UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Corso di Laurea Triennale in Ingegneria Informatica

"Implementazione dell'euristico ZIRound per problemi MIP"

RELATORE:
Prof. **Domenico Salvagnin**

LAUREANDO:
Niccoló Turcato
Matr:1168688

Anno Accademico 2019/2020
Data di laurea: 24/09/2020

Indice

Abstract

Il presente documento ha lo scopo di esporre l'algoritmo euristico "ZI Round" per problemi MIP, inventato dal prof Chris Wallace nel 2009, presentarne un approccio in linguaggio di programmazione Python e introdurre un'estensione che ne migliora le prestazioni.
Dato un problema MIP, il goal dell'algoritmo è cercare di risolverlo partendo dalla soluzione del rilassamento lineare, andando ad arrotondare le variabili intere a valore frazionario, utilizzando gli slack dei vincoli per mantenere il loro soddisfacimento.
Il primo obiettivo prefissato per questo approccio, è la riproduzione dei risultati originariamente ottenuti dall'autore, per verificare ciò, il codice è stato eseguito sullo stesso set di problemi utilizzati dall'autore: MIPLIB 2003.
Nella seconda parte del documento vengono presentate una serie di estensioni, atte a migliorare le prestazioni del codice.
Infine vengono presentati i risultati ottenuti dall'algoritmo e le sue estensioni per il benchmark della MIPLIB 2010.

Introduzione

1 Paradigma MIP

Mixed integer programming - Programmazione lineare misto-intera

Un problema di programmazione lineare intera (misto) è definito da:

- **Variabili:** ognuna definita da nome, dominio, upper bound e lower bound
- **Funzione obiettivo:** funzione lineare delle variabili del problema, da minimizzare o massimizzare
- **Vincoli lineari:** coinvolgono le variabili del problema
- **Vincoli di interezza:** Ad alcune o a tutte le variabili viene imposto il vincolo di assumere solo valori interi

Una generica struttura di un problema MIP è:

$$\begin{cases} min/max\ c^T x \\ a_i^T \cdot x \sim b_i\ i = 1, \ldots, m \\ l_j \leq x_j \leq u_j\ j = 1, \ldots, n = N \\ x_j \in \mathbb{Z} \quad \forall j \in G \subseteq N = \{1, \ldots, n\} \end{cases} \tag{1}$$

Il paradigma MIP, permette di modellare un problema di ottimizzazione, al costo di una restrizione dei vincoli del problema a formule lineari matematiche. Questo porta a notevoli vantaggi nello studio e lo sviluppo di algoritmi risolutivi.
Da un problema MIP si può derivare un altro problema effettuando operazioni di rimozione di vincoli o sostituendo la funzione obiettivo con una versione approssimata. Questo problema derivato si chiama "rilassamento" del problema originario, e le operazioni di rimozione e approssimazione si chiamano "rilassamenti".

Si chiama rilassamento continuo (o lineare) un problema di programmazione lineare derivato da un problema di programmazione lineare intera rilassando i vincoli di interezza imposti sulle variabili.

I dati necessari ad un calcolatore per memorizzare un modello MIP sono:

- **Lista variabili:** ogni variabile può essere identificata da un indice o un nome. Si definiscono anche il vettore dei Domini di appartenenza (variabili continue, intere o binarie), i vettori di upper bound e lower bound
- **Coefficienti funzione obiettivo:** vettore numerico che identifica i coefficienti con cui le variabili compaiono nella funzione obiettivo. In letteratura spesso si indica con un vettore colonna chiamato c.
- **Senso della funzione obiettivo:** semplice variabile che indica se il problema è una minimizzazione o massimizzazione
- **Matrice e vettori dei vincoli:** i vincoli possono essere espressi nella forma

$$A \cdot x \sim b$$

  dove A è una matrice, x e b sono vettori.
  Vengono quindi memorizzati la matrice A che identifica i lhs delle formule dei vincoli, il vettore b che invece contiene i rhs e un ulteriore vettore che contiene i "sensi" dei vincoli.

NOTA: una variabile binaria può essere semplicemente trattata come una variabile intera con UB=1 e LB=0.

2 Algoritmo Euristico

In informatica e matematica, un alg. euristico è un tipo di algoritmo progettato per risolvere un problema in maniera veloce e tipicamente approssimata. Si utilizzano quando i metodi classici di risoluzione di un problema hanno complessità computazionale elevata, e di conseguenza, tempi di esecuzione molto lunghi per il caso d'uso. I risultati dell'applicazione di un euristico a un problema costituiscono in genere un tradeoff tra ottimizzazione, completezza, accuratezza e velocità di esecuzione.
Applicazioni di questi tipi di algoritmi si trovano spesso in intelligenza artificiale, e nel campo d'interesse di questa tesi: l'ottimizzazione discreta.

3 Introduzione a ZI Round

ZI Round è un algoritmo euristico sviluppato da Chris Wallace (East Tennessee State University, Johnson City, USA) per la risoluzione di problemi di ottimizzazione MIP.
L'algoritmo viene proposto come estensione di un altro euristico, chiamato Simple Round, introdotto nel 2008 nel risolutore non commerciale SCIP 1.1 assieme ad altri 23 euristici.
Simple Round risolve un problema di ottimizzazione MIP a partire dalla soluzione del suo rilassamento lineare. La strategia dell'algoritmo è quella di osservare i valori delle variabili che nel problema originale hanno il vincolo di interezza, e arrotondare i valori non interi dove può essere fatto banalmente.
Ipotizzando di avere solo vincoli $\leq$, una variabile x_j può essere banalmente arrotondata per difetto se tutti i coefficienti nella colonna j della matrice A sono non negativi ($a_{ij} \geq 0 \; \forall i$). Analogamente se tutti i coefficienti sono non positivi ($a_{ij} \leq 0 \; \forall i$) allora x_j può essere semplicemente arrotondata per eccesso.
Si definiscono anche le quantità di down-locks e up-locks per una variabile x_j, rispettivamente: il numero di coefficienti $a_{ij} < 0$ e il numero di coefficienti $a_{ij} > 0$. Una variabile con zero down-locks può essere arrotondata per difetto, viceversa se ha zero up-locks.

Algoritmo 1 Simple Rounding

INPUT: Soluzione x del rilassamento lineare, lista G
for $x_j \in G$: **do**
  if x_j ha zero down-locks **then**
    $x_j = \lfloor x_j \rfloor$
  else if x_j ha zero up-locks **then**
    $x_j = \lceil x_j \rceil$
  end if
end for

NOTA: si può agire in maniera analoga considerando solo vincoli $\geq$.
L'interesse verso questo algoritmo viene dal fatto che dimostra buone prestazioni per diverse istanze di problemi ed è molto semplice e veloce, infatti la complessità computazionale è $\mathcal{O}(z \cdot n)$, dove n è il numero di variabili intere del problema, e z il massimo numero coefficienti non zero, di una variabile, nei vincoli.
La struttura dell'algoritmo potrebbe portare a pensare che la complessità sia $\Theta(n)$, ma non si considera che vanno contati i down e up locks per ogni variabile, che ha

complessità $\mathcal{O}(z)$, ipotizzando di poter scorrere la matrice A sia per righe che per colonne.

Parte I

L'algoritmo

Capitolo 1

ZI Round V1

1.1 Definizioni

Ipotizzando un problema di programmazione lineare misto-intera, con vincoli di tipo $\leq$ e $=$, al fine della descrizione e sviluppo dell'algoritmo in oggetto, si definiscono i seguenti oggetti:

- $ZI(x_j) := min\{x_j - \lfloor x_j \rfloor, \lceil x_j \rceil - x_j\}$, frazionalità di x_j, una misura della non interezza della variabile.
- $ZI(x) := \sum_{i \in G} ZI(x_j)$, misura dell'infeasibility intera di una soluzione x per un problema, dove G è la lista degli indici delle variabili con vincolo di interezza.
- Slack s_i di un vincolo: dato un vincolo $a_i x \leq b_i$, si definisce $s_i := b_i - a_i x$, il vincolo è rispettato se $s_i \geq 0$.
- $ub_{x_j} := min_i\{\frac{s_i}{a_{ij}} : a_{ij} > 0\}$, upper bound sullo spostamento di una variabile x_j guardando i vincoli.
- $lb_{x_j} := min_i\{\frac{-s_i}{a_{ij}} : a_{ij} < 0\}$, lower bound sullo spostamento di una variabile x_j guardando i vincoli.
- $UB_{x_j} := min\{ub,\ upperbound(x_j) - x_j\}$.
- $LB_{x_j} := min\{lb,\ x_j - lowerbound(x_j)\}$.
- $s := \{s_1, \ldots, s_m\}$, lista degli slacks dei vincoli del problema.
- Singleton (variabile): variabile di un problema che compare in un unico vincolo, ha un solo coefficiente non zero nella rispettiva colonna della matrice A.

1.2 Descrizione

Come Simple Round, anche questo algoritmo risolve un problema MIP a partire dalla soluzione del rilassamento continuo, e invece ipotizza vincoli di tipo $\leq$ o $=$.
La strategia di ZI Round è cercare, nella soluzione del rilassamento, tra le variabili con vincolo di interezza quelle che possono essere arrotondate per migliorare lo ZI complessivo, finchè questo non diventa zero.
Una soluzione x tale che $ZI(x) = 0$ risolve il problema MIP originale, ma non è assicurato che sia quella ottimale.
Per gli arrotondamenti, ZI Round calcola di quanto una variabile x_j può essere spostata all'interno dei suoi limiti (superiore e inferiore) mantenendo il soddisfacimento dei vincoli del problema. Uno spostamento di una variabile deve mantenere gli slack s_i dei vincoli non negativi.
Quindi x_j non può essere spostata in alto più di $ub_{x_j} := min_i\{\frac{s_i}{a_{ij}} : a_{ij} > 0\}$, dati gli slacks correnti $s = \{s_1, \ldots, s_m\}$. Inoltre lo shifting di x_j è limitato in alto da $UB := min\{ub,\ upperbound(x_j) - x_j\}$, poichè va rispettato l'upper bound di x_j.
Analogamente x_j non può essere spostata in basso più di $LB := min\{lb,\ x_j - lowerbound(x_j)\}$, con $lb_{x_j} := min_i\{\frac{-s_i}{a_{ij}} : a_{ij} < 0\}$.
Nel corpo dell'algoritmo, per ogni variabile $x_j \in G$ si calcolano UB e LB e si decide la direzione dello spostamento della variabile.
Si applica lo spostamento $x_j = x_j + UB$ se esso migliora lo ZI della variabile ed è più conveniente rispetto a $x_j = x_j - LB$. Viceversa si applica $x_j = x_j - LB$.
Nel caso in cui gli spostamenti siano equivalenti ($ZI(x_j + UB) = ZI(x_j - LB)$) allora viene scelto quello che migliora la funzione obiettivo.
L'algortmo termina se non sono più possibili arrotondamenti, oppure tutte le variabili sono state arrotondate.
Se la soluzione x' ritornata da ZI Round ha $ZI(x') = 0$, e rispetta vincoli e bounds, allora essa è una soluzione per il problema MIP originale.
Per velocizzare l'esecuzione dell'euristico, è possibile definire un valore limite (threshold o ϵ) per il calcolo di UB e LB. Se nel calcolo di UB e LB per una variabile x_j, essi cadono entrambi $< \epsilon$, allora si interrompe la computazione, e si considerano entrambi $= 0$.
Nell'implementazione di Wallace, il valore utilizzato per il threshold era: $\epsilon = 10^{-5}$.
Questo è utile perché per piccoli valori di LB e UB le variabili cambiano molto poco, e risulta più conveniente cercare di arrotondare altre variabili in modo tale da aggiornare gli slacks e se possibile arrotondare x_j in un'iterazione successiva.

Infine ora è possibile spiegare perché ZI Round viene presentato come estensione diretta di Simple Round: se una variabile x_j può essere arrotondata banalmente verso il basso, allora x_j ha zero down-locks.
Ciò si traduce in $lb_{x_j} := min_i\{\frac{-s_i}{a_{ij}} : a_{ij} < 0\} = \infty$, quindi si può arrotondare x_j al suo lower bound. È chiaro perciò che ZI Round può arrotondare qualsiasi variabile arrotondabile con Simple Round.
Il grande vantaggio di ZI Round rispetto al suo predecessore Simple Round, è la possibilità di muovere le variabili verso i margini dei vincoli, il che, nella maggior parte dei problemi di ottimizzazione, in genere migliora il valore della funzione obiettivo.

Algoritmo 2 ZI Round

INPUT: Soluzione x del rilassamento lineare, lista G delle variabili intere
repeat
  for $x_j \in G$ *and* $ZI(x_j)$! = 0 **do**
    Update slacks $s = \{s_1, \ldots, s_m\}$
    $UB,\ LB = Calcola(threshold = \epsilon)$
    if $ZI(x_j + UB) == ZI(x_j - LB)$ *and* $ZI(x_j + UB) < ZI(x_j)$ **then**
      Aggiorna x_j per migliorare la funzione obiettivo
    else if $ZI(x_j + UB) < ZI(x_j - LB)$ *and* $ZI(x_j + UB) < ZI(x_j)$ **then**
      $x_j = x_j + UB$
    else if $ZI(x_j - LB) < ZI(x_j + UB)$ *and* $ZI(x_j - LB) < ZI(x_j)$ **then**
      $x_j = x_j - LB$
    end if
  end for
until Nessun aggiornamento possibile

Questa versione dell'algoritmo, tuttavia ha difficoltà a risolvere istanze di problemi che presentano vincoli con equazioni, che legano le variabili intere da arrotondare. Infatti, una uguaglianza soddisfatta ha slack = 0, e dato che l'algoritmo prende in input la soluzione del rilassamento lineare, la quale per definizione di soluzione, rispetta i vincoli, allora tutte le uguaglianze hanno slack nullo.
La soluzione presentata da Wallace per questo problema è una semplice estensione sul calcolo degli slacks dei vincoli: in ogni uguaglianza si controlla se esiste una variabile continua che ha coefficiente non zero, solo per quel vincolo (in letteratura queste variabili si chiamano singleton). Se ne esiste una, allora essa viene usata come slack del vincolo, dove l'escursione diventa quella della variabile continua, data dai suoi upper e lower bound, modulata dal coefficiente nel vincolo.

A questo punto l'algoritmo diventa:

Algoritmo 3 ZI Round esteso

INPUT: Soluzione x del rilassamento lineare, lista G delle variabili intere
Cerca i singleton tra i vincoli
repeat
  for $x_j \in G \; and \; ZI(x_j) \; != \; 0$ **do**
    $Update \; slacks \; s = \{s_1, \ldots, s_m\}$ includendo anche quelli artificiali
    $UB, \; LB = Calcola(threshold = \epsilon)$
    if $ZI(x_j + UB) \; == \; ZI(x_j - LB) \; and \; ZI(x_j + UB) < ZI(x_j)$ **then**
      Aggiorna x_j per migliorare la funzione obiettivo
    else if $ZI(x_j + UB) \; < \; ZI(x_j - LB) \; and \; ZI(x_j + UB) < ZI(x_j)$ **then**
      $x_j = x_j + UB$
    else if $ZI(x_j - LB) \; < \; ZI(x_j + UB) \; and \; ZI(x_j - LB) < ZI(x_j)$ **then**
      $x_j = x_j - LB$
    end if
    Ripristina le variabili singleton per soddisfare i vincoli
  end for
until Nessun aggiornamento possibile

1.3 Analisi Complessità

Si utilizzi un modello di costo per cui le operazioni aritmetiche hanno costo unitario. L'algoritmo è composto da:

- Fase di inizializzazione (per la versione estesa)
- Ciclo Esterno: **repeat** ... **until** (Nessun Aggiornamento possibile)
- Ciclo Interno: **for** $(x_j \in G \; and \; ZI(x_j) \; != \; 0)$
- Corpo interno del codice

Dato un problema MIP, si definiscano le quantità:

- n = numero di variabili del problema $= n_i + n_c$
- n_i = numero di variabili intere
- $\overline{n_i}$ = numero di variabli intere da arrotondare
- n_c = numero di variabli continue

- m = numero di vincoli
- m_e = numero di vincoli di uguaglianza
- $\overline{m_e}$ = numero di vincoli con singleton = numero di singleton (utilizzati)
- z : il massimo numero coefficienti non zero, di una variabile, nei vincoli.

Ipotizzando di poter attraversare la matrice A sia per righe che per colonne, il corpo interno nella sua interezza ha complessità computazionale $\mathcal{O}(z)$, principalmente data dal calcolo degli slack dei vincoli, UB e LB.
Il ripristino dei singleton ($\mathcal{O}(\overline{m_e})$), incide sulla complessità a seconda del tipo di problema, comunque vale questa relazione: $0 \leq \overline{m_e} \leq m$.
La fase di inizializzazione, in questa versione, consiste nella ricerca delle variabili singleton, la quale ha complessità $\Theta(n_c \cdot m_e) = \Theta(n \cdot m_e)$, dato che per ogni vincolo di uguaglianza va controllata la rispettiva riga nella matrice A e per ogni variabile continua che compare nel vincolo va controllata la rispettiva colonna.
Diventa complicata l'analisi dei due cicli, dato che il numero di esecuzioni dipende esclusivamente dalle caratteristiche dei vincoli del problema in input e dalla soluzione di partenza.
Comunque si possono ottenere le complessità al caso migliore e al peggiore approssimate, ipotizzando una semplificazione nell'analisi: si assume che quando l'algoritmo cerca di arrotondare una variabile, questa sia esclusivamente arrotondata oppure non modificata. In questo modo si semplifica notevolmente l'analisi al caso peggiore.
Il **caso migliore** si ottiene semplicemente quando tutte le variabili intere vengono arrotondate alla prima esecuzione del ciclo esterno, quindi l'algoritmo termina perché ha raggiunto lo scopo, oppure se nessuna variabile viene modificata e di conseguenza gli slack dei vincoli rimangono quelli originali, dunque permettendo nessun arrotondamento. In questo caso si ha una esecuzione del ciclo esterno con $\overline{n_i}$ esecuzioni del corpo interno e una seconda esecuzione a vuoto del ciclo esterno se le variabili sono state arrotondate.
Il **caso peggiore** si ha (in caso di successo) se per ogni esecuzione del ciclo esterno, il ciclo for interno opera un unico arrotondamento ogni volta,
quindi $\sum_{i=1}^{\overline{n_i}} i = \frac{1}{2}\overline{n_i}(\overline{n_i}+1) = \mathcal{O}(\overline{n_i}^2)$ ripetizioni del corpo interno.
Complessivamente:

- **Best case:** $C(n, \overline{n_i}, z, m_e) = \Theta(n \cdot m_e) \ + \ \mathcal{O}(\overline{n_i} \ \cdot \ z) = \mathcal{O}(n \cdot z)$
- **Worst case:** $C(n, \overline{n_i}, z, m_e) = \Theta(n \cdot m_e) \ + \ \mathcal{O}(\overline{n_i}^2 \ \cdot \ z) = \mathcal{O}(n^2 \cdot z)$

Capitolo 2

ZI Round V2

2.1 Descrizione

Il prof. Wallace presenta anche una seconda versione dell'algoritmo, che in alcuni casi migliora le prestazioni dal punto di vista della funzione obiettivo del problema. Questa evoluzione prevede di modificare anche le variabili intere non frazionali, cioè con vincolo di interezza rispettato, con lo scopo di migliorare la funzione obiettivo. Ciò può essere realizzato semplicemente estendendo il ciclo più interno a tutte quante le variabili con vincolo di interezza e impostando threshold=1 per quelle con valore già intero.
Questa seconda versione mantiene la stessa complessità computazionale della prima, ma nella pratica potrebbe essere molto meno performante dal punto di vista temporale.
Infatti non è escluso che una soluzione del rilassamento lineare non contenga più variabili intere già arrotondate che non, il che aumenta considerevolmente il tempo di calcolo nella seconda versione.
Queste considerazioni sono confermate dai risultati sperimentali di Wallace, che trovano tempo di esecuzione raddoppiato nella seconda versione, per i benchmark eseguiti.

Lo pseudocodice rimane quasi invariato:

Algoritmo 4 ZI Round V2

INPUT: Soluzione x del rilassamento lineare, lista G delle variabili intere
Cerca i singleton tra i vincoli
repeat
  for $x_j \in G$ **do**
    Update slacks $s = \{s_1, \ldots, s_m\}$ includendo anche quelli artificiali
    if $ZI(x_j) == 0$ **then**
      $UB,\ LB = Calcola(threshold = 1)$
      Aggiorna x_j per migliorare la funzione obiettivo
    else
      $UB,\ LB = Calcola(threshold = \epsilon)$
    end if
    if $ZI(x_j + UB)\ ==\ ZI(x_j - LB)\ and\ ZI(x_j + UB) < ZI(x_j)$ **then**
      Aggiorna x_j per migliorare la funzione obiettivo
    else if $ZI(x_j + UB)\ <\ ZI(x_j - LB)\ and\ ZI(x_j + UB) < ZI(x_j)$ **then**
      $x_j = x_j + UB$
    else if $ZI(x_j - LB)\ <\ ZI(x_j + UB)\ and\ ZI(x_j - LB) < ZI(x_j)$ **then**
      $x_j = x_j - LB$
    end if
    Ripristina le variabili singleton per soddisfare i vincoli
  end for
until Nessun aggiornamento possibile

2.2 Analisi Complessità

Valgono le stesse considerazioni presentate nella **sezione 1.3** per la prima versione dell'algoritmo, per ZI Round V2 cambia solo l'esecuzione del ciclo for interno che è applicato a tutte le n_i variabili intere del problema in input.
Complessivamente:

- **Best case:** $C(n, n_i, z, m_e) = \Theta(n \cdot m_e)\ +\ \mathcal{O}(n_i \cdot z) = \mathcal{O}(n \cdot z)$
- **Worst case:** $C(n, n_i, z, m_e) = \Theta(n \cdot m_e)\ +\ \mathcal{O}(n_i^2 \cdot z) = \mathcal{O}(n^2 \cdot z)$

Va aggiunto che questa analisi, come per la prima versione, presenta forti approssimazioni soprattutto per gli ultimi passaggi, dove si presentano le forme finali delle funzioni di complessità.

Questo viene confermato dal fatto che le due versioni ottengono tempi di calcolo drasticamente differenti. Le formule più precise sono quelle nella forma $(\Theta + \mathcal{O})$

Parte II

Implementazione

Capitolo 3

Aproccio all'algoritmo

3.1 Scelte implementative

Gli elementi necessari a sviluppare un approccio a questo algoritmo sono:

- Un **risolutore** per problemi di ottimizzazione lineare (e misto-intera)
- Un **linguaggio di programmazione** compatibile con il risolutore scelto
- **Ambiente di sviluppo** e strumenti di **controllo versione**
- Un pool di problemi di **ottimizzazione lineare misto-intera** adeguato per la raccolta dati

Nell'articolo che descrive originariamente l'algoritmo, vengono presentati i risultati sperimentali per la libreria di problemi di ottimizzazione MIPLIB 2003, disponibile al momento della stesura, in cui i problemi sono forniti in formato *mps*.
Perciò è importante avere a disposizione un risolutore in grado di leggere i dati in questo formato, per ottimizzare lo sviluppo ed essere in grado di confrontare i risultati di questa implementazione con quelli originari.
È stato quindi selezionato il risolutore ILOG CPLEX 12.10 di IBM, perché possiede la funzionalità sopracitata e l'interfaccia per il linguaggio di programmazione scelto è molto concreta e adatta agli scopi implementativi.
Il linguaggio scelto è Python, per la compatibilità con il risolutore scelto e la sua globale semplicità, in accoppiata con l'IDE PyCharm, che incorpora il controllo di sintassi e stile per PEP 8, e gli strumenti di controllo versione git/github.
L'algoritmo non necessita di particolari strutture, se non le liste per memorizzare i coefficienti e le variabili. Un'accortezza che migliora notevolmente il consumo di

memoria è l'utilizzo di una struttura dati *sparsa* per la matrice A, dato che la dimensione $(n \cdot m)$ può essere molto grande e comunque la densità dei coefficienti non zero nei vincoli, nella maggior parte dei problemi è nell'ordine di 10^{-4}. In questa implementazione è stata utilizzata la struttura dati $cplex.SparsePair$ per memorizzare la matrice dei vincoli lineari per righe e colonne sparse. Inoltre ogni istanza deve essere preprocessata convertendo eventuali vincoli $\geq$ in $\leq$.

$$(a^T \cdot x \geq b) \equiv (-a^T \cdot x \leq -b) \tag{3.1}$$

3.2 Descrizione dell'approccio

Il primo obiettivo da raggiungere di questa implementazione è certamente la riproduzione dei risultati originariamente ottenuti dal creatore dell'algoritmo.
Come anticipato, è data la lista P dei problemi di ottimizzazione su cui originariamente è stato testato con successo l'algoritmo, per ogni problema $p \in P$ risolto da ZIRound è disponibile il valore funzione obiettivo ottenuto con la soluzione trovata dall'euristico. Inoltre nella MIPLIB è possibile trovare le soluzione ottime (o *best known*).
È stata quindi applicata la metodologia del test driven development, dunque è stata definita la struttura del test principale in questo modo:

Algoritmo 5 Metodologia di svilupo

Data una lista di problemi MIP P
$\forall p \in P : x_p$ = *miglior soluzione conosciuta di p*
for all $p \in P$ **do**
  p' = *rilassamento lineare di p*
  c = *obiettivo di p*
  $x' = Solve(p')$
  $x_r = ZIRound(x',\ p)$
  if x_r *rispetta i vincoli di p* **AND** $c^T \cdot x_r$ *compatibile con* $c^T \cdot x_p$ **then**
    L'algoritmo esegue correttamente sull'istanza p
  else
    L'algoritmo **NON** esegue correttamente sull'istanza p
  end if
end for

Questo semplice approccio già fornisce molte informazioni sulla correttezza del codice, tra cui una certa misura di "affidabilità", in base al numero di istanze risolte

correttamente.
Un metro di misura per la verifica della riuscita dell'implementazione è anche il confronto tra i valori di funzione obiettivo della soluzione raggiunta dall'euristico e della soluzione ottima (o la migliore nota).
Infatti, dato un problema della lista P, non è preoccupante che il codice arrivi ad una soluzione x_r migliore rispetto a quella dichiarata da Wallace, è invece motivo di sospetto il fatto che x_r sia migliore dell'ottimo (o del *best known*).
Questo perché, se $c^T \cdot x_r$ è migliore del risultato ottimo, allora x_r deve necessariamente non rispettare uno o più vincoli (espressioni lineari o sulle singole variabili), poiché per definizione, una soluzione ottima è la miglior possibile soluzione del problema che rispetta i vincoli.
Anche se per un problema si ha disponibile per confronto, solamente una soluzione *best known*, quindi non necessariamente l'ottimo, è assai poco probabile che ZI Round sia in grado di migliorarla, dato che essendo un euristico di arrotondamento, il suo *goal* è arrivare in tempi fruibili ad una soluzione accettabile e se possibile, non pessima.
Una situazione come quella sopra descritta può accadere se la soluzione trovata non rispetta i vincoli del problema, pertanto se i test sul soddisfacimento dei vincoli non segnalano errori, essi stessi contengono molto probabilmente degli errori. Altri indicatori utilizzati per i controlli sono:

- **Slack** dei vincoli negativi: per definizione si tratta di quantità positive
- **UB** e **LB** calcolati per le variabili: anch'essi positivi per definizione
- Congruità delle **dimensioni**

È stato possibile anche implementare per step le versioni dell'algoritmo, infatti per testare la prima versione (**Algoritmo 2**), è sufficiente escludere da P le istanze di problemi con vincoli che legano le variabili intere da arrotondare, per poi riaggiungerle quando si va a testare **Algoritmo 3**.
Nella documentazione di IBM è presente un interessante passaggio: *"There is no point in counting seconds if your problem works in terms of years"*, dato che i calcolatori hanno precisione finita, seppure sempre più alta, va tenuto conto di questo anche nei test.

3.3 Calcolo degli Slack

Il calcolo degli slack dei vincoli e degli scostamenti UB e LB occupano la gran parte dell'attività del corpo di ZI Round, per i vincoli $\leq$ e $\geq$ l'implementazione rispecchia direttamente le definizioni.
Un'accortezza, che viene sottointesa dal creatore dell'algoritmo, è quella di limitare il calcolo degli slack ai soli vincoli in cui compare la prossima variabile da arrotondare. Data una variabile x_j e il numero di coefficienti non zero nei vincoli z_j, si ha che in genere $z_j << m$ e se si ipotizza di poter leggere la matrice dei vincoli sia per righe che per colonne, questo migliora notevolmente le prestazioni temporali.

I vincoli di uguaglianza invece vengono trattati con l'estensione suggerita da Wallace nel seguente modo:

$$a^T \cdot x = b \tag{3.2}$$

3.2 descrive un generico vincolo di uguaglianza, dove a è il vettore dei coefficienti, x il vettore delle variabili e b uno scalare. Se il vincolo non contiene variabili continue singleton, allora non è possibile modificare le variabili che compaiono e quindi lo slack di questo dato vincolo viene assunto zero. Se invece il vincolo contiene almeno un singleton, si procede esprimemendolo come combinazione di **3.3** e **3.4**

$$a^T \cdot x \leq b \tag{3.3}$$

$$(a^T \cdot x \geq b) \rightarrow (-a^T \cdot x \leq -b) \tag{3.4}$$

È conveniente effettuare la trasformazione in **3.4** per avere solo vincoli $\leq$ e non cambiare il resto della struttura dell'algoritmo.

$$(a^T \cdot x = b) \equiv \begin{cases} a^T \cdot x \leq b \\ a^T \cdot x \geq b \end{cases} \equiv \begin{cases} a^T \cdot x \leq b \\ -a^T \cdot x \leq -b \end{cases} \tag{3.5}$$

Gli slack artificiali a questo punto sono 2 per equazione, uno per ognuna di **3.3** e **3.4**, sia x_j la variabile singleton, sia a_{ij} il coefficiente con cui compare nel vincolo.

$$s_i = \begin{cases} a_{ij} \cdot (x_j - lowerbound(x_j)), \ if \ a_{ij} > 0 \\ a_{ij} \cdot (upperbound(x_j) - x_j), \ if \ a_{ij} < 0 \end{cases} \tag{3.6}$$

Il valore degli slack artificiali rappresenta di quanto è possibile, durante la fase di

arrotondamento delle variabili intere, spostare verso l'alto o verso il basso il lhs del vincolo **3.2**, a seconda dell'escursione del singleton.
Poiché la variabile singleton deve correggere la modifica del vincolo, quest'ultima deve essere limitata da upper e lower bound della variabile, trattare l'uguaglianza come descritto in **3.5** permette di non modificare l'architettura di Zi Round.
Utilizzando questa politica, tutte le variabili intere che compaiono in questo tipo di vincoli di uguaglianza, ottengono per ogni equazione un ub e un lb, direttamente influenzati dalla possibile escursione del singleton.
Come descritto in **Algoritmo 3** a seguito di un arrotondamento, l'algoritmo si deve anche preoccupare di eseguire le correzioni per i vincoli con slack artificiali, altrimenti rimarrebbero sfalsati per le prossime esecuzioni, la correzione si esegue semplicemente con:

$$x'_j = \frac{b - (a^T \cdot x - a_{ij} \cdot x_j)}{a_{ij}} \tag{3.7}$$

I singleton sono, tutto sommato, rari nei problemi di ottimizzazione, quindi in un caso d'uso reale, non ha molto senso preoccuparsi di verificare per ogni riga se ci sono più singleton e eventualmente utilizzarli tutti come slack artificiali.

3.4 Sorting delle Variabili

Un aspetto non considerato dall'autore dell'algoritmo è l'ordine con cui si approcciano le variabili nel ciclo interno, quindi nelle versioni finora considerate, l'attraversamento viene eseguito nell'ordine di definizione delle variabili del problema in input.
A seconda del tipo di problema e della sua struttura, l'ordine con cui si arrotondano le variabili potrebbe avere conseguenze più o meno rilevanti sull'output.
In questa sezione si presentano alcune delle possibili metodologie di sorting:

- **Integer infeasibility:** per una generica variabile intera $x_j \in G$, in questa argomentazione, viene espressa dalla quantità $ZI(x_j)$. Ordinare per questa quantità significa cercare di arrotondare prima le variabili *"meno intere"*, se l'ordine è discendente, le quali perturbano maggiormente i vincoli, nel momento in cui sono arrotondate. Viceversa per l'ordinamento ascendente.

- **Numero di coefficienti a_{ij} non zero:** pari al numero di vincoli in cui compaiono le variabili, rappresenta quanto una variabile è vincolata, oppure quanti vincoli vengono perturbati dalla modifica di una variabile. A seconda del problema può essere conveniente arrotondare prima le variabili meno vincolate, o viceversa.

- **Rilevanza nei vincoli:** il fatto che una variabile compaia in molti vincoli, non necessariamente significa che non sia modificabile, si definisca:

  $$s_{x_j} = min\{s_i,\ if\ a_{ij} \neq 0\}$$

  Usando s_{x_j} per ordinare le variabili $x_j \in G$, in senso ascendente, si discriminano le variabili in base all'appartenenza a vincoli rilevanti, cioé quei vincoli che hanno poco margine di modifica. Viceversa in senso discendente.

Tutti questi ordinamenti possono essere svolti nella fase di inzializzazione per ottenere un ordinamento statico, oppure, nel caso di **Integer infeasibility** e **Rilevanza nei vincoli**, nel ciclo esterno per un ordinamento dinamico.
Naturalmente questo aggiunge un overhead non indifferente, soprattutto nel caso dinamico, ma può aver senso per problemi non troppo grandi, in quanto si è visto migliorare i risultati di successo. in alcune istanze di test.
Per questo motivo, durante lo sviluppo è stato deciso di approfondire l'impatto del sorting delle variabili, i risultati sono commentati nella **parte III**.

Algoritmo 6 ZI Round con Sorting

INPUT: Soluzione x del rilassamento lineare, lista G delle variabili intere
Cerca i singleton tra i vincoli
for all Vincolo ($a^T \cdot x = b$) con singleton **do**
Sostituisci con $\begin{cases} a^T \cdot x \leq b \\ -a^T \cdot x \leq -b \end{cases}$
end for
if Ordinamento statico **then**
Ordina le variabili secondo il criterio scelto
end if
repeat
if Ordinamento dinamico **then**
Ordina le variabili secondo il criterio scelto
end if
for $x_j \in G$ *and* $ZI(x_j) \neq 0$ **do**
$Update\ slacks\ s = \{s_1, \ldots, s_m\}$ includendo anche quelli artificiali
$UB,\ LB = Calcola(threshold = \epsilon)$
Arrotondamento di x_j
Ripristina le variabili singleton per soddisfare i vincoli
end for
until Nessun aggiornamento possibile

Applicando il sorting in questo modo, non va cambiata la struttura dell'algoritmo, né vanno sostituite le strutture dati.

3.5 Ulteriori estensioni

In questa sezione si elencheranno tutte le restanti aggiunte e i tweaks applicati al codice rispetto alle versioni originali dell'algoritmo.
Un aspetto che non è possibile controllare nello sviluppo è quello di avere a disposizione le stesse soluzioni dei rilassamenti, utilizzate dall'ideatore dell'algoritmo, e il fatto che queste siano state ottenute più di 10 anni prima di questa implementazione abbassa le probabilità di ritrovarle uguali.
Per questo motivo non dovrebbe stupire che i risultati ottenuti in questa implementazione varino da quelli ottenuti originariamente da Wallace. Alcuni dei seguenti ritocchi sono serviti ad avvicinarsi ai risultati originali.

- **Soglia di correttezza vincolo:** valore ≥ 0 che rappresenta di quanto è concesso *"sbagliare"* quando è creato uno slack artificiale in un vincolo, dato che i calcoli con valori molto grandi o molto piccoli possono creare errori dati da arrotondamenti.

- **Margine di interezza:** valore $\in [0, 1)$, una variabile x_j è considerata intera o arrotondata se $ZI(x_j) \leq margine$. Utile nelle fasi di testing e debugging.

- **Opzioni slack artificiali:** gli slack artificiali possono essere trattati come descritto dal creatore dell'algoritmo, opppure con le seguenti opzioni implementate in questo approccio:

  - **Estesi a tutti i vincoli:** si è rivelato necessario per risolvere l'istanza di test "$nsrand - ipx$". La modalità di funzionamento rimane identica, naturalmente a seconda dei dati del problema, può incidere sui tempi di esecuzione.
  - **Spenti:** per problemi per cui non si desidera creare slack artificiali o dove non sono necessari, utile ad abbassare il tempo di esecuzione.

- **Modalità promiscua:** in questa modalità non si effettuano controlli sui limiti superiore e inferiore delle variabili usate come slack artificiali nella fase di ripristino. Sempre a causa delle approssimazioni eseguite dal calcolatore, nella fase di ripristino si possono introdurre errori (tipicamente piccoli, $< 10^{-11}$). Può essere utile se per per un dato problema si ritiene più prioritario il rispetto dei vincoli che il rispetto dei bounds delle variabili.
  Non incide sul successo dell'algoritmo.

Parte III

Risultati sperimentali

Capitolo 4

Risultati Sperimentali

4.1 Risultati

Come accennato nelle sezioni precedenti, il code che realizza l'algoritmo, è stato eseguito su 3 gruppi di istanze di problemi MIP:

- **Test Set:** Istanze della MIPLIB 2003 già risolte dall'implementazione originale (Wallace)
- **Benchmark 1:** Istanze della MIPLIB 2003 che non venivano risolte dall'implementazione originale
- **Benchmark 2:** Istanze di benchmark della MIPLIB 2010.

Per un totale di 136 istanze.
Tutte i set sono stati eseguiti con le seguenti impostazioni:

- La ricerca degli slack artificiali è stata abilitata per le sole istanze contenenti uguaglianze e solo per esse.
- Modalità promiscua disattivata
- Tolleranze su vincoli e variabili pari a quelle di default su cplex 12.10
- Threshold per il calcolo di UB e LB pari a 10^{-5}, come in origine.

Come è ragionevole aspettarsi, questa implementazione di Zi Round, non riproduce esattamente i risultati dell'implementazione originale, ma confrontando i risultati del test set, ci si accorge che per le istanze risolte, i valori di funzione obiettivo sono comunque molto vicini.

Ciò si spiega considerando che le soluzioni dei rilassamento ottenuti da IBM cplex 12.10 possono differire per i valori delle variabili, rispetto a quelli ottenuti da Wallace nel 2009 principalmente per le differenze dei risolutori: il risolutore utilizzato da Wallace era CPLEX 11.00, più vecchio di un decennio rispetto a cplex 12.10.

Le istanze non risolte da questa implementazione hanno tutte quante dei vincoli di uguaglianza che legano le variabili con vincolo di interezza e sono:

- **timtab1 e timtab2:** la soluzione ottenuta da cplex per i rilassamenti non permette di creare slack artificiali nei vincoli, tali da poter arrotondare tutte le variabili.
- **mkc:** Contiene due soli vincoli di uguaglianza più altre disequazioni, non è possibile creare slack artificiali per entrambe le equazioni.
- **sp97ar:** Non viene risolta da ZI Round base (**Algoritmo 3**), ma si trova una soluzione applicando il sorting per integer infeasibility ascendente.

Confronto con i risultati di Wallace (valori funzione obiettivo):				
Nome istanza	I.I. asc. N.D.	Nessun sort	Ref. Orig.	Soluzione ottima
cap6000	-2442801	-2442801	-2442801	-2451377
fast0507	326	326	353	174
fixnet6	12170	12170	4536	3983
manna81	-13155	-13155	-12880	-13164
markshare1	518	230	230	1
markshare2	562	454	674	1
mkc	n.d.	n.d.	-26.99	-563.846
mod011	-42783986.67	-42783986.67	125430296	-54558500.0
modglob	20786787	20786787	20786788	20740500
nsrand-ipx	646400	646400	81120	51200
pp08a	15000	15000	14300	7350
pp08aCUTS	16630.4	16630.4	16630.4	7350
qiu	1805.18	1805.18	1805.18	-132.873
set1ch	110241.5	110241.5	107691.50	54537.8
seymour	607	583	595	423
sp97ar	1027302955.89	n.d.	1094926720	660706000
timtab1	n.d.	n.d.	1719551	764772
timtab2	n.d.	n.d.	2449798	1096557

Legenda:
I.I. = Integer Infeasibility
#Vincoli = numero di vincoli (=numero di coefficienti non zero nei vincoli)
R. = rilevanza nei vincoli
asc. / desc.: ascendente / discendente
D. / N.D.: dinamico / non dinamico

I 3 set vengono poi eseguiti per ogni modalità di ordinamento descritta in **3.4**, i risultati ottenuti sono poi presentati utilizzando i valori di percentuale di successo, primal gap percentuale e tempo di esecuzione, calcolati come segue:

- **Media geometrica shiftata (tempi di esecuzione)**
  Considerando il vettore dei tempi di esecuzione $x = (x_1, \ldots, x_N)$.

$$G(x, a) = \sqrt[N]{(x_1 + a) \cdot \cdots \cdot (x_N + a)} - a \tag{4.1}$$

  I risultati presentati sono calcolati con $a = 1s$.

- **Primal gap:** [1]
  Si definiscano:

  - o_r**:** valore della funzione obiettivo per la soluzione "arrotondata"
  - o_b**:** valore della funzione obiettivo per la soluzione ottima, ottenuto dagli archivi

  Si utilizza la seguente formula:

$$gap(o_r, o_b) = \begin{cases} 0, & if\ |o_b| = |o_r| = 0 \\ 1, & if\ o_b \cdot o_r < 0 \\ \frac{|o_b - o_r|}{max\{|o_b|,\ |o_r|\}}, & else \end{cases} \tag{4.2}$$

NOTA: tutte le istanze sono problemi di minimizzazione.
Il primal gap del riferimento (risultati di Wallace) è calcolato anche senza i risultati delle istanze timtab* e mkc, ma includendo l'istanza sp97ar, per poter confrontare questa implementazione.
La media g.s. dei tempi di esecuzione è calcolata sia per l'intera esecuzioni su tutte le istanze dei set, che per le sole istanze che vengono risolte. In questo modo si sono prodotte 4 misure per ogni modalità di esecuzione, che sintetizzano i rinsultati delle singole istanze, utili per i confronti.

Per ogni istanza è stato impostato un time limit di 5 minuti per la risoluzione del rilassamento lineare e attrettanti 5 minuti per l'esecuzione dell'algoritmo, nessuna istanza è andata in timeout.

Esecuzione su Test Set				
Modalità	Successo (%)	Primal gap (% media)	Tempo medio successi (s) [*]	Tempo medio sul set (s) [*]
Nessun sort	77.78	50.88	0.20	0.22
I.I. desc. N.D.	77.78	50.07	0.18	0.21
I.I. desc. D.	77.78	50.07	0.20	0.24
I.I. asc. N.D.	83.33	50.08	0.21	0.18
I.I. asc. D.	83.33	49.99	0.24	0.21
#Vincoli desc.	77.78	50.62	0.22	0.23
#Vincoli asc.	77.78	51.38	0.21	0.22
R. desc. N.D.	77.78	50.88	0.24	0.26
R. desc. D.	77.78	50.88	0.27	0.29
R. asc. N.D.	77.78	50.88	0.24	0.25
R. asc. D.	77.78	50.88	0.27	0.29
Ref. Origin.	100	48.28	//	//
(tutte le istanze)	100	51.68	//	//

[*] : calcolata come media geometrica shiftata, definita a inizio sezione

Tenendo presente le considerazioni presentate poc'anzi, si può affermare che il codice implementi con sufficiente fedeltà l'algoritmo in analisi.
In più i risultati mostrano che che l'applicazione degli ordinamenti non ha particolari effetti negativi sui tempi di esecuzione, mentre "I.I. asc." permette di risolvere una istanza in più (sp97ar) rispetto all'implementazione senza sorting.
È inoltre degno di nota il fatto che, nelle modalità per cui non cambia la percentuale di successo, il primal gap percentuale migliori o rimanga inalterato rispetto all'esecuzione base.

Di seguito sono riportati i risultati ottenuti dall'esecuzione del codice sui due benchmark

Esecuzione su Benchmark 1 (42 istanze)				
Modalità	Successo (%)	Primal gap (% media)	Tempo medio successi (s) [*]	Tempo medio sul set (s) [*]
Nessun sort	2.38	$2.4 \cdot 10^{-5}$	$8 \cdot 10^{-3}$	0.85
I.I. desc. N.D.	4.76	48.68	3.44	0.46
I.I. desc. D.	2.38	$2.4 \cdot 10^{-5}$	$8 \cdot 10^{-4}$	0.84
I.I. asc. N.D.	4.76	48.70	3.46	0.47
I.I. asc. D.	2.38	$2.4 \cdot 10^{-5}$	$8 \cdot 10^{-4}$	0.84
#Vincoli desc.	2.38	$2.4 \cdot 10^{-5}$	$8 \cdot 10^{-4}$	0.87
#Vincoli asc.	2.38	$2.4 \cdot 10^{-5}$	$9 \cdot 10^{-4}$	0.85
R. desc. N.D.	2.38	$2.4 \cdot 10^{-5}$	$2 \cdot 10^{-3}$	0.92
R. desc. D.	2.38	$2.4 \cdot 10^{-5}$	$3 \cdot 10^{-3}$	1.04
R. asc. N.D.	2.38	$2.4 \cdot 10^{-5}$	$2 \cdot 10^{-3}$	0.92
R. asc. D.	2.38	$2.4 \cdot 10^{-5}$	$4 \cdot 10^{-3}$	1.05

Delle 42 istanze vengono risolte solo "noswot" e "ds".
L'istanza "noswot" viene risolta da tutte le modalità senza differenze apprezzabili.

Esecuzione su Benchmark 2 (76 istanze)				
Modalità	Successo (%)	Primal gap (% media)	Tempo medio successi (s) [*]	Tempo medio sul set (s) [*]
Nessun sort	21.05	57.47	0.76	0.94
I.I. desc. N.D.	23.68	55.99	0.58	0.74
I.I. desc. D.	21.05	54.24	0.77	0.96
I.I. asc. N.D.	25.00	55.84	0.60	0.73
I.I. asc. D.	22.37	53.91	0.79	0.95
#Vincoli desc.	22.37	53.22	0.83	0.97
#Vincoli asc.	23.68	53.49	0.88	0.98
R. desc. N.D.	21.05	57.47	0.84	1.13
R. desc. D.	21.05	57.47	0.87	1.26
R. asc. N.D.	21.05	57.47	0.84	1.19
R. asc. D.	21.05	57.47	0.87	1.25

4.2 Commenti ai risultati

Dai risultati si possono trarre alcune considerazioni:

- **ZI Round** risulta applicabile anche a distanza di 11 anni dalla sua creazione. Sono ancora valide le considerazioni di Wallace: l'euristico in media risulta sufficientemente veloce da poter essere applicato in fase di risoluzione di una istanza MIP.
  Se da una parte, con le soluzioni dei rilassamenti lineari attualmente prodotte dal risolutore, ZI Round perde qualche istanza della miplib2003 che originariamente risolveva, dall'altra, ne guadagna due nuove.

- Complessivamente, l'applicazione del **sorting delle variabili** non ha particolari effetti collaterali sull'euristico, se non l'aggiunta di un generale overhead, infatti non determina nessun calo di successo.

- Risulta evidente che, nel complesso, **l'ordinamento più utile** per questo euristico sia "I.I. asc. N.D.", poiché non solo fa raggiungere, in ogni set, la percentuale di successo massima, ma riduce anche il tempo di esecuzione medio (sul set), rispetto all'esecuzione base. Questo si spiega con il fatto che visitando le variabili in questo ordine, si riduca il numero di iterazioni del ciclo più esterno.
  Inoltre è degno di nota che questo ordinamento migliori anche il primal gap. Non è da escludere che questa tecnica non sia efficace anche per altri euristici di arrotondamento.

- **L'ordinamento meno efficace in assoluto** è "R.", che produce le stesse soluzioni dell'esecuzione senza sorting, ma più lentamente. Ciò è visibile in entrambe le colonne dei tempi di esecuzione, in tutte le tabelle.
  Si può ipotizzare che questo ordinamento NON sia in grado di discriminare le variabili efficacemente e pertanto non cambi in maniera sostanziale l'ordine rispetto a quello definito nelle istanze.

4.3 Conclusioni

L'approccio presentato implementa con sufficiente fedeltà e affidabilità l'algoritmo in esame. È stata inoltre presentata un'estensione che determinana un miglioramento complessivo delle performance, sia dal punto di vista della qualità della soluzione, che per la velocità di esecuzione.
Si rinnovano le conclusioni tratte da Wallace nel 2009, l'euristico potrebbe certamente essere utile a un risolutore MIP, in particolare nella sua versione estesa presentata in questa analisi.

Parte IV

Appendici

Appendice A

Risorse

A.1 Documentazione IBM CPLEX

Per l'implementazione dell'algoritmo è stata utilizzata la versione 12.10 *Academic*

A.2 Moduli python

- **Math**
- **Numpy**
- **Datetime**
- **Time**
- **Signal**
- **OS**
- **Decimal**

A.3 MIPLIB

- **MIPLIB 2003**
- **MIPLIB 2010**

Bibliografia

[1] T. Berthold, *Measuring the Impact of Primal Heuristics*. Zuse Institute Berlin, 2012, riferimento per calcoli dei risultati sperimentali.

[2] C. Wallace, "ZI round, a MIP rounding heuristic," *Journal of Heuristics*, 2009, principale riferimento.

[3] M. Fischetti, *Lezioni di Ricerca Operativa*. Independently published, 2018, riferimenti MIP/LP.

[4] Wikipedia, "Heuristic (computer science)," 2020, approfondimenti e curiosità.

[5] Python, *Style Guide for Python Code*. PEP 8: Python.org, 2020, guida allo stile di scrittura per codice Python.