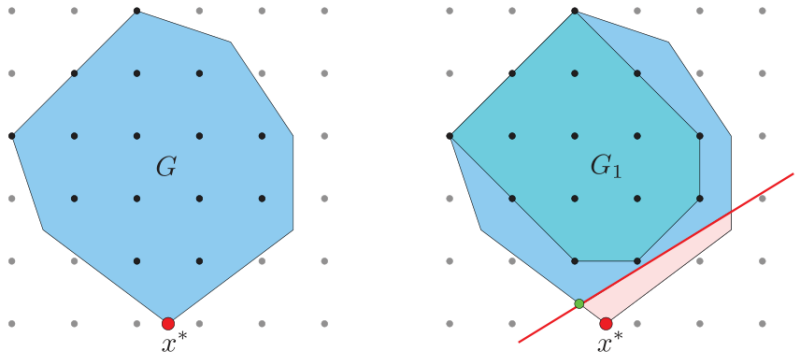

Figura 1.4: Applicazione di un piano di taglio dal punto di vista della regione ammissibile.

1.5 Alternating Criteria Search

L'ACS è un algoritmo euristico che opera su problemi MIP. In generale gli euristici sono metodi che cercano di trovare una buona soluzione impiegando poco tempo ed effettuando scelte basate su alcune caratteristiche rilevanti del problema considerato. Nel nostro contesto un algoritmo euristico applicato ad un problema MIP, permette di ottenere una soluzione che rispetta tutti i vincoli del problema ma non consente di determinare se si tratta della migliore possibile [4] [5]. L'obiettivo in questo caso non è quello di trovare una soluzione ottima ma di ottenerne una ammissibile di alta qualità.

L'approccio utilizzato dall'ACS è quello di risolvere, in maniera iterativa, molteplici *sub-MIP* ricavati a partire dal problema principale. I sub-MIP considerati appartengono ad una delle seguenti due tipologie:

- *FMIP*: questi problemi puntano a minimizzare il grado dell'inammissibilità della soluzione.
- *OMIP*: questi problemi condividono la stessa funzione obiettivo del MIP di partenza e puntano ad ottimizzare il valore della soluzione.

Allo scopo di raggiungere questi due obiettivi vengono risolte delle varianti in cui un certo numero di variabili vengono fissate.

L'ACS trova la sua utilità sia come algoritmo standalone per risolvere problemi molto difficili che richiedono elevati costi computazionali per raggiungere l'ottimalità, sia come strumento aggiuntivo da integrare ad un algoritmo esatto per velocizzare il processo di risoluzione. Infatti l'implementazione allo stato dell'arte di B&C [6] sfrutta degli euristici per ottenere soluzioni utili per la dimostrazione di ottimalità. Questi metodi permettono in molti casi di ricavare soluzioni di alta qualità più velocemente rispetto ad utilizzare solamente il branching.

Capitolo 2

L'algoritmo ACS

In capitolo capitolo viene descritto il funzionamento dell'algoritmo Alternating Criteria Search [7]: viene definita la struttura dei sub-MIP e vengono descritti i passi che permettono all'algoritmo di ottenere una soluzione ammissibile per un problema MIP generico.

2.1 Struttura dei problemi ausiliari

Prendiamo in considerazione il seguente MIP:

$$\begin{cases} \min c^t x \\ Ax = b \\ l \leq x \leq u \\ x_i \in \mathbb{Z} \qquad \forall i \in I \end{cases} \tag{2.1}$$

dove $c \in \mathbb{R}^m$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $I \subseteq \{1, \ldots, n\}$ è l'insieme contenente gli indici delle variabili intere, x è il vettore di variabili $[x_1, \ldots, x_n]$, infine $l \in \overline{\mathbb{R}}^n$ e $u \in \overline{\mathbb{R}}^n$ sono i limiti imposti alle variabili e $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$.

2.1.1 FMIP

Il modello di FMIP è il seguente:

$$\begin{cases} \min \sum_{i=0}^{m-1} \Delta_i^+ + \Delta_i^- \\ Ax + I_m\Delta^+ - I_m\Delta^- = b \\ x_i = \hat{x}_i & \forall i \in F \\ l \leq x \leq u \\ x_i \in \mathbb{Z} & \forall i \in I \\ \Delta^+ \leq 0, \quad \Delta^- \leq 0 \end{cases} \tag{2.2}$$

dove $I_m \in \mathbb{R}^{m \times m}$ è la matrice identità e Δ^+, Δ^- sono due vettori di variabili continue aventi dimensione m in accordo con il numero di vincoli del problema. FMIP è ottenuto da MIP andando a modificare la funzione obiettivo e aggiungendo le variabili Δ_i^+, Δ_i^- chiamate *slack*. Dalla 2.2 notiamo che una soluzione ammissibile per FMIP è ammissibile anche per il MIP se e solo se le variabili slack sommano a zero. FMIP non viene risolto direttamente ma vengono fissate un sottoinsieme $F \subseteq I$ delle variabili intere a dei valori presi dal vettore di input $[\hat{x}, \Delta^+, \Delta^-]$. La presenza delle variabili di slack permette di utilizzare un vettore $\hat{x}$ che non è necessariamente una soluzione ammissibile, l'unica restrizione è che siano rispettati i vincoli di interezza e i limiti delle variabili. In altre parole si deve avere che $l \leq \hat{x} \leq u$ e $\hat{x}_i \in \mathbb{Z}, \forall i \in I$.

Minimizzare la somma delle slack permette di migliorare il grado di ammissibilità della soluzione mentre il fissaggio delle variabili consente di ridurre la complessità del problema.

2.1.2 OMIP

Il secondo problema ausiliario ha come obiettivo quello di migliorare il vettore parzialmente ammissibile $[\hat{x}, \Delta^+, \Delta^-]$. Il suo modello si ottiene partendo dal MIP e aggiungendo ad ogni vincolo le variabili slack:

$$\begin{cases} \min c^t x \\ Ax + I_m\Delta^+ - I_m\Delta^- = b \\ \sum_{i=0}^{m-1} \Delta_i^+ + \Delta_i^- \leq \sum_{i=0}^{m-1} \hat{\Delta}_i^+ + \hat{\Delta}_i^- \\ x_i = \hat{x}_i & \forall i \in F \\ l \leq x \leq u \\ x_i \in \mathbb{Z} & \forall i \in I \end{cases} \tag{2.3}$$

In maniera analoga a FMIP, anche in OMIP vengono fissate alcune delle variabili intere per ridurre la complessità del problema. L'insieme F non deve essere necessariamente lo stesso di 2.2. Infine, per assicurare che la soluzione ottima di OMIP rimanga, al caso peggiore, tanto inammissibile quanto il vettore di input $\hat{x}$, viene aggiunto il vincolo $\sum_i \Delta_i^+ + \Delta_i^- \leq \sum_i \hat{\Delta}_i^+ + \hat{\Delta}_i^-$ che limita il valore della somma delle variabili slack.

2.2 Il processo risolutivo

L'ACS inizia generando una soluzione di partenza intera $\hat{x}$, successivamente:

1. Vengono fissate alcune variabili intere a $x_i = \hat{x}_i$ in FMIP.
2. FMIP viene risolto, la sua soluzione è $[\hat{x},\ \Delta^+,\ \Delta^-]$.
3. Vengono fissate alcune variabili intere a $x_i = \hat{x}_i$ in OMIP.
4. OMIP viene risolto, la sua soluzione è $[\hat{x},\ \Delta^+,\ \Delta^-]$.

Questi quattro passi vengono ripetuti iterativamente finché l'algoritmo non converge ad una soluzione ammissibile, ovvero quando tutte le variabili slack hanno valore nullo.

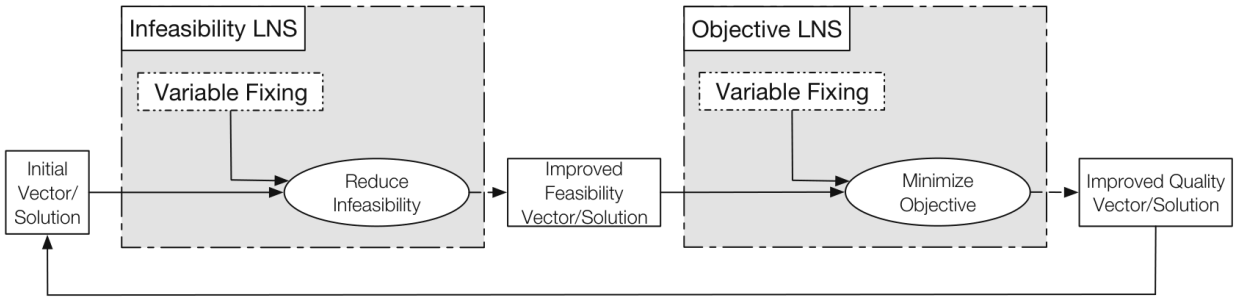

Figura 2.1: Schema a blocchi del processo risolutivo.

Per come sono costruiti i sub-MIP l'inammissibilità della soluzione decresce in maniera monotona, tuttavia non è garantita la convergenza ad una soluzione e per questo motivo è necessario fissare dei limiti che consentano all'algoritmo di terminare. Questo ed altri parametri che verranno introdotti successivamente hanno un'influenza diretta sulle prestazioni dell'algoritmo. La Figura 2.2 mostra il comportamento ideale dell'algoritmo.

Una volta trovata una soluzione ammissibile, inizia una seconda fase in cui viene risolto OMIP allo scopo di migliorare la qualità della soluzione. Conclusa questa fase l'algoritmo termina restituendo la soluzione trovata.

Algoritmo 1: Alternating Criteria Search

Ensure: Feasible solution $\hat{x}$ if found
  initialize $[\hat{x}, \Delta^+, \Delta^-]$ ▷ Soluzione intera che rispetta i limiti
  $slack\_sum \leftarrow -1$
  while time limit not reached **and** $slack\_sum \neq 0$ **do**
    $F \leftarrow$ randomized variable index subset, $F \subseteq I$
    $[\hat{x}, \Delta^+, \Delta^-] \leftarrow$ solve_FMIP$(F, \hat{x})$
    $slack\_sum \leftarrow \sum_i \Delta_i^+ + \Delta_i^-$
    $F \leftarrow$ randomized variable index subset, $F \subseteq I$
    $[\hat{x}, \Delta^+, \Delta^-] \leftarrow$ solve_OMIP$(F, \hat{x}, sum_slack)$
    $slack\_sum \leftarrow \sum_i \Delta_i^+ + \Delta_i^-$
  end while
  if $slack\_sum = 0$ **then** ▷ Miglioramento qualità della soluzione
    while time limit not reached **do**
      $F \leftarrow$ randomized variable index subset, $F \subseteq I$
      $[\hat{x}, \Delta^+, \Delta^-] \leftarrow$ solve_OMIP$(F, \hat{x}, 0)$
    end while
  end if
  return $[\hat{x}, \Delta^+, \Delta^-]$

  function SOLVE_FMIP$(F, \hat{x})$
    return $\min\{\sum_i \Delta_i^+ + \Delta_i^- | Ax + I_m\Delta^+ - I_m\Delta^- = b, x_j = \hat{x}_j \forall j \in F, x_j \in \mathbb{Z} \forall j \in I\}$
  end function

  function SOLVE_OMIP$(F, \hat{x}, \hat{\Delta})$
    return $\min\{c^t x | Ax + I_m\Delta^+ - I_m\Delta^- = b, \sum_i \Delta_i^+ + \Delta_i^- \leq \hat{\Delta}, x_j = \hat{x}_j \forall j \in F, x_j \in \mathbb{Z} \forall j \in I\}$
  end function

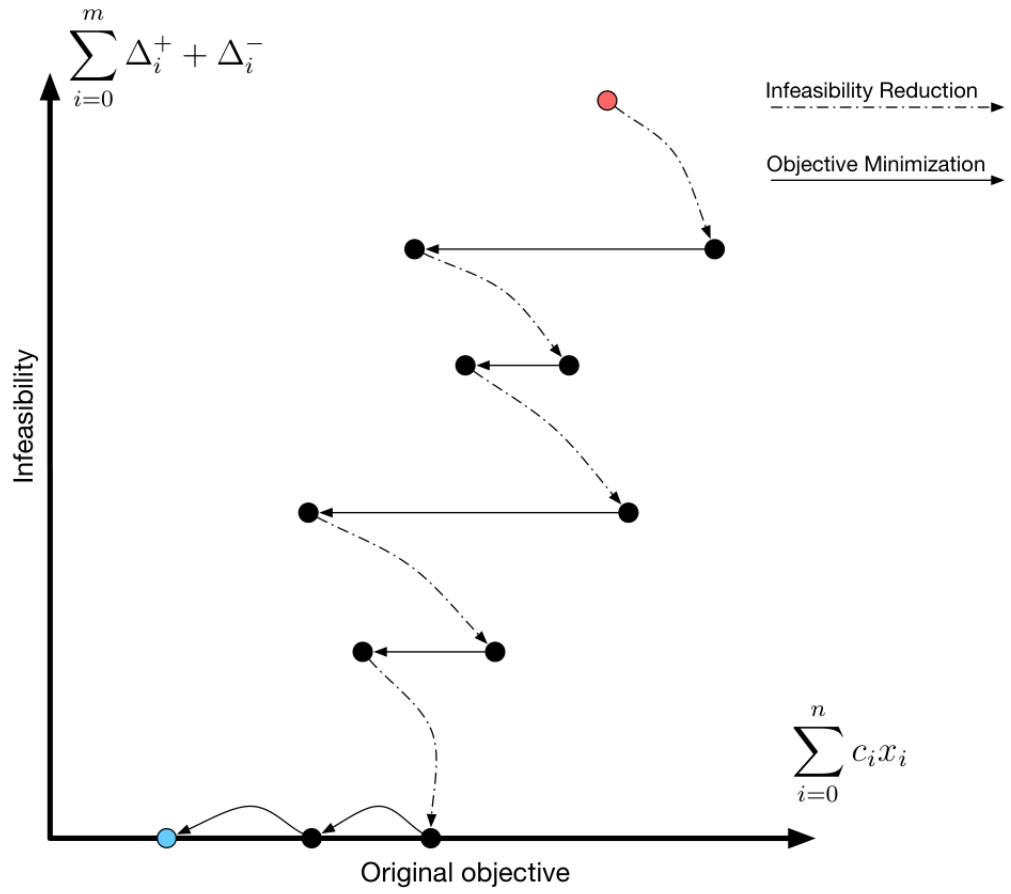

Figura 2.2: Transizione da una soluzione non ammissibile (punto rosso) ad una ammissibile qualitativamente buona (punto azzurro).

2.2.1 Trovare il vettore di soluzioni iniziale

Alternating Criteria Search richiede solamente che il vettore iniziale rispetti i vincoli di interezza delle variabili e i loro limiti. Può essere utilizzato un qualsiasi vettore che rispetti queste due richieste. Esistono diverse strategie che si possono adottare per ottenere il vettore iniziale: la più semplice e la meno costosa prevede di generare i valori in maniera randomica, tuttavia è molto probabile che il vettore generato in questo modo abbia un alto grado di non ammissibilità. Un approccio comune è quello di risolvere il rilassamento lineare di MIP e poi arrotondare i valori frazionari all'intero più vicino. Il problema in questo caso è che il rilassamento potrebbe essere molto costoso da risolvere. L'Algoritmo 2 propone un approccio ibrido: si tratta di un algoritmo euristico che combina la generazione randomica dei valori alla risoluzione di un problema lineare. Ad ogni iterazione un certo numero di variabili intere vengono fissate a dei valori casuali che rispettano i limiti, quelle rimanenti vengono ottimizzate verso l'ammissibilità. L'algoritmo termina quando tutte le variabili sono state fissate.

Algoritmo 2: Vettore iniziale euristico

Require: Percentage of variables to fix θ, $0 < \theta \leq 100$, Fixed bound constat c_b
Ensure: Starting integer-feasible solution vector $\hat{x}$
  $F \leftarrow \varnothing$
  while $\hat{x}$ is not integer feasible **and** $F \neq I$ **do**
    $K \leftarrow$ index set of the first $\theta\%$ of unfixed variables of I
    for $k \in K$ **do**
      $\hat{x}_k \leftarrow$ random integer value between $[\max(l_k, -c_b), \min(u_k, c_b)]$
    end for
    $F \leftarrow F \cup K$
    $[x, \Delta^+, \Delta^-] \leftarrow \min\{\sum_i \Delta_i^+ + \Delta_i^- | Ax + I_m\Delta^+ - I_m\Delta^- = b, x_j = \hat{x}_j \forall j \in F\}$
    $Q \leftarrow$ index set of integer variables of x with integer value
    $\hat{x}_q \leftarrow x_q, \forall q \in Q$
    $F \leftarrow F \cup Q$
  end while
  return $\hat{x}$

L'algoritmo riceve in input θ e c_b: θ è la percentuale di variabili che vengono fissate ad un valore casuale ad ogni iterazione; c_b viene utilizzato per gestire i casi in cui i limiti delle variabili tendono a $\pm\infty$. Una volta fissate in maniera randomica le prime $\theta\%$ variabili di I non ancora fissate, viene risolto il rilassamento lineare di FMIP. Se esistono delle variabili che non sono ancora state fissate e che hanno un valore intero nella soluzione del rilassamento, allora queste vengono fissate. Ne risulta che ad ogni iterazione almeno $\theta\%$ di variabili vengono fissate, di conseguenza l'algoritmo effettua $\lceil \frac{100}{\theta} \rceil$ iterazioni nel caso peggiore. Il parametro θ rappresenta un tradeoff tra la difficoltà del rilassamento lineare e l'ammissibilità del vettore di soluzioni iniziale.

2.2.2 Il fissaggio delle variabili

Una volta ottenuto un vettore di soluzioni, prima di passare a risolvere un sub-MIP, bisogna decidere quante e quali variabili intere fissare. Il numero di variabili da fissare influisce sulla complessità del sub-MIP: fissarne poche potrebbe non semplificare a sufficienza il problema ausiliario e dunque ottenere dei miglioramenti richiederebbe molto tempo; d'altro canto, fissarne troppe potrebbe restringere eccessivamente la ricerca e portare a dei minimi miglioramenti.

Trovare un metodo intelligente per scegliere quali variabili fissare può essere molto difficile visto che esistono molte strutture di problemi da considerare. L'Algoritmo 3 implementa un'idea molto semplice che si adatta a qualsiasi tipo di struttura e permette di scegliere la quantità di variabili da fissare. Viene scelta casualmente un variabile intera x e successivamente vengono fissate un certo numero di variabili consecutive a x. Il parametro ρ fornito in input indica la

quantità di variabili da fissare. Se viene raggiunta la fine di I e non sono state fissate abbastanza variabili, allora le rimanenti vengono fisste partendo dall'inizio di I in modo circolare.

Algoritmo 3: Selezione delle variabili da fissare

Require: Fraction of variables to fix ρ, $0 < \rho < 1$
Ensure: Set of integer indices F
  function RANDOM_FIXINGS(ρ)
    $i \leftarrow$ random element in I
    $F \leftarrow$ first $\rho \cdot |I|$ consecutive integer variable indices starting from i in a circular fashion
    return F
  end function

Solitamente la formulazione di un problema MIP è strutturata in modo tale che variabili dello stesso tipo e collegate a livello logico vengano disposte consecutivamente. Considerare insiemi composti da variabili consecutive spesso permette di individuare sottostrutture coese all'interno di un problema. Fissare variabili appartenenti alla stessa sottostruttura e lasciare libere le altre può produrre sub-MIP più semplici da risolvere.

Capitolo 3

Implementazione e risultati degli esperimenti

In questo capitolo vengono illustrate le scelte implementative adottate, il metodo impiegato per eseguire i test e i risultati ottenuti per le varie configurazioni di parametri.

3.1 Setup sperimentale e ambiente di test

3.1.1 Insieme delle istanze considerate

Le performance dell'algoritmo sono state ricavate andandolo a testare sull'insieme di problemi fornito dalla libreria MIPLIB 2017 [8]. In particolare sono stati utilizzati i problemi appartenenti al *Benchmark Set* nel quale si trovano:

- 220 istanze facili;
- 19 istanze difficili;
- una istanza "aperta".

La difficoltà di un'istanza è definita da MIPLIB in base al tempo necessario per raggiungere l'ottimalità o per dimostrarne l'impossibilità. Le istanze classificate come "aperte" non sono ancora state risolte.

3.1.2 Software e hardware utilizzato

L'algoritmo è stato realizzato utilizzando il linguaggio di programmazione C. Per la manipolazione e la risoluzione di problemi MIP e LP sono state utilizzate le funzioni offerte dalla *CPLEX*

Callable Library 22.1.1 [9]. I test sono stati effettuati sul cluster del Dipartimento di Ingegneria dell'Informazione. La macchina utilizzata dispone di un processore Intel(R) Xeon(R) CPU E3-1220 V2 @ 3.10GHz e 16GB di RAM.

3.1.3 Statistiche osservate

Le statistiche ricavate da ogni test sono le seguenti:

- Numero di problemi per cui è stata trovata una soluzione ammissibile per MIP[1].
- Numero di iterazioni medie per ottenere una soluzione ammissibile per MIP.
- Tempo medio per ottenere una soluzione ammissibile per MIP.
- Qualità media della soluzione trovata.

Per il calcolo della qualità è stata utilizzata la formula introdotta in [7]. Data una soluzione x per un problema MIP avente soluzione ottima $\hat{x}$, la qualità $\gamma(x) \in [0, 1]$ di x è definita come segue:

$$\gamma(x) = \begin{cases} 0 & \text{se } |c^t\hat{x}| = |c^t x| = 0 \\ 1 & \text{se } c^t\hat{x} \cdot c^t x < 0 \\ \frac{|c^t\hat{x} - c^t x|}{\max\{|c^t\hat{x}|, |c^t x|\}} & \text{altrimenti.} \end{cases} \tag{3.1}$$

3.2 Dettagli implementativi

L'intera implementazione è disponibile online [10].

3.2.1 I parametri

Di seguito viene riportata una lista contenente tutti i parametri da fissare prima di lanciare l'algoritmo, per ognuno viene specificato il nome e una breve descrizione. Da notare che alcuni parametri sono stati fissati ad un valore costante e non sono più stati modificati, il questo caso viene riportato il valore assegnato.

- `MAX_ITR`: massimo numero di iterazioni dell'algoritmo. Una iterazione è completa quando viene risolto FMIP e successivamente OMIP.
- `dettime_lim`: limite di tempo espresso in *ticks* [11]. Si tratta del tempo massimo che ha a disposizione l'ottimizzatore di CPLEX per risolvere un problema.

[1] Nella sezione dedicata ai risultati questo valore è indicato da "# soluzioni".

- `BOUND_CONSTANT`: fa riferimento al parametro c_b introdotto nell'Algoritmo 2. Il suo valore è stato fissato a 10^6.

- θ: percentuale di valori generati randomicamente ad ogni iterazione dell'algoritmo che genera il vettore iniziale (vedi Algoritmo 2). Il suo valore è stato fissato a 1%.

- ρ: percentuale di variabili intere fissate nei sub-MIP (vedi Algoritmo 3).

- `seed`: seme utilizzato per la generazione di numeri pseudo-casuali.

Per la generazione di numeri casuali è stata utilizzata la funzione `rand()` di C. Per garantire la riproducibilità dei risultati e per facilitare il processo di debugging, sono state utilizzate sequenze di numeri pseudo-casuali ottenute tramite l'ausilio della funzione `srand()`, passando come argomento il parametro `seed`. Sono stati utilizzati i seguenti semi $\{19410524, 7010598105, 715471\}$. Il valore di θ è stato fissato a 1% in quanto si è notato che per la maggior parte dei 240 problemi considerati, il rilassamento lineare di FMIP non richiede troppo tempo per essere risolto. Nel caso in cui un rilassamento risulta essere troppo complesso e viene sforato il limite di tempo imposto da `dettime_lim`, θ viene incrementato del 50%. Infine `dettime_lim` è stato calcolato utilizzando la formula

$$\texttt{dettime\_lim} = \mu(x,y) = \max(x, \min(nz/y, 10^5)) \tag{3.2}$$

dove μ è una funzione, mentre: nz, il numero di valori diversi da zero all'interno della matrice dei vincoli del problema MIP che vogliamo risolvere; y un qualsiasi valore reale; x è il minimo valore di tempo che vogliamo assegnare al parametro. Il tempo massimo assegnabile è stato fissato a 10^5. Utilizzare questa formula permette di assegnare in maniera dinamica il tempo limite entro il quale l'ottimizzatore CPLEX può trovare una soluzione di un sub-MIP.

3.2.2 Pseudocodice

Di seguito viene riportato lo pseudocodice dell'algoritmo. Si tratta di una versione più dettagliata dell'Algoritmo 1 in cui sono presenti anche le chiamate alle funzioni che implementano l'Algoritmo 2 e l'Algoritmo 3.

```
AlternatingCriteriaSearch(mip, MAX_ITR, dettime_lim, θ, ρ, BOUND_CONSTANT) {
    fmip = create_fmip(mip);
    omip = create_omip(mip);
    x̂ = generate_starting_vector(θ, BOUND_CONSTANT);

    sum_slack = -1
    for (itr = 0; itr < MAX_ITR && sum_slack != 0; itr++) {
        variable_fixing(fmip, ρ);            // Fissa le variabili.
        [x̂, Δ+, Δ−] = solve_fmip(fmip, dettime_lim);
        restore_bounds(fmip, mip);           // Rimuove le variabili fissate.
        sum_slack = Σ_i Δ_i^+ + Δ_i^−;

        variable_fixing(omip, ρ);
        [x̂, Δ+, Δ−] = solve_omip(omip, sum_slack, dettime_lim);
        restore_bounds(omip, mip);
        sum_slack = Σ_i Δ_i^+ + Δ_i^−;
    }

    if (sum_slack == 0) {                    // Miglioramento qualita'.
        best_solution = 0;
        do {
            best_solution = c^t x̂;
            variable_fixing(omip, ρ);
            [x̂, Δ+, Δ−] = solve_omip(omip, 0, dettime_lim);
            restore_bounds(omip, mip);
        } while (c^t x̂ < best_solution);
        x̂ = best_solution;
    }

    return [x̂, Δ+, Δ−];
}
```

Le funzioni `solve_fmip` e `solve_omip` ricevono come argomento il tempo limite entro il quale deve essere trovata una soluzione per il sub-MIP. Il risolutore riesce sempre a trovare una soluzione ammissibile sia per FMIP che per OMIP. Questo comportamento è stato ottenuto sfruttando il meccanismo dei *MIP starts* offerti dalla libreria CPLEX [12]. Prima di risolvere un problema, è possibile passare un MIP start, ovvero una soluzione, che verrà utilizzata dall'ottimizzatore come punto di partenza nel processo di risoluzione. Siccome una soluzione ammissibile per FMIP lo è anche per OMIP, nel nostro caso risulta vantaggioso passare come

MIP start a OMIP la soluzione trovata da FMIP e, successivamente, passare la soluzione di OMIP prima di risolvere FMIP. In questo modo, oltre a garantire la presenza di una soluzione ammissibile, ci si assicura che la somma delle variabili slack decresca in maniera monotona di iterazione in iterazione.

La fase di miglioramento della qualità della soluzione segue una semplice logica: si risolve iterativamente OMIP finché non è più possibile ottenere un miglioramento. Anche se idealmente questa fase potrebbe continuare all'infinito, nella pratica si è osservato che il ciclo termina perché viene trovata la stessa soluzione due volte di fila.

3.3 Metodo adottato per i test

Trattandosi di un algoritmo euristico non è possibile dimostrare rigorosamente che una combinazione di valori assegnati ai parametri è migliore rispetto ad un altra. Si tratta quindi di verificare in via sperimentale quale combinazione offre le migliori prestazioni. Per semplificare l'analisi i parametri sono stati considerati indipendenti: sono stati esaminati uno alla volta e per ognuno è stato scelto il valore che ha prodotto i risultati migliori. Infine, per ogni combinazione di parametri, sono stati svolti tre test, uno per ciascun seme, e i risultati finali sono stati calcolati tramite media aritmetica.

3.4 Risultati

3.4.1 Primo test: scelta parametro α

Nel primo test è stato preso in considerazione il parametro $\theta \in \{20, 50, 80\%\}$.

Parametro	**Valore**
`MAX_ITR`	10
`dettime_lim`	$\mu(10^3, 100)$
θ	1%
ρ	*Soggetto test*

Tabella 3.1: Parametri utilizzati per il primo test.

ρ	**# soluzioni**	**Iteraz. medie**	**Tempo medio** [s]	**Qualità media**
20%	163	2.06	23.33	23.42%
50%	132	2.6	20.34	37.06%
80%	95	3	11.13	58.67%

Tabella 3.2: Risultati del primo test.

3.4.2 Secondo test: scelta parametro dettime_lim

Nel secondo test è stato considerato il parametro `dettime_lim`.

Parametro	**Valore**
`MAX_ITR`	10
`dettime_lim`	*Soggetto test*
θ	1%
ρ	20%

Tabella 3.3: Parametri utilizzati per il secondo test.

`dettime_lim`	**# soluzioni**	**Iteraz. medie**	**Tempo medio** [s]	**Qualità media**
$\mu(10 \cdot 10^3, 100)$	179	2.2	46.01	17.91%
$\mu(15 \cdot 10^3, 100)$	181	2.16	55.77	16.86%
$\mu(20 \cdot 10^3, 100)$	186	2.3	72.06	17.79%

Tabella 3.4: Risultati del secondo test.

3.4.3 Terzo test: scelta parametro MAX_ITR

Il parametro preso in considerazione in questo test è `MAX_ITR`.

Parametro	**Valore**
`MAX_ITR`	*Soggetto test*
`dettime_lim`	$\mu(20 \cdot 10^3, 100)$
θ	1%
ρ	20%

Tabella 3.5: Parametri utilizzati per il terzo test.

`MAX_ITR`	**# soluzioni**	**Iteraz. medie**	**Tempo medio** [s]	**Qualità media**
11	187	2.32	72.32	17.82%
12	187	2.33	72.73	17.81%
13	188	2.37	74.38	17.82%

Tabella 3.6: Risultati del terzo test.

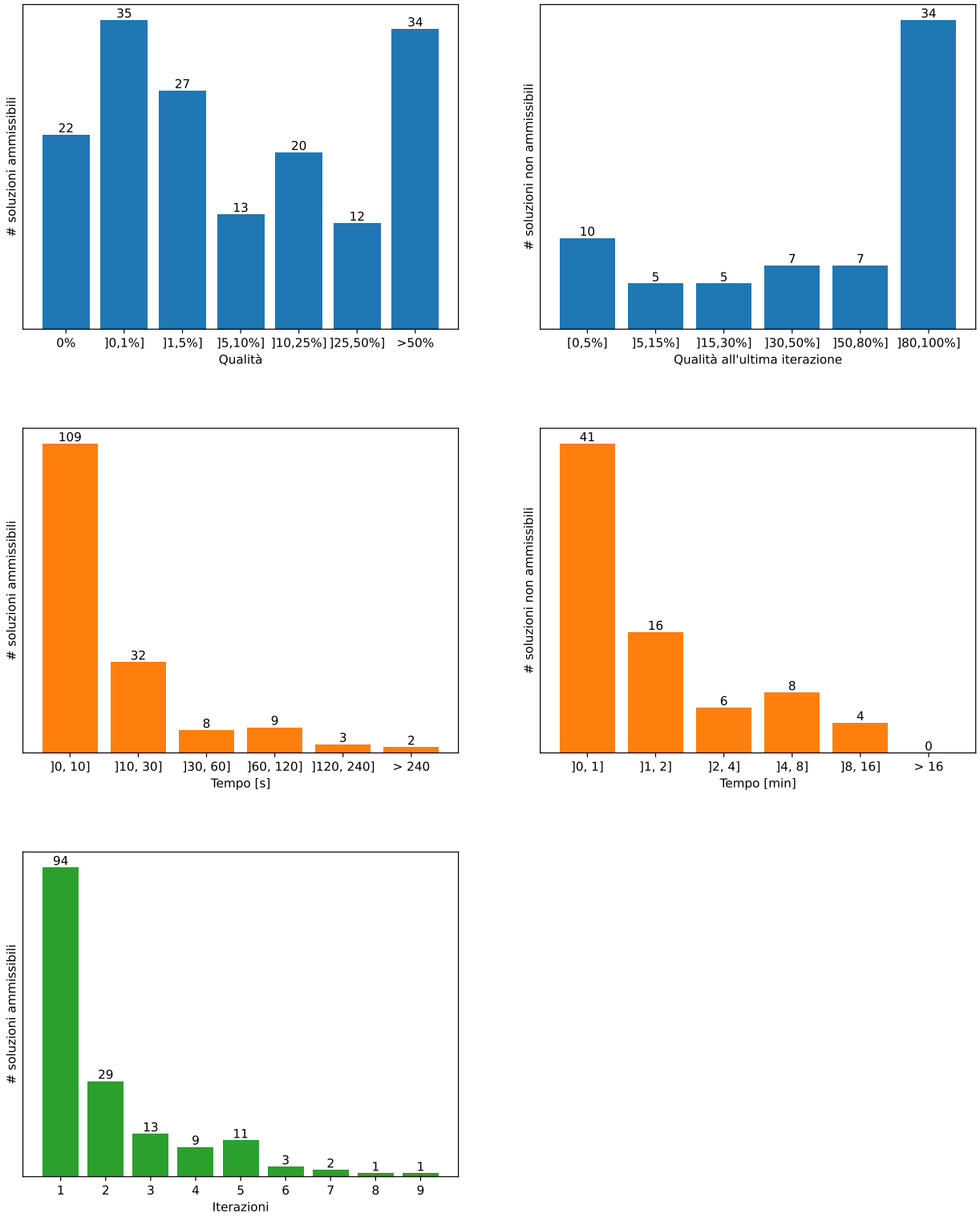

Figura 3.1: Istogrammi con i risultati del primo test.

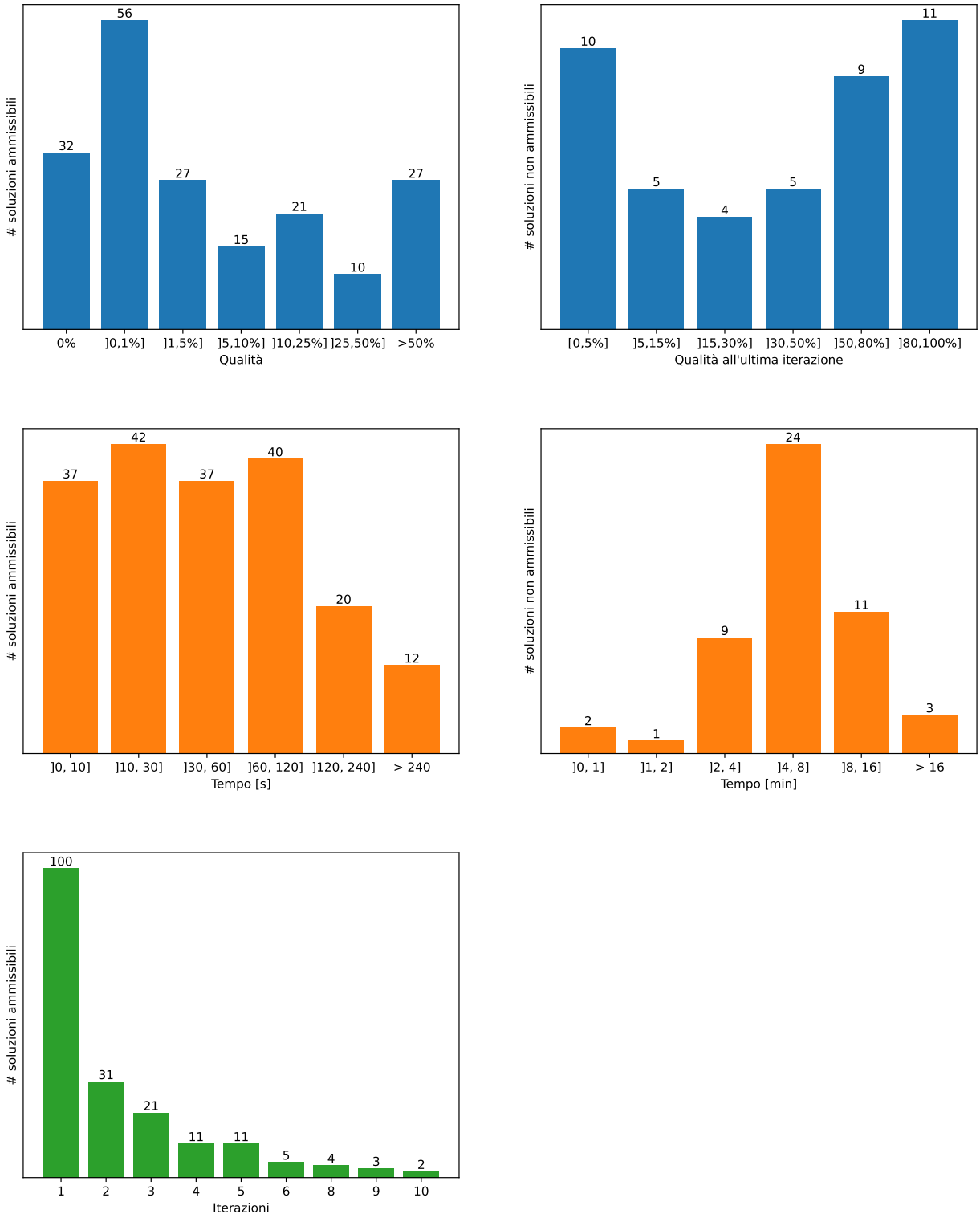

Figura 3.2: Istogrammi con i risultati del secondo test.

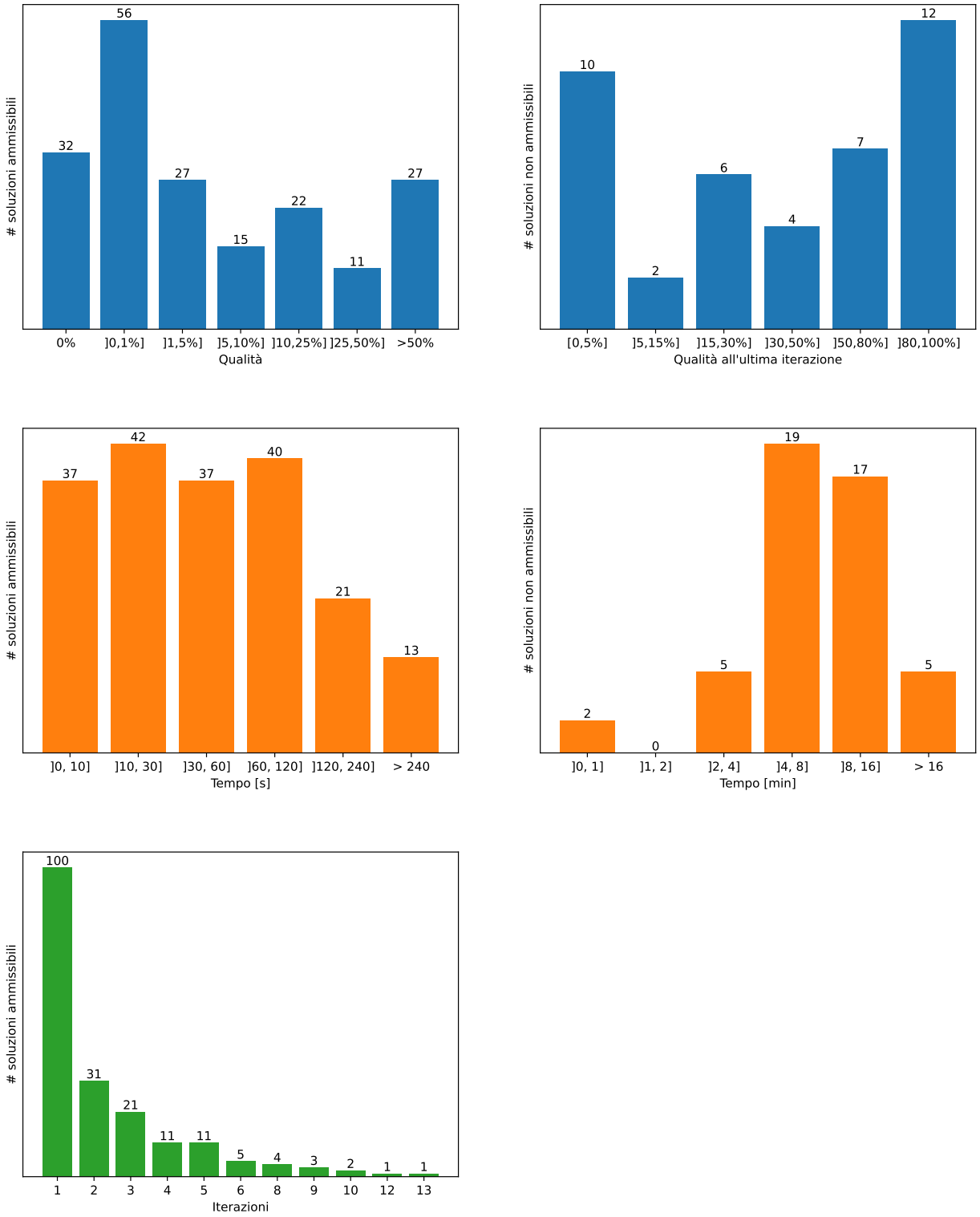

Figura 3.3: Istogrammi con i risultati del terzo test.

Capitolo 4

Conclusioni

4.1 Analisi dei risultati

Nel primo test è stato fatto variare il valore della percentuale di variabili intere fissate, i migliori risultati si ottengono con il 20%. Negli altri casi il problema principale è che vengono fissate troppe variabili, di conseguenza il valore della somma delle slack subisce minimi decrementi di iterazione in iterazione e all'ultima risulta ancora diverso da zero.

Nel secondo test è stato preso in considerazione il parametro che indica il tempo massimo che l'ottimizzatore ha a disposizione per risolvere un problema. Si tratta del parametro che incide di più sui risultati ed è il più difficile da settare: se il suo valore è molto alto, vengono effettuate molte iterazioni e il tempo totale cresce notevolmente; viceversa se il valore è troppo piccolo potrebbero avvenire minimi miglioramenti che non permettono di ottenere una soluzione ammissibile per il problema fornito in input all'algoritmo. Dai risultati si nota che aumentando il tempo vengono trovate sempre più soluzioni ammissibili, si potrebbe quindi aumentare ulteriormente il valore ma potrebbe risultare svantaggioso in termini di tradeoff tempo/risultato.

Nel terzo e ultimo test è stato modificato il valore del massimo numero di iterazioni dell'algoritmo. I risultati ottenuti sono pressoché identici a quelli del test precedente, il numero di soluzioni ammissibili trovate aumenta solamente di due unità. Questo significa che nei casi rimanenti, dopo 10 iterazioni, il grado di inammissibilità della soluzione è ancora molto alto e poche iterazioni in più non bastano per ottenere una soluzione ammissibile, oppure il grado può essere basso ma l'algoritmo non converge ad una soluzione. Andando ad aumentare il valore massimo, il numero di soluzioni ammissibili potrebbe crescere, ma anche in questo caso potrebbe non essere conveniente in termini di costi computazionali.

4.2 Possibili miglioramenti

Attualmente il problema principale dell'algoritmo è che i casi in cui non viene trovata una soluzione ammissibile per il MIP di partenza, richiedono molto tempo. Ad esempio, nel miglior caso del test 3, è stata trovata una soluzione ammissibile per 190 problemi in circa 4 ore, mentre i restanti 60 hanno richiesto un totale di 6 ore. Per migliorare questo aspetto, così come altre statistiche, è necessario trovare una strategia che, basandosi su una qualche caratteristica del problema passato in input, scelga dinamicamente il valore da assegnare a ciascun parametro. Trovare un algoritmo del genere che funzioni per problemi MIP generici può essere arduo. Forse l'utilizzo dell'intelligenza artificiale potrebbe aiutare a trovare caratteristiche non così evidenti ma che permettono di ottenere prestazioni migliori.

Infine l'utilizzo della parallelizzazione, come esposto in [7], consente di migliorare le prestazioni.

Elenco delle figure

Elenco delle tabelle

Elenco degli Algoritmi

Bibliografia

[1] D. Salvagnin, «Introduzione all'ottimizzazione discreta,» pp. 1,3–4, 2018. indirizzo: https://www.dei.unipd.it/~salvagni/didattica/intro.

[2] D. Salvagnin, «Cenni di Programmazione Lineare,» p. 1, 2011. indirizzo: https://www.dei.unipd.it/~salvagni/didattica/lp.

[3] D. Salvagnin, «Cenni di Programmazione Lineare Intera,» 2011. indirizzo: https://www.dei.unipd.it/~salvagni/didattica/mip.

[4] IBM. «What are heuristics?» (2022), indirizzo: https://www.ibm.com/docs/en/icos/22.1.1?topic=heuristics-what-are.

[5] T. Berthold, «Primal Heuristics for Mixed Integer Programs,» pp. 5–6, 2006. indirizzo: https://opus4.kobv.de/opus4-zib/files/1029/Berthold_Primal_Heuristics_For_Mixed_Integer_Programs.pdf.

[6] IBM. «IBM ILOG CPLEX Optimization Studio.» (2022), indirizzo: https://www.ibm.com/docs/en/icos/22.1.1.

[7] L. Munguía, S. Ahmed, D. Bader, G. Nemhauser e Y. Shao, «Alternating criteria search: A parallel large neighborhood search algorithm for mixed integer programs,» English (US), *Computational Optimization and Applications*, vol. 69, n. 1, pp. 1–24, gen. 2018, Publisher Copyright: © Springer Science+Business Media, LLC 2017., ISSN: 0926-6003. DOI: 10.1007/s10589-017-9934-5.

[8] A. Gleixner, G. Hendel, G. Gamrath et al., «MIPLIB 2017: Data-Driven Compilation of the 6th Mixed-Integer Programming Library,» *Mathematical Programming Computation*, 2021. DOI: 10.1007/s12532-020-00194-3. indirizzo: https://doi.org/10.1007/s12532-020-00194-3.

[9] IBM. «Callable Library.» (2022), indirizzo: https://www.ibm.com/docs/en/icos/22.1.1?topic=apis-callable-library.

[10] «ACS-MIP.» (2023), indirizzo: https://github.com/luca037/ACS-MIP.

[11] IBM. ≪Deterministic time limit.≫ (2022), indirizzo: https://www.ibm.com/docs/en/icos/22.1.1?topic=parameters-deterministic-time-limit.

[12] IBM. ≪Starting from a solution: MIP starts.≫ (2022), indirizzo: https://www.ibm.com/docs/en/icos/22.1.1?topic=mip-starting-from-solution-starts.