UNIVERSITÀ DEGLI STUDI DI MILANO
Facoltà di Scienze Matematiche Fisiche e Naturali
Dipartimento di Tecnologie dell'Informazione

Algoritmi di programmazione matematica per un problema di classificazione di testi

RELATORE

Prof. Giovanni RIGHINI

CORRELATORE

Prof. Anna CORAZZA

Tesi di Laurea di:
Sandro BOSIO
Matricola 587180

Anno Accademico 2001-2002

Indice

1 Introduzione **3**

1.1 Text Covering with Strings Subset (TCSS) 4
1.2 Modello di Programmazione Lineare Intera 5
1.3 Sottoproblema Max Path Problem (MPP) 6
1.3.1 Algoritmo di Programmazione Dinamica per la soluzione di MPP 6
1.4 Dati utilizzati 8
1.5 Strumenti utilizzati 9
1.6 Metodi applicati 9

2 Utilizzo di un *solver* generico **10**

2.1 Riduzione del problema 10
2.1.1 Eliminazione di vincoli 10
2.1.2 Eliminazione di variabili 11
2.2 Formulazioni equivalenti per il problema intero 12
2.3 Rilassamento continuo 13
2.4 Risultati 14

3 Riformulazione di Benders **16**

3.1 Punti estremi e raggi estremi 16
3.2 Introduzione alla riformulazione di Benders 17
3.3 Riformulazione di Benders 17
3.4 Rilassamento della riformulazione di Benders 19
3.5 Applicazione a TCSS 21
3.5.1 Riformulazione A 21
3.5.2 Riformulazione B 22
3.6 Risultati e considerazioni 24

4 Rilassamento Lagrangeano **26**

4.1 Introduzione al rilassamento Lagrangeano 26
4.2 Metodo del Sottogradiente 27
4.3 Applicazione al problema 28
4.4 Propagazione dei vincoli e riduzione 28

4.5 Rilassamento 1 29
4.5.1 Euristica Lagrangeana 30
4.5.2 Riduzione del problema 30
4.5.3 Strategia di *branching* 30
4.6 Rilassamento 2 31
4.6.1 Euristica Lagrangeana 32
4.6.2 Riduzione del problema 33
4.6.3 Strategia di *branching* 33
4.7 Risultati 34

5 Algoritmi di ricerca locale **35**
5.1 Strategie di ricerca locale 35
5.2 *Threshold Accepting* 36
5.2.1 Applicazione di *Threshold Accepting* a TCSS 36
5.2.2 Risultati 38
5.3 *Simulated Annealing* 41
5.3.1 Applicazione di *Simulated Annealing* a TCSS 41
5.3.2 Taratura dei parametri 43
5.3.3 Risultati 44

6 Confronto dei diversi metodi **46**
6.1 Confronto su un'istanza di classe B 46
6.2 Confronto su un'istanza di classe C 48

7 Sviluppi futuri **49**

Capitolo 1

Introduzione

La presenza di una grande quantità di documenti digitali non catalogati ha portato allo sviluppo di strumenti automatici di classificazione (vedere [1] per un confronto tra i principali approcci al problema di *Text Categorization*). Passo preliminare alla classificazione di un testo è la sua segmentazione in unità fondamentali; generalmente tale operazione viene eseguita secondo due modalità:

- segmentare in parole, dove una parola è un insieme di caratteri alfabetici non separati da nessun segno di interpunzione. Questo approccio è noto come *bag-of-words*.
- segmentare basandosi su un'analisi morfologica del testo.

La prima modalità è a basso costo, ma poiché una parola normalmente è presente in un testo in diverse forme, porta a un problema di sparsità di dati, con il rischio che diverse forme di una parola contribuiscano in modi diversi, eventualmente contrastanti, alla classificazione del testo.

La seconda modalità è senz'altro più accurata, ma necessita di un'analisi morfologica della lingua a cui si vuole applicarla; quindi porta a significativi costi aggiuntivi in caso di *porting* del classificatore verso nuove lingue.

Un nuovo approccio alla classificazione si basa su un diverso metodo di segmentazione del testo, che cerca di ovviare al problema delle diverse forme corrispondenti allo stesso lemma senza i costi dell'analisi linguistica. Tale metodo consiste nella costruzione dell'albero dei suffissi (vedere [2] per un algoritmo di costruzione in tempo lineare del *Suffix Tree* di un testo) di un testo di *training* e nella sua successiva riduzione. Una prima riduzione consiste nell'eliminare le stringhe più lunghe di un valore massimo e più corte di un valore minimo, e le stringhe che compaiono più raramente nel testo.

Il problema affrontato in questa tesi riguarda l'ottimizzazione di un'ulteriore riduzione dell'albero dei suffissi in modo che esso mantenga buone caratteristiche di copertura sul testo di *training*: si vuole cioè trovare la copertura massima di un testo utilizzando un sottoinsieme minimo delle stringhe dell'albero dei suffissi.

Una volta ridotto l'albero dei suffissi, viene costruito un modello di Markov nascosto (HMM, Hidden Markov Model) con il quale si possono segmentare testi nella stessa lingua del testo di *training*. Questa struttura rende l'approccio indipendente dalla lingua, poiché non ci sono regole specifiche di una particolare lingua né nella costruzione dell'albero né nella sua riduzione. Si prevede che il suo utilizzo possa esse-

re particolarmente utile nel caso di lingue fortemente inflesse, quale è il caso ad esempio dell'italiano e dell'arabo.

1.1 Text Covering with Strings Subset (TCSS)

I dati del problema sono i seguenti:

- un testo W composto da una sequenza finita di caratteri di un alfabeto Σ.
- un sottoinsieme S delle stringhe di W.

Ogni stringa può avere diverse occorrenze in diversi punti del testo. Si definisca quindi:

- l'insieme O delle occorrenze delle stringhe di S nel testo W.

Problema: trovare un sottoinsieme $Y \subseteq S$ ed un sottoinsieme $X \subseteq O$ tali che:

- le occorrenze in X non si sovrappongano.
- le occorrenze in X siano occorrenze delle stringhe in Y.

e che soddisfino i seguenti requisiti di ottimalità:

- la funzione $f(X)$, numero di caratteri di W coperti dalle occorrenze in X, sia massimizzata.
- una funzione di costo $g(Y)$ sia minimizzata.

La funzione di costo g deve tenere conto della lunghezza delle stringhe: poiché si suppone che stringhe più lunghe abbiano più significato ai fini della classificazione, si è scelto di dare come peso ad una stringa l'inverso della sua lunghezza.

La prima funzione obiettivo richiede che il testo sia coperto al massimo, mentre la seconda richiede che si usino poche stringhe e di preferenza lunghe. I due obiettivi sono evidentemente in conflitto tra loro, dal momento che per coprire più testo occorre usare un numero di stringhe maggiore, comprese quelle più corte. Non esistendo un modo esatto per pesare i due obiettivi, si è scelto di combinarli utilizzando un parametro $\alpha \in (0, 1)$, ottenendo la funzione obiettivo:

$$\max z = \alpha f(X) - (1 - \alpha) g(Y)$$

e sviluppando i diversi metodi risolutivi come *subroutine* che ricevono il parametro α in input. Una volta inserito l'algoritmo all'interno del programma di segmentazione, il valore di α deve essere tarato in base ai risultati di classificazione.

1.2 Modello di Programmazione Lineare Intera

Un modello di Programmazione Lineare Intera per questo problema è:

$$\text{TCSS}) \qquad \max z = \alpha \sum_{j=1}^{|O|} l_{u(j)} x_j - (1-\alpha) \sum_{i=1}^{|S|} \frac{1}{l_i} y_i$$

$$s.t. \begin{cases} \sum_{j=1}^{|O|} a_{tj} x_j \leqslant 1 & t = 1 \dots |W| \quad (1.1) \\ x_j - y_{u(j)} \leqslant 0 & j = 1 \dots |O| \quad (1.2) \\ x_j \in \{0,1\} & j = 1 \dots |O| \\ y_i \in \{0,1\} & i = 1 \dots |S| \end{cases}$$

dove:

- y_i è la variabile binaria corrispondente alla stringa $i = 1 \dots |S|$.
- l_i è la lunghezza della stringa i.
- x_j è la variabile binaria corrispondente all'occorrenza $j = 1 \dots |O|$.
- $u(j)$ è la stringa che genera l'occorrenza j $(u : O \to S)$.
- a_{tj} è una matrice binaria che indica se il carattere t del testo ($t = 1 \dots |W|$) può essere coperto dall'occorrenza j.

Il vincolo (1.1) impone che le occorrenze non possano sovrapporsi: per ogni carattere t del testo non può essere scelta più di un'occorrenza j che copre tale carattere.

Il vincolo (1.2) impone che se è stata scelta l'occorrenza j, ovvero se $x_j = 1$, allora sia necessario scegliere la rispettiva stringa $u(j)$, ovvero deve essere $y_{u(j)} = 1$.

Altre definizioni usate nel seguito sono:

- $O_S(i)$ è l'insieme delle occorrenze della stringa i.
- $start(j)$ è il primo carattere coperto dall'occorrenza j.

1.3 Sottoproblema Max Path Problem (MPP)

Per ogni scelta delle variabili y relative alle stringhe si ottiene il sottoproblema:

$$\text{MPP}) \qquad \max z(y) = \alpha \sum_{j=1}^{|O|} l_{u(j)} x_j$$

$$s.t. \begin{cases} \sum_{j=1}^{|O|} a_{tj} x_j \leqslant 1 & t = 1 \dots |W| \\ x_j \leqslant y_{u(j)} & j = 1 \dots |O| \\ x_j \in \{0, 1\} & j = 1 \dots |O| \end{cases}$$

Questo sottoproblema consiste nell'ottimizzare la copertura del testo con occorrenze delle sole stringhe disponibili. Il parametro α non incide sulla soluzione ottima ma solo sul suo valore, quindi può essere eliminato dalla formulazione.

Grazie alla particolare struttura della matrice a_{tj}, indotta dal fatto che ogni occorrenza copre solo caratteri adiacenti, è possibile risolvere questo problema in tempo polinomiale con un algoritmo di Programmazione Dinamica.

1.3.1 Algoritmo di Programmazione Dinamica per la soluzione di MPP

Il testo corrisponde ad un grafo con $|W| + 1$ nodi e $|O| + |W|$ archi, in cui ogni carattere t è un nodo e ogni occorrenza j un arco di valore pari al numero di caratteri coperti; il j-esimo arco esce dal nodo corrispondente al carattere $start(j)$ ed entra nel nodo corrispondente al carattere $(start(j) + l_{u(j)})$. Oltre a questi archi vanno aggiunti $|W|$ archi di valore nullo da ogni carattere t al carattere successivo $t + 1$, che rappresentano la scelta di non coprire il carattere t; in questo modo esiste almeno un percorso da ogni carattere verso qualsiasi carattere a lui successivo. È necessario un nodo finale in cui entra l'arco corrispondente alla mancata copertura dell'ultimo carattere ed eventuali archi corrispondenti a occorrenze che coprono anche l'ultimo carattere.

Si tratta di un grafo orientato aciclico, quindi il problema di cammino massimo dal nodo sorgente (il primo carattere) al nodo destinazione (il nodo fittizio aggiunto) si può risolvere con complessità lineare nel numero di archi. Il valore della copertura massima è dato dall'etichetta del nodo destinazione.

Se oltre al valore ottimo si è interessati anche al valore delle variabili x_j nella soluzione ottima, occorre mantenere un vettore per indicare per ogni carattere quale occorrenza è stata utilizzata per raggiungerlo. Percorrendo a ritroso il cammino dal nodo destinazione al nodo sorgente si può ricavare quali siano le occorrenze scelte nella soluzione ottima.

Lo pseudocodice per questo algoritmo è il seguente:

```
function SolveMPP( Y )
    for ⟨ t := 1...|W|+1 ⟩
        label[t] := 0
    for ⟨ t := 1...|W| ⟩
        // ARCO DI LUNGHEZZA ZERO
        label[t+1] := max (label[t+1], label[t])
        // ARCHI USCENTI DAL NODO t
        for ⟨ j | start(j) = t,  u(j) ∈ Y ⟩
            label[t+l_j] := max (label[t+l_j], label[t]+l_j)
    return label[|W|+1]
```

Dal momento che l'efficienza di questo algoritmo è critica per l'efficienza degli algoritmi in cui viene utilizzato come *subroutine*, e dal momento che spesso nei casi in cui viene utilizzato un'istanza differisce dalla precedente per il valore di una sola variabile y_i, ho sviluppato una versione più efficiente nel caso in cui una stringa venga inserita nell'insieme delle stringhe scelte, ovvero in cui una variabile da $y_i = 0$ diventi $y_i = 1$.

Lo pseudocodice per l'algoritmo che risolve questo caso particolare è il seguente:

```
function UpdateMPP(Y, i, label[ ])
    for ⟨ t := 1...|W|+1 ⟩
        improving[t] := false
    for ⟨ j | u(j) = i ⟩
        improving[start(j)] := true
    for ⟨ t := 1...|W| ⟩
        if ( improving[t] )
            // ARCO DI LUNGHEZZA ZERO
            if ( label[t+1] < label[t] )
                label[t+1] := label[t]
                improving[t+1] := true
            // ARCHI USCENTI DAL NODO t
            for ⟨ j | start(j) = t,  u(j) ∈ Y ⟩
                if ( label[t+l_j] < label[t]+l_j )
                    label[t+l_j] := label[t]+l_j
                    improving[t+l_j] := true
    return label[|W|+1]
```

Il valore delle etichette si basa sul valore calcolato al passo precedente. Viene mantenuto un vettore di variabili booleane, una per ogni nodo, che indicano se da quel nodo esiste la possibilità di migliorare l'etichetta dei nodi a lui adiacenti; questa possibilità esiste, per il nodo t, se uno dei suoi archi uscenti corrisponde ad un'occorrenza della stringa aggiunta alla soluzione, o se la sua etichetta è stata aumentata. In questo modo il tempo necessario al calcolo del cammino ottimo viene ridotto in media del 20%.

1.4 Dati utilizzati

Il *data set* utilizzato è "20 Newsgroups", una raccolta di messaggi di posta elettronica in lingua inglese suddivisi in venti diverse aree tematiche. La versione usata è la versione modificata "20news18828", disponibile all'indirizzo: *http://www.ai.mit.edu/people/jrennie/20_newsgroups*.

La costruzione del testo di *training* avviene nei seguenti passi:

1. Viene costruito un testo prendendo a caso 40 messaggi per ogni *newsgroup* e concatenandoli.
2. Vengono eliminati tutti gli *header* dei messaggi di posta.
3. Tutti i caratteri maiuscoli vengono trasformati in minuscoli.
4. Tutti i caratteri non alfabetici (numeri inclusi) vengono trasformati in un carattere $\emptyset$.
5. Ogni sequenza di $\emptyset$ adiacenti viene ridotta ad un unico carattere $\emptyset$.
6. Le linee del testo vengono permutate in modo casuale.

Dal testo che si ottiene vengono quindi prodotti testi più piccoli, per avere istanze di dimensione diversa, prendendone le prime n linee. Il numero di linee prese identifica la classe dell'istanza:

1. Classe A: 200 linee; viene utilizzata per la verifica della correttezza dei metodi sviluppati.
2. Classe B: 1000 linee; viene utilizzata per i test su ogni singolo metodo, per verificare quale sia la migliore scelta dei suoi parametri.
3. Classe C: 5000 linee; viene utilizzata per i confronti tra i diversi metodi.

Per ognuno di questi testi viene costruito l'albero dei suffissi, su cui avviene una prima riduzione, basata su tre parametri:

- un numero minimo di occorrenze della stringa nel testo C_{MIN}.
- una lunghezza minima della stringa L_{MIN}.
- una lunghezza massima della stringa L_{MAX}.

Le stringhe residue, con le loro occorrenze, sono l'input di TCSS. Il numero di stringhe, di occorrenze e di caratteri ottenuti per ogni classe è il seguente:

CLASSE	$\|S\|$	$\|O\|$	$\|W\|$
A	170	2850	8870
B	2200	55700	49000
C	9950	375000	218000

1.5 Strumenti utilizzati

Il sistema operativo utilizzato per lo sviluppo ed i test è Linux, distribuzione Red Hat 7.3. Gli algoritmi sono stati sviluppati in C++, secondo il paradigma della programmazione orientata agli oggetti, ed il compilatore utilizzato è "g++" versione 2.96, con opzioni di compilazione standard. Il *solver* generico utilizzato è CPLEX versione 6.5, di ILOG. Il computer utilizzato per i test è un Intel PIV 1.60 Ghz, con 640MB di memoria.

1.6 Metodi applicati

Nel capitolo 2 viene descritta l'applicazione al problema di un *solver* generico per la soluzione esatta del problema; questo metodo risulta molto efficace per istanze di classe B, per cui riesce a fornire entro alcuni minuti di calcolo un *upper bound* molto stringente ed in alcune ore la soluzione ottima, ma fallisce per problemi di tempo e di memoria su istanze di dimensione maggiore, non riuscendo a calcolare nemmeno il valore ottimo del rilassamento continuo nel nodo radice.

Nel capitolo 3 viene mostrata l'applicazione al problema della *riformulazione di Benders* con un algoritmo *cutting planes*. TCSS viene riformulato, in due diversi modi, in un problema intero *master* e in un sottoproblema continuo, la cui risoluzione produce tagli validi per il problema intero.

Nel capitolo 4 viene applicato al problema il metodo del *rilassamento Lagrangeano*, sviluppando due diverse riformulazioni.

Le istanze di interesse per il problema di classificazione risultano di dimensioni troppo elevate per i metodi esatti; pertanto nel capitolo 5 vengono sviluppati due algoritmi di ricerca locale per la soluzione approssimata del problema, basati su *Threshold Accepting* e *Simulated Annealing*, che non forniscono garanzie di ottimalità ma in tempi brevi riescono a fornire buone soluzioni.

Nel capitolo 6 i diversi metodi vengono messi a confronto su un'istanza di classe C.

Nel capitolo 7 vengono proposti dei possibili sviluppi del lavoro svolto nell'ambito di questa tesi.

Capitolo 2

Utilizzo di un *solver* generico

CPLEX è un solutore generico di problemi lineari, continui o misti interi, prodotto da ILOG. È disponibile sia come programma interattivo che come librerie utilizzabili all'interno di un programma.

Si basa su un *branch and bound* in cui l'*upper bound* viene calcolato tramite rilassamento continuo con il metodo del simplesso o del simplesso duale, con euristiche primali per ottenere soluzioni intere ammissibili.

2.1 Riduzione del problema

Con una preelaborazione dei dati in input al problema è possibile cercare di ridurne le dimensioni, eliminando tutti i vincoli ridondanti e tutte le variabili il cui valore nella soluzione ottima sia predicibile. Queste operazioni, utilizzando un solutore generico come CPLEX, sono particolarmente importanti perché permettono di ridurre la quantità di memoria utilizzata per descrivere il problema e di ridurre il tempo di calcolo, con il risultato finale di poter risolvere istanze di dimensione maggiore.

2.1.1 Eliminazione di vincoli

Si definisca

$$O_T(t) = \{j \in O \mid a_{tj} = 1\}$$

l'insieme delle occorrenze *attive* sul carattere t, ovvero che possono coprirlo.

Se $O_T(t_1) \subseteq O_T(t_2)$, ovvero se tutte le occorrenze attive sul carattere t_1 sono attive anche sul carattere t_2, il vincolo di non sovrapposizione sul carattere t_2 domina l'analogo vincolo sul carattere t_1, che quindi è ridondante e può essere eliminato dalla formulazione.

Se inoltre su un carattere sono attive meno di due occorrenze, il vincolo relativo risulta superfluo e si può eliminare.

Poiché le occorrenze coprono solo caratteri adiacenti, per decidere se il vincolo relativo al carattere t è dominato non è necessario confrontarlo con ogni altro vincolo, ma basta confrontarlo con il vincoli relativi ai caratteri $t + 1$ e $t - 1$.

Il seguente algoritmo, in forma di pseudocodice, elimina i vincoli ridondanti:

V := $\{\}$
k := 1
for $\langle$ $t \in \{2 \dots |W|\}$ $\rangle$
    if ($O_T(k) \subseteq O_T(t)$)
        // IL VINCOLO k È RIDONDANTE, NON VIENE CONSIDERATO
        k := t
    else
        if ($O_T(t) \subseteq O_T(k)$)
            // IL VINCOLO t È RIDONDANTE, NON VIENE CONSIDERATO
        else
            // I DUE VINCOLI SONO ALMENO PARZIALMENTE DISGIUNTI
            if ($|O_T(k)| > 1$)
                V := $V \cup \{k\}$
            k := t
V := $V \cup \{k\}$

2.1.2 Eliminazione di variabili

Se un vincolo relativo ad un carattere coperto da una sola occorrenza j, e quindi eliminabile, risulta anche non dominato da nessun'altro vincolo di copertura, significa che tale occorrenza non è sovrapposta a nessun'altra occorrenza, su nessun carattere. In questo caso è possibile eliminare, oltre al vincolo, anche la variabile corrispondente all'occorrenza, aggiungendo il suo coefficiente nella funzione obiettivo $\alpha l_{u(j)}$ al coefficiente nella funzione obiettivo della stringa $u(j)$.

Esiste un certo (piccolo) numero di occorrenze eliminate nelle istanze utilizzate. A seguito di queste eliminazioni, che modificano i coefficienti nella funzione obiettivo delle variabili associate alle stringhe, può verificarsi uno dei seguenti casi:

- se esiste una stringa senza più occorrenze e la sua variabile ha coefficiente negativo, è possibile fissarla a 0.
- se esiste una stringa la cui variabile ha coefficiente positivo è possibile fissarla ad 1, poiché in ogni caso conviene sceglierla.

Può essere fatta un'ulteriore riduzione dipendente da α. Il valore massimo di una stringa è dato da:

$$\alpha l_i |O_S(i)| - \frac{1-\alpha}{l_i}$$

ovvero il valore di tutte le sue occorrenze meno il costo della stringa. Questo valore può diventare negativo, ed in tal caso è possibile eliminare la stringa e tutte le sue occorrenze dalla formulazione perché in nessun caso verrà scelta. Questo avviene per

$$|O_S(i)| \leqslant \frac{1-\alpha}{\alpha l_i^2}$$

Ad esempio, per $\alpha = 0.01$ una stringa di lunghezza 3 deve avere almeno undici occorrenze per non essere eliminata.

2.2 Formulazioni equivalenti per il problema intero

Oltre alla formulazione originale, che è un problema intero puro, sono stati provati alcuni rilassamenti equivalenti; questo perché se un problema è meno vincolato, o se ha delle variabili continue, la risoluzione tramite CPLEX risulta facilitata.

Rilassamento 1. Il primo rilassamento si ottiene rilassando il vincolo di integralità sulle variabili y.

$$\text{TCSS}_1\ ) \qquad \max z = \alpha \sum_{j=1}^{|O|} l_{u(j)} x_j - (1-\alpha) \sum_{i=1}^{|S|} \tfrac{1}{l_i} y_i$$

$$s.t. \begin{cases} \sum_{j=1}^{|O|} a_{tj} x_j \leqslant 1 & t = 1 \dots |W| \\ x_j - y_{u(j)} \leqslant 0 & j = 1 \dots |O| \\ x_j \in \{0,1\} & j = 1 \dots |O| \\ y_i \in [0,1] & i = 1 \dots |S| \end{cases}$$

Nella soluzione ottima (x^*, y^*) non si può avere $y_i^* \in (0,1)$ per una stringa i, poiché ponendo $y_i^* = 0$ si otterrebbe una soluzione ammissibile di valore superiore. Quindi la soluzione ottima di questo problema deve essere intera.

Rilassamento 2. Il secondo rilassamento si ottiene rilassando il vincolo di integralità sulle variabili x.

$$\text{TCSS}_2\ ) \qquad \max z = \alpha \sum_{j=1}^{|O|} l_{u(j)} x_j - (1-\alpha) \sum_{i=1}^{|S|} \tfrac{1}{l_i} y_i$$

$$s.t. \begin{cases} \sum_{j=1}^{|O|} a_{tj} x_j \leqslant 1 & t = 1 \dots |W| \\ x_j - y_{u(j)} \leqslant 0 & j = 1 \dots |O| \\ x_j \in [0,1] & j = 1 \dots |O| \\ y_i \in \{0,1\} & i = 1 \dots |S| \end{cases}$$

Tale formulazione risulta equivalente poiché il problema ottenuto fissando le variabili y è il rilassamento continuo di MPP illustrato nel capitolo 1.3, la cui soluzione ottima è intera.

Rilassamento 3. Il terzo rilassamento consiste nel surrogare i vincoli (1.2) con un unico vincolo (2.1) per ogni stringa.

$$\text{TCSS}_3\ \Big) \qquad \max z = \alpha \sum_{j=1}^{|O|} l_{u(j)} x_j - (1-\alpha) \sum_{i=1}^{|S|} \tfrac{1}{l_i} y_i$$

$$s.t. \begin{cases} \sum_{j=1}^{|O|} a_{tj} x_j \leqslant 1 & t = 1 \dots |W| \\ \sum_{j \in O_S(i)} x_j \leqslant y_i |O_S(i)| & i = 1 \dots |S| \\ x_j \in [0,1] & j = 1 \dots |O| \\ y_i \in \{0,1\} & i = 1 \dots |S| \end{cases} \tag{2.1}$$

Come nel precedente rilassamento le variabili x relative alle occorrenze sono continue. Tale formulazione risulta equivalente per gli stessi motivi per cui lo è quella precedente.

2.3 Rilassamento continuo

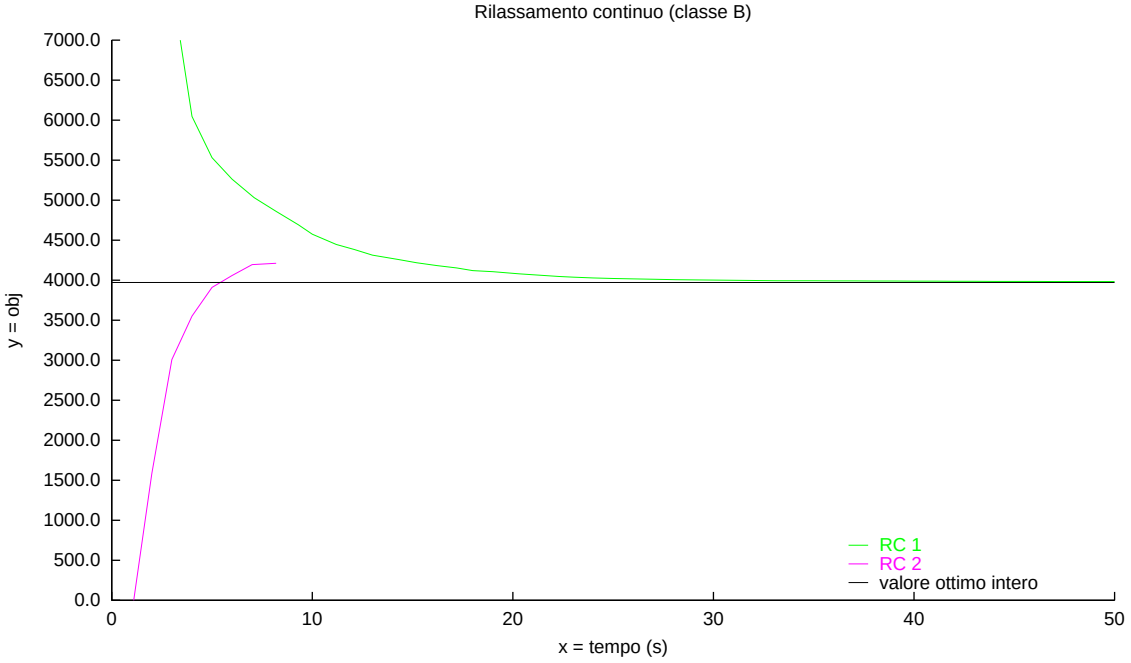
Il rilassamento continuo di un problema di massimizzazione ne fornisce un *upper bound* valido. Con CPLEX è possibile risolvere un problema con variabili continue con il metodo del simplesso, primale o duale, o con un metodo di punto interno. Il metodo di punto interno, sebbene abbia garanzia di polinomialità, per il problema in analisi ha un'eccessiva richiesta di memoria. Il metodo del simplesso ha un costo computazionale (nel caso peggiore) esponenziale nel numero di vincoli del problema, tuttavia sperimentalmente risulta spesso preferibile.

Il rilassamento continuo di TCSS è chiaramente lo stesso di TCSS_1 e di TCSS_2. Per tutte le istanze considerate si ha $|W| \gg |S|$, quindi questo rilassamento ha molte meno variabili che vincoli, e per questo motivo il simplesso duale funziona meglio del simplesso primale. Nei test questo rilassamento viene indicato con $\{\text{RC } 1\}$.

Per istanze di classe B o superiore si ha $|O| > |W|$; TCSS_3 ha pertanto più variabili che vincoli, e quindi per risolverlo ho utilizzato il simplesso primale. Nei test questo rilassamento viene indicato con $\{\text{RC } 2\}$.

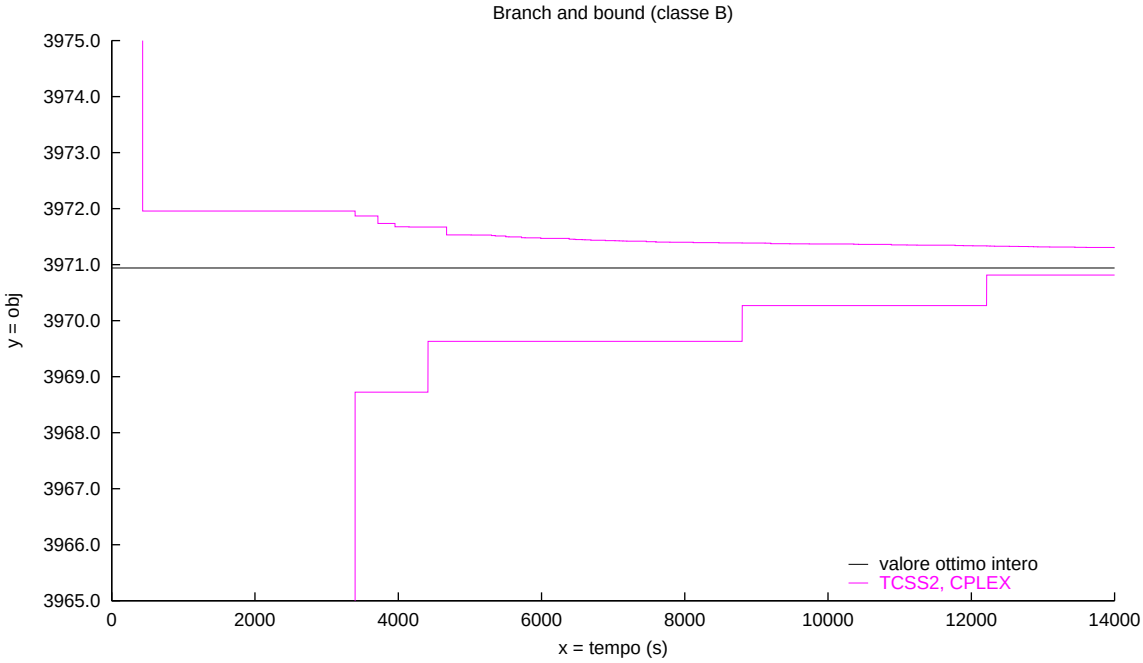
2.4 Risultati

Soluzione del rilassamento continuo. Nel grafico seguente i due rilassamenti continui vengono confrontati su un input di classe B.



Il rilassamento $\{\text{RC } 1\}$ termina oltre il limite di scala del grafico, precisamente dopo 435 secondi con un *gap* dall'ottimo intero dello 0.026%. Il rilassamento $\{\text{RC } 2\}$ termina in 8 secondi ma produce un gap dall'ottimo intero del 6%. L'aggregazione dei vincoli riduce notevolmente il tempo di calcolo, ma abbassa sensibilmente la qualità del *bound* prodotto.

Soluzione del problema intero. Sullo stesso input di classe B la sola formulazione con cui CPLEX ottiene dei risultati è TCSS_2.



La soluzione ottima, che è quella con cui si valutano gli altri metodi, è stata ottenuta con questo rilassamento in 42 ore. Le altre formulazioni non forniscono nessuna soluzione intera nel limite di tempo utilizzato in questo test, quindi non sono presentate nel grafico.

La formulazione TCSS_3 ha molti meno vincoli ma il suo comportamento è molto peggiore di quello di TCSS_2, con la quale ha in comune la natura delle variabili. Questo probabilmente è dovuto al modo con cui CPLEX risolve il problema, calcolando ad ogni nodo il rilassamento continuo, che essendo molto peggiore per TCSS_3 porta a produrre molti meno tagli nell'albero di *branching*. Inoltre si nota sperimentalmente che con la formulazione con più vincoli CPLEX trova spesso rilassamenti continui con soluzioni ottime intere, e quindi produce soluzioni ammissibili più rapidamente.

Capitolo 3

Riformulazione di Benders

La Riformulazione di Benders è una tecnica per trasformare un problema misto intero in un problema intero con una sola variabile continua, introducendo vincoli che descrivano lo spazio delle soluzioni continue eliminate. Il numero di vincoli aggiunti è tipicamente esponenziale nelle dimensioni dell'input; tuttavia, poiché ci si aspetta che solo un limitato sottoinsieme di tali vincoli sia attivo nella soluzione ottima, un rilassamento naturale si ottiene eliminando la maggior parte dei vincoli e generando tagli aggiuntivi con un algoritmo *cutting planes*.

3.1 Punti estremi e raggi estremi

Consideriamo il generico problema di PL in versione di massimizzazione:

PL) $\max z = cx$

$$s.t. \begin{cases} Ax \leqslant b \\ x \in \mathcal{R}^n \end{cases}$$

Definiamo $P = \{x \in \mathcal{R}^n \mid Ax \leqslant b\}$ come l'insieme dei punti ammissibili per il problema. I punti estremi di P sono i vertici del suo poliedro. Una definizione di punto estremo è la seguente.

- Un punto $x \in P$ è un punto estremo di P se $\nexists x^1, x^2 \in P$ distinti tali che $x = \frac{1}{2}(x^1 + x^2)$.

Definiamo inoltre $P^0 = \{r \in \mathcal{R}^n : Ar \leqslant 0\} \setminus \{0\}$. Se $P \neq \emptyset$ allora ogni $r \in P^0$ viene detto *raggio* di P. Un raggio è una direzione verso cui il poliedro è illimitato. Poiché P è convesso vale la seguente proprietà.

- Se r è un raggio di P allora $\forall x \in P, \forall \lambda \in \mathcal{R}, \{x^1 \in \mathcal{R}^n : x^1 = x + \lambda r\} \subseteq P$.

Una definizione di raggio estremo è la seguente.

- Un raggio $r \in P^0$ è un raggio estremo di P se $\nexists r^1, r^2 \in P^0$ distinti tali che $r = \frac{1}{2}(r^1 + r^2)$.

Per un'approfondimento a riguardo vedere [5], capitolo I.4.4.

3.2 Introduzione alla riformulazione di Benders

Nel seguito vengono esposte alcune nozioni fondamentali relative alla riformulazione di Benders, seguendo l'esposizione di Nemhauser e Wolsey [5], capitoli II.3.7 e II.5.4.

Consideriamo il problema misto intero:

$$\text{MIP}) \qquad \max z = c\omega + h\gamma$$

$$s.t. \begin{cases} A\omega + G\gamma \leqslant b & \text{m vincoli} \\ \omega \in \Omega \subseteq \mathcal{Z}_+^n \\ \gamma \in \mathcal{R}_+^p \end{cases}$$

Possiamo vedere le variabili discrete ω come variabili complicanti, senza le quali avremmo un problema di Programmazione Lineare, così come possiamo vedere le variabili continue γ a loro volta come variabili complicanti, senza le quali avremmo un problema puro di Programmazione Lineare Intera.

Per esempio, in un problema di flusso dove le variabili discrete rappresentano decisioni riguardo all'uso degli archi nella rete, una volta fissate le variabili ω rimane un problema ordinario di flusso nello spazio delle γ.

Le procedure che seguono mostrano come il problema sopra descritto in uno spazio $\Omega \times R_+^p$ possa essere riformulato in uno spazio $\Omega \times R^1$, ovvero dove c'è una sola variabile continua.

3.3 Riformulazione di Benders

Come primo passo, supponiamo di aver fissato le variabili discrete ω. Ne risulta il problema di PL:

$$\text{LP}(\omega)) \qquad \max z_{LP}(\omega) = h\gamma$$

$$s.t. \begin{cases} G\gamma \leqslant b - A\omega \\ \gamma \in \mathcal{R}_+^p \end{cases}$$

il cui problema duale è:

$$\text{LP}_d(\omega)) \qquad \min z_{LP_d}(\omega) = \upsilon(b - A\omega)$$

$$s.t. \begin{cases} \upsilon G \geqslant h \\ \upsilon \in \mathcal{R}_+^m \end{cases}$$

Possiamo caratterizzare LP(ω), ovvero stabilire se sia inammissibile, limitato o illimitato, usando la rappresentazione del poliedro duale nei termini dei suoi punti estremi e raggi estremi.

Siano:

- $Q = \left\{ \upsilon \in \mathcal{R}^m_+ : \upsilon G \geqslant h \right\}$ l'insieme di tutti i punti validi per il problema $\text{LP}_d(\omega)$.
- $\left\{ \upsilon^k \in Q : k \in K \right\}$ l'insieme dei punti estremi di Q, con K insieme degli indici dei punti estremi di Q.
- $P = \left\{ \tau \in \mathcal{R}^m_+ : \tau G \geqslant 0 \right\}$ l'insieme dei raggi di Q.
- $\left\{ \tau^j \in P : j \in J \right\}$ l'insieme dei raggi estremi di P, con J insieme degli indici dei raggi estremi di P.

Si noti che se $Q \neq \emptyset$ allora l'insieme dei raggi estremi di P è anche l'insieme dei raggi estremi di Q.
Possiamo caratterizzare $z_{LP}(\omega)$ come segue:

- Se $Q = \emptyset$, cioè se non esistono punti che soddisfano il problema duale, allora:
  - $z^*_{LP}(\omega) = \infty$ se $\tau^j(b - A\omega) \geqslant 0 \quad \forall j \in J$.
  - $z^*_{LP}(\omega) = -\infty$ altrimenti.
- Se $Q \neq \emptyset$, cioè se esistono punti che soddisfano il problema duale, allora:
  - $z^*_{LP}(\omega) = \min\limits_{k \in K} \upsilon^k(b - A\omega) < \infty$ se $\tau^j(b - A\omega) \geqslant 0 \quad \forall j \in J$.
  - $z^*_{LP}(\omega) = -\infty$ altrimenti.

Pertanto, quando $Q \neq \emptyset$, il problema iniziale può essere posto come:

$$\text{MIP}^* \big) \qquad \max_{\omega \in \Omega} z = \left(c\omega + \min_{k \in K} \upsilon^k (b - A\omega) \right)$$
$$s.t. \quad \tau^j(b - A\omega) \geqslant 0 \qquad \forall j \in J$$

Il significato di questa formulazione è che l'ottimo del problema iniziale è dato dall'ottimo su $\forall \omega \in \Omega$ dell'ottimo del sottoproblema duale, espresso nei termini dei suoi punti estremi e raggi estremi.

La formulazione appena descritta, modificata per ottenere una funzione obiettivo standard, porta alla riformulazione di Benders del problema misto intero iniziale:

$$\text{MIP}' \big) \qquad \max z = \eta$$
$$s.t. \begin{cases} \eta \leqslant c\omega + \upsilon^k(b - A\omega) & \forall k \in K \\ \tau^j(b - A\omega) \geqslant 0 & \forall j \in J \\ \omega \in \Omega \\ \eta \in \mathcal{R}^1 \end{cases}$$

Dimostrazione:

- Se $\nexists \omega \in \Omega$ tale che $\tau^j(b - A\omega) \geqslant 0 \quad \forall j \in J$ allora $z^*_{LP}(\omega) = -\infty \quad \forall \omega \in \Omega$ e pertanto $z^* = -\infty$.
- Se $\exists \omega \in \Omega$ tale che $\tau^j(b - A\omega) \geqslant 0 \quad \forall j \in J$, e se $Q = \emptyset$, allora $K = \emptyset$ e pertanto $z^* = \infty$.
- Altrimenti la formulazione MIP$'$ è equivalente alla formulazione MIP.

3.4 Rilassamento della riformulazione di Benders

La riformulazione di Benders ha tipicamente un elevato numero di vincoli, esponenziale nelle dimensioni del problema, pertanto un approccio naturale consiste nel considerare solo un piccolo numero di questi per volta. Quello che segue è il metodo presentato in letteratura per la generazione di vincoli per il rilassamento di MIP$'$.

- *Inizializzazione:* impostare gli insiemi iniziali (potenzialmente vuoti) $K^1 \subseteq K$, $J^1 \subseteq J$. Porre:

$$S_R^1 = \left\{\eta \in \mathcal{R}^1, \omega \in \Omega : \quad \eta \leqslant c\omega + v^k(b - A\omega) \quad \forall k \in K^1, \quad \tau^j(b - A\omega) \geqslant 0 \quad \forall j \in J^1\right\}$$

- *Iterazione t:*

  1. Risolvere il rilassamento di MIP$'$:

  MIP^t) $\quad \max z^t = \eta$

  $$s.t. \quad (\eta, \omega) \in S_R^t$$

  Quindi:

  (a) se MIP^t non ammette soluzione, fermarsi: MIP$'$ non ammette soluzione.

  (b) se MIP^t è illimitato, trovare una soluzione ammissibile (η^t, ω^t) con $\eta^t > \omega$ per qualche valore (grande) di ω.

  (c) altrimenti sia (η^t, ω^t) la soluzione ottima.

  2. *Separazione.* Ottenuta la soluzione (η^t, ω^t), risolvere il problema lineare:

  LP(ω^t)) $\quad \max z_{LP}(\omega^t) = h\gamma$

  $$s.t. \begin{cases} G\gamma \leqslant b - A\omega^t \\ \gamma \in \mathcal{R}_+^p \end{cases}$$

  o il suo duale. Quindi:

  (a) Se $z_{LP}^*(\omega^t) = \infty$, fermarsi: MIP$'$ è illimitato.

  (b) Se $z_{LP}^*(\omega^t)$ è finito, siano γ^t la soluzione primale e v^t la soluzione duale.

  (c) Se LP(ω^t) non ammette soluzione, sia τ^t un raggio duale con $\tau^t(b - A\omega) < 0$. Notare che con l'indicazione di inammissibilità si ottiene anche un punto duale estremo v^t.

  (d) *Test di ottimalità.* Se $c\omega^t + h\gamma^t \geqslant \eta^t$, fermarsi: (γ^t, ω^t) è una soluzione ottima per MIP$'$.

  (e) *Violazione.* Se $c\omega^t + h\gamma^t < \eta^t$, almeno un vincolo di MIP$'$ è violato.

  i. Se $z_{LP}^*(\omega^t)$ è finito, è violato il vincolo $\eta \leqslant c\omega + v^t(b - A\omega)$. Porre $K^{t+1} = K^t \bigcup \{t\}$, ovvero:

$$S_R^{t+1} = S_R^t \bigcap \left\{(\eta, \omega) : \eta \leqslant c\omega + v^t(b - A\omega)\right\}$$

ii. Se $z^*_{LP}(\omega^t) = -\infty$, è violato il vincolo $\tau^t(b - A\omega) \geqslant 0$. Porre $J^{t+1} = J^t \bigcup \{t\}$, ovvero

$$S_R^{t+1} = S_R^t \bigcap \{(\eta, \omega) : \tau^t(b - A\omega) \geqslant 0\}$$

Nonostante non sia necessario, si può anche aggiornare K^t ponendo $K^{t+1} = K^t \bigcup \{t\}$, in modo da ottenere:

$$S_R^{t+1} = S_R^t \bigcap \{(\eta, \omega) : \tau^t(b - A\omega) \geqslant 0, \quad \eta \leqslant c\omega + \upsilon^t(b - A\omega)\}$$

(f) *Iterazione.* Aggiornare $t \leftarrow t + 1$

Nemhauser e Wolsey segnalano diverse difficoltà nell'implementazione della decomposizione di Benders, concernenti la soluzione del rilassamento (di seguito riscritto in modo più completo):

$$\text{MIP}^t \big) \qquad \max z^t = \eta$$

$$s.t. \begin{cases} \eta \leqslant c\omega + \upsilon^k(b - A\omega) & \forall k \in K^t \\ \tau^j(b - A\omega) \geqslant 0 & \forall j \in J^t \\ \eta \in \mathcal{R}^1 \\ \omega \in \Omega \subseteq \mathcal{Z}_+^n \end{cases}$$

dove K^t e J^t sono gli insiemi dei vincoli disponibili dopo le prime t iterazioni.

Una prima difficoltà è che MIP^t è un problema misto intero con una variabile continua. Un modo per eliminare tale difficoltà consiste nel rimpiazzare la variabile η con un valore di soglia η^* nel problema originale MIP$'$. Una volta effettuata tale modifica si ottiene un problema intero puro non di ottimizzazione ma di sola ammissibilità, per lo stesso set di vincoli di MIP^t. Pertanto se il problema risultante diventa ammissibile (inammissibile) si fa crescere (decrescere) η^*, oppure avendo un valore di η^* per cui il problema è ammissibile e uno per cui è inammissibile (un *lower bound* e un *upper bound*) si può procedere per bisezione.

Una seconda difficoltà consiste nel fatto che spesso c'è degenerazione nel sottoproblema continuo LP(ω^t), pertanto non esiste una sola soluzione ottima duale υ^t. La scelta di quale punto estremo duale aggiungere ai vincoli può essere importante. Un possibile approccio, nel caso in cui sia possibile enumerare un insieme di punti duali ottimi, consiste nel generare dei tagli che non siano dominati da altri vincoli.

Un terzo problema riguarda la scelta dei sottoinsiemi iniziali K^1 e J^1. Se non si tengono in considerazione tali insiemi iniziali (ad esempio li si prende vuoti) si potrebbe osservare un comportamento instabile dell'algoritmo. Un modo per inizializzare gli insiemi è quello di risolvere all'ottimo un rilassamento continuo di MIP$'$ e inizializzare K^1 e J^1 con gli insiemi dei punti estremi e dei raggi estremi necessari per generare la soluzione ottima del problema rilassato. In alternativa si può cercare un'euristica per definire una "buona" soluzione iniziale $(\tilde{\omega}, \tilde{\gamma})$ per MIP$'$ e derivare un taglio iniziale dalla soluzione di LP($\tilde{\omega}$).

3.5 Applicazione a TCSS

La riformulazione di Benders è applicabile a problemi misti, mentre il nostro problema appare come un problema intero puro, senza variabili continue. Come è già stato mostrato nel capitolo 2 le formulazioni che si ottengono trasformando in continue le variabili relative alle occorrenze o quelle relative alle stringhe sono formulazioni equivalenti. Su queste due formulazioni, TCSS_1 e TCSS_2, vengono sviluppate due diverse riformulazioni di Benders.

3.5.1 Riformulazione A

Se trasformiamo in continue le variabili x_j relative alle occorrenze otteniamo il problema:

$$\text{MIP}^{\,A}\ \Big) \qquad \max z^A = -(1-\alpha)\sum_{i=1}^{|S|}\tfrac{1}{l_i}y_i + \alpha\sum_{j=1}^{|O|} l_{u(j)}x_j$$

$$s.t.\begin{cases} 0+\sum_{j=1}^{|O|} a_{tj}x_j \leqslant 1 & t=1\ldots|W| \\ -y_{u(j)}+x_j \leqslant 0 & j=1\ldots|O| \\ y_i \in \{0,1\} & i=1\ldots|S| \\ x_j \in \mathcal{R}^+ & j=1\ldots|O| \end{cases}$$

Della formulazione iniziale generica MIP possiamo riconoscere gli oggetti:

$$c = -(1-\alpha)\left[\frac{1}{l_i}\right] \quad h = \alpha\left[l_{u(j)}\right] \quad \Omega = \{y_i \in \{0,1\}\} \quad \omega = [y_i] \quad \gamma = [x_j]$$

$$A = \overbrace{\begin{bmatrix} 0 \\ e_{ji} \end{bmatrix}}^{|S|}\ \begin{matrix}\} |W| \\ \} |O|\end{matrix} \quad G = \overbrace{\begin{bmatrix} a_{tj} \\ I \end{bmatrix}}^{|O|}\ \begin{matrix}\} |W| \\ \} |O|\end{matrix} \quad b = \begin{bmatrix} 1 \\ 0 \end{bmatrix}\ \begin{matrix}\} |W| \\ \} |O|\end{matrix}$$

dove:

$$e_{ji} = \begin{cases} -1 & \text{se} \quad i = u(j) \\ 0 & \text{altrimenti} \end{cases}$$

Definiamo inoltre:

$$\upsilon^k = [\upsilon_1^k\ \upsilon_2^k] \quad \upsilon_1^k = [\overbrace{\upsilon_{1t}^k}^{|W|}] \quad \upsilon_2^k = [\overbrace{\upsilon_{2j}^k}^{|O|}]$$

I vincoli $\eta \leqslant cx + \upsilon^k(b - Ax)$ diventano in questo caso:

$$\eta \leqslant -(1-\alpha)\left[\frac{1}{l_i}\right][y_i] + [\upsilon_1^k\ \upsilon_2^k]\left(\begin{bmatrix} 1 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ e_{ji} \end{bmatrix}[y_i]\right)$$

$$\eta \leqslant -(1-\alpha)\left[\frac{1}{l_i}\right][y_i] + \upsilon_1^k[1] - \upsilon_2^k\left([e_{ij}]\,[y_i]\right)$$

$$\eta \leqslant -(1-\alpha)\left[\frac{1}{l_i}\right][y_i] + \upsilon_1^k[1] - \upsilon_2^k[-y_{u(j)}]$$

$$\eta \leqslant -(1-\alpha)\sum_{i=1}^{|S|}\frac{y_i}{l_i} + \sum_{t=1}^{|W|}\upsilon_{1t}^k + \sum_{j=1}^{|O|}\upsilon_{2j}^k y_{u(j)}$$

Il problema riformulato, una volta eseguite le opportune semplificazioni, è il seguente:

MIP′ ^{A}) $\quad \max z^A = \eta$

$$s.t.\begin{cases} \eta + \displaystyle\sum_{i=1}^{|S|}\left(\frac{(1-\alpha)}{l_i} - \sum_{j\in O_S(i)}\upsilon_{2j}^k\right) y_i \leqslant \sum_{t=1}^{|W|}\upsilon_{1t}^k & \forall k \in K \\ y_i \in \{0,1\} & i = 1\ldots|S| \\ \eta \in \mathcal{R}^1 & \end{cases}$$

Non ci sono vincoli riguardanti i raggi estremi perché il sottoproblema di PL è sempre ammissibile.

Per questo problema i vincoli sono iterativamente generati dalla soluzione del sottoproblema:

LP(ω^t) ^{A}) $\quad \max z_{LP}^A(x^t) = \alpha\sum_{j=1}^{|O|} l_{u(j)}x_j$

$$s.t.\begin{cases} \displaystyle\sum_{j=1}^{|O|} a_{tj}x_j \leqslant 1 & t = 1\ldots|W| \\ x_j \leqslant y_{u(j)}^t & j = 1\ldots|O| \\ x_j \in \mathcal{R}^+ & j = 1\ldots|O| \end{cases}$$

3.5.2 Riformulazione B

Se trasformiamo in continue le variabili y_i relative alle stringhe otteniamo il problema:

MIP ^{B}) $\quad \max z^B = \alpha\sum_{j=1}^{|O|} l_{u(j)}x_j - (1-\alpha)\sum_{i=1}^{|S|}\frac{1}{l_i}y_i$

$$s.t.\begin{cases} x_j - y_{u(j)} \leqslant 0 & j = 1\ldots|O| \\ \displaystyle\sum_{j=1}^{|O|} a_{tj}x_j \leqslant 1 & t = 1\ldots|W| \\ x_j \in \{0,1\} & j = 1\ldots|O| \\ y_i \in \mathcal{R}^+ & i = 1\ldots|S| \end{cases}$$

Della formulazione iniziale generica MIP possiamo riconoscere gli oggetti:

$$c = \alpha\left[l_{u(j)}\right] \quad h = -(1-\alpha)\left[\frac{1}{l_i}\right] \quad \Omega = \left\{\sum_{j=1}^{|O|} a_{tj}x_j \leqslant 1, \quad x_j \in \{0,1\}\right\} \quad \omega = [x_j] \quad \gamma = [y_i]$$

$$A = \overbrace{[I]}^{|O|}\}|O| \quad G = \overbrace{[e_{ji}]}^{|S|}\}|O| \quad b = [0]\}|O|$$

dove:

$$e_{ji} = \begin{cases} -1 & \text{se} \quad i = u(j) \\ 0 & \text{altrimenti} \end{cases}$$

Definiamo inoltre:

$$\upsilon^k = [\overbrace{\upsilon_j^k}^{|O|}]$$

I vincoli $\eta \leqslant cx + \upsilon^k(b - Ax)$ diventano in questo caso:

$$\eta \leqslant \alpha \left[l_{u(j)}\right][x_j] + \upsilon^k([0] - [I][x_j])$$

$$\eta \leqslant \alpha \left[l_{u(j)}\right][x_j] - \upsilon^k[x_j]$$

$$\eta \leqslant \alpha \sum_{j=1}^{|O|} l_{u(j)} x_j - \sum_{j=1}^{|O|} \upsilon_j^k x_j$$

$$\eta \leqslant \sum_{j=1}^{|O|} \left(\alpha l_{u(j)} - \upsilon_j^k\right) x_j$$

Il problema riformulato, una volta eseguite le opportune semplificazioni, è il seguente:

MIP′ ^{B}) $\quad \max z^B = \eta$

$$s.t. \begin{cases} \eta + \sum_{j=1}^{|O|} \left(\upsilon_j^k - \alpha l_{u(j)}\right) x_j \leqslant 0 & \forall k \in K \\ \sum_{j=1}^{|O|} a_{tj} x_j \leqslant 1 & j = 1 \ldots |O| \\ x_j \in \{0,1\} & j = 1 \ldots |O| \\ \eta \in \mathcal{R}^1 & \end{cases}$$

Anche in questo caso non ci sono vincoli riguardanti i raggi estremi perché il sottoproblema di PL è sempre ammissibile. Per questo problema i vincoli sono iterativamente generati dalla soluzione del sottoproblema:

LP(ω^t) ^{B}) $\quad \max z_{LP}^B(x^t) = -(1-\alpha) \sum_{i=1}^{|S|} \frac{1}{l_i} y_i$

$$s.t. \begin{cases} -y_{u(j)} \leqslant -x_j & j = 1 \ldots |O| \\ y_i \in \mathcal{R}^+ & i = 1 \ldots |S| \end{cases}$$

3.6 Risultati e considerazioni

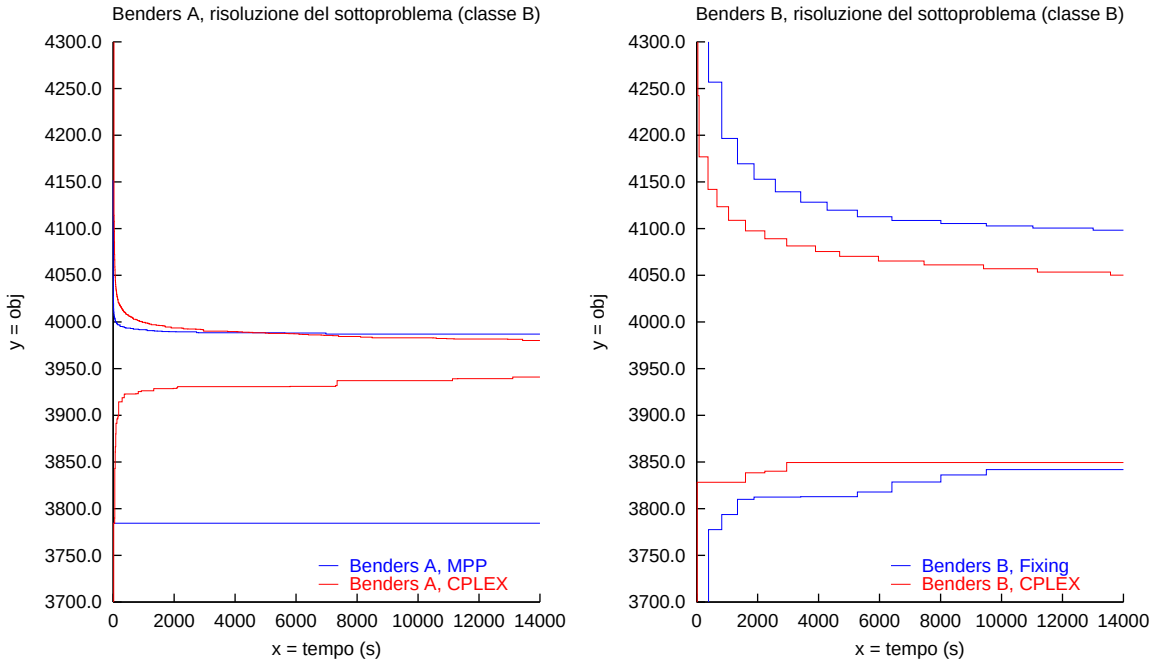
Soluzione esatta del problema master. Per la soluzione esatta del problema *master* sono stati tentati due approcci:

- risolvere il problema *master* con CPLEX.
- rendere la variabile η un parametro, risolvendo più problemi interi e variando il parametro per bisezione. Per risolvere il problema intero si è provato sia ad utilizzare CPLEX, sia ad applicare il metodo di Balas [11].

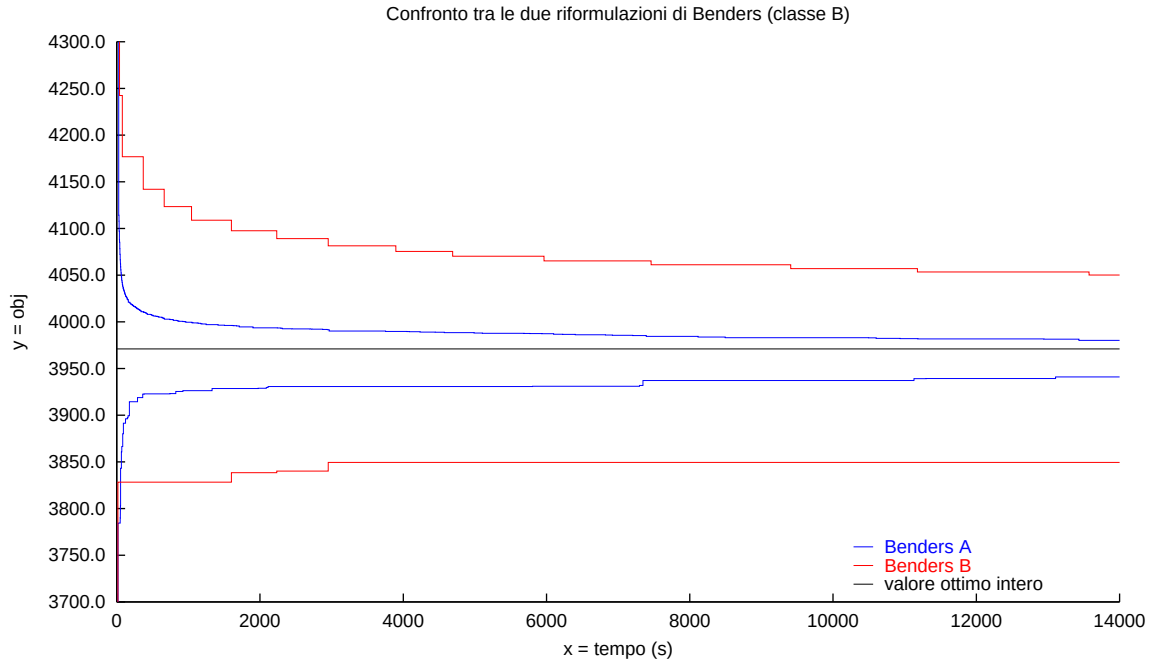
Utilizzando un'istanza di classe B, con nessuno degli approcci utilizzati è stato possibile risolvere il problema *master* con più di sette-otto vincoli, sia per problemi di memoria (dimensioni dell'albero di *branching*) che di tempo.

Soluzione approssimata del problema master. Risolvere in modo approssimato il problema *master* significa non avere la garanzia che il taglio prodotto dal sottoproblema continuo renda inammissibile l'attuale valore ottimo di η, e quindi rischiare di bloccarsi su una soluzione non ottima. Tuttavia si è notato sperimentalmente che spesso CPLEX trova rapidamente la soluzione ottima, ma che spende molto tempo a confermarne l'ottimalità. Per poter avere dei risultati con cui confrontare la riformulazione di Benders con gli altri metodi è stato scelto di risolvere il problema con CPLEX in modo approssimato, con un margine d'errore massimo dello 0.01%.

Soluzione del sottoproblema continuo. Per risolvere il sottoproblema continuo ho utilizzato sia CPLEX, con il metodo del simplesso, che metodi risolutivi polinomiali specifici per il sottoproblema delle diverse riformulazioni (i sottoproblemi continui sono MPP e un problema banale), utilizzando poi CPLEX per ottenere il punto duale corrispondente a tale ottimo.



Come mostrano i due grafici, l'approccio che si è rivelato più efficace è quello di risolvere il sottoproblema completamente con CPLEX. I tagli prodotti dai punti duali relativi all'ottimizzazione con il simplesso risultano infatti più efficaci di quelli ottenuti dagli altri algoritmi. Questo effetto è particolarmente sensibile nella riformulazione A, a causa di una forte degenerazione nel sottoproblema continuo in cui esistono numerosi cammini ottimi equivalenti.



L'input di questo test è lo stesso input di classe B usato nel capitolo precedente. La riformulazione A risulta decisamente più efficace.

Capitolo 4

Rilassamento Lagrangeano

Il rilassamento Lagrangeano consiste nell'eliminare un insieme di vincoli, inserendo un termine nella funzione obiettivo che ne penalizzi la violazione. Eliminando dal problema i vincoli "difficili" si ottiene un problema più semplice da risolvere, eventualmente polinomiale, la cui soluzione ottima fornisce un *bound* duale utilizzabile in un algoritmo *branch and bound*.

4.1 Introduzione al rilassamento Lagrangeano

Dato un generico problema (in forma di massimizzazione):

IP) $\max z = cx$

$$s.t. \begin{cases} Ax \leqslant a & m \text{ vincoli complicanti} \\ Bx \leqslant b & q \text{ vincoli semplici} \\ x \in \mathcal{Z}_+^n \end{cases}$$

si ottiene il rilassamento:

LR(λ)) $\max z_{LR} = cx + \lambda(a - Ax)^T$

$$s.t. \begin{cases} Bx \leqslant b & q \text{ vincoli semplici} \\ x \in \mathcal{Z}_+^n \end{cases}$$

dove $\lambda \in \mathcal{R}_+^m$ è un vettore di coefficienti, detti appunto moltiplicatori Lagrangeani. Poiché $\lambda_i \geqslant 0$, le violazioni su un vincolo di tipo "$\leqslant$" si risolvono in una penalizzazione nella funzione obiettivo.

Per mostrare che LR(λ) è un rilassamento è sufficiente mostrare che ogni soluzione ammissibile per IP lo è anche per LR(λ) e che per ogni soluzione x ammissibile per IP si ha $z_{LR}(x) \geqslant z(x)$, e questo si può verificare sapendo che $\lambda_i \geqslant 0$ e $Ax \leqslant a$, da cui $\lambda(a - Ax) \geqslant 0$ e pertanto $cx + \lambda(a - Ax) \geqslant cx$.

LR(λ) è un *upper bound* di IP per qualsiasi $\lambda \in \mathcal{R}_+^m$. Il minimo *upper bound* è dato dalla soluzione del seguente problema, detto Lagrangeano Duale di IP:

$$\text{LD}) \qquad z_{LD} = \min z_{LR}(\lambda)$$
$$s.t. \quad \lambda \in \mathcal{R}_+^m$$

Per una descrizione formale del rilassamento Lagrangeano vedere [6].

4.2 Metodo del Sottogradiente

Per la soluzione, in genere approssimata, di LD possono essere utilizzati diversi algoritmi. Il metodo applicato nel nostro caso, *Subgradient Optimization* o metodo del Sottogradiente, consiste in iterazioni successive in cui λ viene modificato tramite un sottogradiente, ovvero una direzione di possibile miglioramento.

Ad ogni iterazione k si ha un vettore di moltiplicatori λ^k, con cui si risolve il sottoproblema LR(λ^k) e si ottiene la sua soluzione ottima x^*. Da essa si calcola un sottogradiente $g^k = a - Ax^*$ e si ottiene il nuovo valore dei moltiplicatori $\lambda_i^{k+1} = \max(0, \lambda_i^k + \phi_k \cdot g_i^k)$ dove ϕ identifica una serie monotona non crescente con limite 0.

In questo capitolo si indicano con z_{ub} il valore corrente dell'*upper bound* e con z_{lb} il valore della miglior soluzione ammissibile trovata.

In [7] vengono descritti alcuni accorgimenti da utilizzare nell'applicare questo metodo:

- per determinare un corretto valore di ϕ_k viene aggiunto un parametro $0 < \pi_k \leqslant 2$ e viene definito

$$\phi_k = \frac{\pi_k(1.05 z_{ub} - z_{lb})}{den_k} \quad \text{dove} \quad den_k = \sum_{i:(g_i^k>0) \vee (\lambda_i^k>0)} (g_i^k)^2$$

  ovvero la somma dei quadrati dei sottogradienti che modificano i rispettivi moltiplicatori. In questo modo più si è vicini alla soluzione ottima e meno vengono modificati i moltiplicatori. Il parametro π deve decrescere con il numero di iterazioni; per ottenere questo effetto il modo suggerito è di dimezzare π ogni N iterazioni senza miglioramento di z_{ub}, e viene indicato $N = 30$ come valore ragionevole. Il metodo che ho utilizzato per far decrescere π è quello di moltiplicarlo ad ogni iterazione per un parametro $\beta < 1$, opportunamente tarato.

- la scelta iniziale dei moltiplicatori non dovrebbe condizionare troppo il risultato dell'algoritmo; tuttavia, essendo lo stesso inserito in un algoritmo *branch and bound*, i moltiplicatori vengono inizializzati al valore ottimo dei moltiplicatori del nodo padre, e ad ogni miglioramento di z_{ub} vengono aggiornati. Per il solo nodo radice si ha $\lambda_i^0 = 0 \quad \forall i$.

- viene indicato come valore iniziale di π il valore $\pi_0 = 2$; tuttavia poiché nei nodi figlio si parte con i migliori moltiplicatori del nodo padre, e visto che la soluzione non cambia in modo sensibile dal nodo padre al nodo figlio, risulta ragionevole utilizzare un valore iniziale molto più basso; il valore usato nel nodo radice è quello indicato in letteratura mentre per i nodi successivi è $\pi_0 = 0.02$.

- dal momento che difficilmente si ottiene la terminazione per raggiungimento del valore ottimo, sono necessari dei test di terminazione ausiliari; l'algoritmo viene terminato dopo un numero massimo di iterazioni, nonché dopo un numero massimo di iterazioni in cui non si è avuto un miglioramento di z_{ub} o di z_{lb}.

4.3 Applicazione al problema

Lo schema con cui è stato applicato il rilassamento Lagrangeano consiste in un *branch and bound* in cui ad ogni nodo viene risolto LD in modo approssimato tramite il metodo del Sottogradiente; ad ogni iterazione di questo metodo si ottiene una soluzione duale, quindi un *upper bound*, da cui si ricava con un'euristica una soluzione primale, quindi un *lower bound*. L'albero di ricerca è gestito con strategia *best first* tramite una coda di priorità, scegliendo ad ogni passo come nodo attivo quello con *upper bound* maggiore.

Il problema in analisi ha due gruppi ben distinti di vincoli, e su questi due gruppi si possono sviluppare due diversi rilassamenti. Sono state provate strategie di *branching* diverse, che vengono discusse insieme al rilassamento al quale vengono applicate.

4.4 Propagazione dei vincoli e riduzione

In un generico nodo dell'albero di ricerca, sia come effetto dell'operazione di *branching* che per l'applicazione di tecniche di riduzione del problema, può capitare che alcune variabili risultino fissate. È possibile allora fissare il valore di altre variabili, riducendo ulteriormente la dimensione del problema, tramite alcune semplici tecniche di propagazione dei vincoli. Siano:

- O^N l'insieme delle occorrenze non fissate, e O^F l'insieme di quelle fissate.
- S^N l'insieme delle stringhe non fissate, e S^F l'insieme di quelle fissate.
- $O_S^N(i) = O_S(i) \cap O^N$ l'insieme delle occorrenze non fissate della stringa i (sia essa libera o fissata), e $O_S^F(i) = O_S(i) \cap O^F$ l'insieme di quelle fissate.

Dopo aver fissato $x_j \leftarrow 0$, è possibile controllare alcuni casi:

- se $u(j) \in S^N$ e $O_S^N(u(j)) = \emptyset$ si può fissare $y_{u(j)} \leftarrow 0$.
- se $u(j) \in S^N$ e vale

$$\frac{(1-\alpha)}{l_{u(j)}} \geqslant \sum_{k \in O_S^F(u(j))} (\alpha l_{u(k)} x_k) + \sum_{k \in O_S^N(u(j))} (\alpha l_{u(k)})$$

  allora si può fissare $y_{u(j)} \leftarrow 0$.
- se esiste un'occorrenza $k \in O^N$ non sovrapposta a nessun'altra occorrenza libera, e inoltre se $y_k = 1$ oppure se $O_S^N(u(k)) = \{k\}$ si può fissare $x_k \leftarrow 1$.

Dopo aver fissato $x_j \leftarrow 1$:

- se $u(j) \in S^N$ si deve fissare $y_{u(j)} \leftarrow 1$
- si devono fissare a 0 tutte le variabili relative a occorrenze sovrapposte anche solo parzialmente a j:

$$\forall k \in O^N \mid (\exists t \in W \mid a_{tj} = a_{tk} = 1) \qquad x_k \leftarrow 0$$

Dopo aver fissato $y_i \leftarrow 0$, si deve fissare $x_j \leftarrow 0 \quad \forall j \in O_S^N(i)$.

Dopo aver fissato $y_i \leftarrow 1$, se esiste $j \in O_S^N(i)$ non sovrapposta a nessun'altra occorrenza libera, si può fissare $x_j \leftarrow 1$.

4.5 Rilassamento 1

Rilassando l'insieme dei vincoli di non sovrapposizione otteniamo il problema:

$$\text{LR1)} \qquad \max z_{LR1}(\lambda) = \left(\alpha \sum_{j=1}^{|O|} l_{u(j)} x_j - (1-\alpha) \sum_{i=1}^{|S|} \frac{1}{l_i} y_i \right) + \sum_{t=1}^{|W|} \lambda_t (1 - \sum_{j=1}^{|O|} a_{tj} x_j)$$

$$s.t. \begin{cases} x_j - y_{u(j)} \leqslant 0 & j = 1 \dots |O| \\ x_j \in \{0,1\} & j = 1 \dots |O| \\ y_i \in \{0,1\} & i = 1 \dots |S| \end{cases}$$

La funzione obiettivo può essere riscritta come:

$$z_{LR1}(\lambda) = \sum_{j=1}^{|O|} \left(\alpha l_{u(j)} - \sum_{t=1}^{|W|} \lambda_t a_{tj} \right) x_j - (1-\alpha) \sum_{i=1}^{|S|} \frac{1}{l_i} y_i + \sum_{t=1}^{|W|} \lambda_t$$

Il termine della funzione obiettivo $\sum_{t=1}^{|W|} \lambda_t$, una volta scelti i moltiplicatori, è costante e non viene considerato nell'ottimizzazione. Il valore ottimo di questo problema si può ricavare valutando i costi ridotti delle variabili.

Sia $c_j = \alpha l_{u(j)} - \sum_{t=1}^{|W|} \lambda_t a_{tj}$ il costo ridotto delle variabili x_j, e si definisca l'insieme $O_+(\lambda) \subseteq O$ delle occorrenze con costo ridotto positivo : $O_+(\lambda) = \{j : c_j > 0\}$.

Non essendoci più vincoli di non sovrapposizione il costo ridotto h_i delle variabili y_i si può calcolare come il costo della stringa più la somma dei costi ridotti di tutte le sue occorrenze in $O_+(\lambda)$:

$$h_i = \sum_{j \in O_+(\lambda) \cap O_S(i)} (c_j) - \frac{1-\alpha}{l_i}$$

Si può costruire la soluzione ottima scegliendo le sole stringhe con costo ridotto positivo, ovvero fissare:

$$y_i = \begin{cases} 1 & \text{se } h_i > 0 \\ 0 & \text{altrimenti} \end{cases}$$

e fissare di conseguenza le x_j ai *bound* corrispondenti:

$$x_j = \begin{cases} y_{u(j)} & \text{se } c_j > 0 \\ 0 & \text{altrimenti} \end{cases}$$

4.5.1 Euristica Lagrangeana

La soluzione ottima (x^*, y^*) di LR1 può violare alcuni vincoli di non sovrapposizione. È possibile ricavare una soluzione ammissibile per il problema originale risolvendo il sottoproblema MPP con i valori di y^*.

4.5.2 Riduzione del problema

Data la soluzione ottima del rilassamento Lagrangeano (x^*, y^*), con moltiplicatori ottimi λ^*, se provando a invertire il valore di una variabile libera, stringa od occorrenza, si ottiene un *upper bound* più basso del miglior *lower bound* trovato, si può ridurre il problema, fissando la variabile.

Sia $z^*_{LR1}(\lambda^*)$ il valore ottimo del rilassamento Lagrangeano. Si danno i seguenti casi:

- Fissando $y_i \leftarrow (1 - y^*_i)$ l'ottimo del rilassamento Lagrangeano, dati gli stessi moltiplicatori, diventa $z'_{LR1}(\lambda^*) = z^*_{LR1}(\lambda^*) - |h_i|$. Se $z'_{LR1}(\lambda^*) \leqslant z_{LB}$ si può fissare $y_i \leftarrow y^*_i$.
- Fissando $x_j \leftarrow (1 - x^*_j)$:
  - se $x^*_j = 1$ oppure $y^*_{u(j)} = 1$ l'ottimo diventa $z'_{LR1}(\lambda^*) = z^*_{LR1}(\lambda^*) - |c_j|$.
  - altrimenti:
    - se $c_j > 0$ l'ottimo diventa $z'_{LR1}(\lambda^*) = z^*_{LR1}(\lambda^*) - |h_{u(j)}|$
    - se $c_j < 0$ l'ottimo diventa $z'_{LR1}(\lambda^*) = z^*_{LR1}(\lambda^*) - |h_{u(j)}| - |c_j|$

  Se $z'_{LR1}(\lambda^*) \leqslant z_{LB}$ si può fissare $x_j \leftarrow x^*_j$.

4.5.3 Strategia di *branching*

Per questo rilassamento sono state provate tre diverse strategie di *branching*. Data la soluzione del rilassamento Lagrangeano:

- scegliere il carattere coperto dal maggior numero di occorrenze (sia q tale numero) e generare $q + 1$ nodi. Nel nodo k (con $k = 1..q$) fissare a 1 la variabile relativa alla k-esima occorrenza, e nel nodo $(q + 1)$ fissare a 0 tutte le variabili relative alle q occorrenze.
- sia x_j la variabile con costo ridotto maggiore, rispetto ai moltiplicatori ottimi, tra le q occorrenze di cui sopra. Generare due nodi fissando rispettivamente $x_j = 0$ e $x_j = 1$.
- scegliere la variabile $y_{u(j)}$, relativa alla stringa dell'occorrenza utilizzata nel metodo precedente, e generare due nodi fissando rispettivamente $y_{u(j)} = 0$ e $y_{u(j)} = 1$.

La seconda strategia produce un cattivo *upper bound*, quasi uguale a quello del nodo padre, nel nodo in cui viene fissata $x_j = 1$. Questa strategia può essere utile in una politica di gestione dell'albero *depth first*, ma con la politica che ho utilizzato si ottiene un miglioramento troppo lento dell'*upper bound*.

La prima strategia produce molti nodi, e l'utilizzo della memoria cresce esponenzialmente con il numero di nodi analizzati, soprattutto nella fase iniziale in cui non ci sono tagli all'albero di *branching*; inoltre essa risente dello stesso problema della seconda strategia.

La terza strategia dal punto di vista teorico è una cattiva scelta, visto che nel nodo in cui viene fissata $y_{u(j)} = 1$ non viene invalidata la soluzione ottima del nodo padre, in cui tale variabile aveva già quel valore. Tuttavia i risultati sperimentali mostrano come anche per questo nodo si ottenga un discreto decremento dell'*upper bound*; tale effetto è dovuto però alle iterazioni di sottogradiente, dove i moltiplicatori del nodo padre risultano una buona scelta iniziale.

La strategia utilizzata per il confronto con il secondo rilassamento quindi è la terza.

4.6 Rilassamento 2

Rilassando il set di vincoli $x_j - y_{u(j)} \leqslant 0$ otteniamo il problema:

$$\text{LR2)} \qquad \max z_{LR2}(\mu) = \left(\alpha \sum_{j=1}^{|O|} l_{u(j)} x_j - (1-\alpha) \sum_{i=1}^{|S|} \tfrac{1}{l_i} y_i \right) + \sum_{j=1}^{|O|} \mu_j \left(y_{u(j)} - x_j \right)$$

$$s.t. \begin{cases} \sum_{j=1}^{|O|} a_{tj} x_j \leqslant 1 & t = 1 \dots |W| \\ x_j \in \{0,1\} & j = 1 \dots |O| \\ y_i \in \{0,1\} & i = 1 \dots |S| \end{cases}$$

La funzione obiettivo può essere riscritta come:

$$\sum_{j=1}^{|O|} \left(\alpha l_{u(j)} - \mu_j \right) x_j + \sum_{i=1}^{|S|} \left(\sum_{j \in O_S(i)} (\mu_j) - \frac{1-\alpha}{l_i} \right) y_i$$

Dal momento che non ci sono vincoli che legano i due gruppi di variabili, LR2 è scomponibile nel sottoproblema:

$$\text{LR2y)} \qquad \max z_{LR2y}(\mu) = \sum_{i=1}^{|S|} \left(\sum_{j \in O_S(i)} (\mu_j) - \tfrac{1-\alpha}{l_i} \right) y_i$$

$$s.t. \quad y_i \in \{0,1\} \qquad i = 1 \dots |S|$$

risolvibile banalmente fissando:

$$y_i = \begin{cases} 1 & \text{se } \sum_{j \in O_S(i)} \mu_j > \frac{1-\alpha}{l_i} \\ 0 & \text{altrimenti} \end{cases}$$

e nel sottoproblema:

$$\text{LR2x)} \qquad \max z_{LR2x}(\mu) = \sum_{j=1}^{|O|} \left(\alpha l_{u(j)} - \mu_j\right) x_j$$

$$s.t. \begin{cases} \sum_{j=1}^{|O|} a_{tj} x_j \leqslant 1 & t = 1 \dots |W| \\ x_j \in \{0,1\} & j = 1 \dots |O| \end{cases}$$

che non è altro che una versione di MPP con coefficienti modificati, in cui tutte le stringhe sono attive.

4.6.1 Euristica Lagrangeana

La soluzione ottima (x^*, y^*) di LR2 può violare alcuni vincoli $x_j - y_{u(j)} \leqslant 0$. Per completare la soluzione è sufficiente fissare a 0 alcune variabili x e a 1 alcune variabili y, fino alla soddisfazione dei vincoli. Un modo per decidere quali variabili fissare consiste nel valutare, per ogni stringa y_i che andrebbe fissata ad 1, il suo costo rispetto al valore delle occorrenze che la forzano ad 1 e fissare

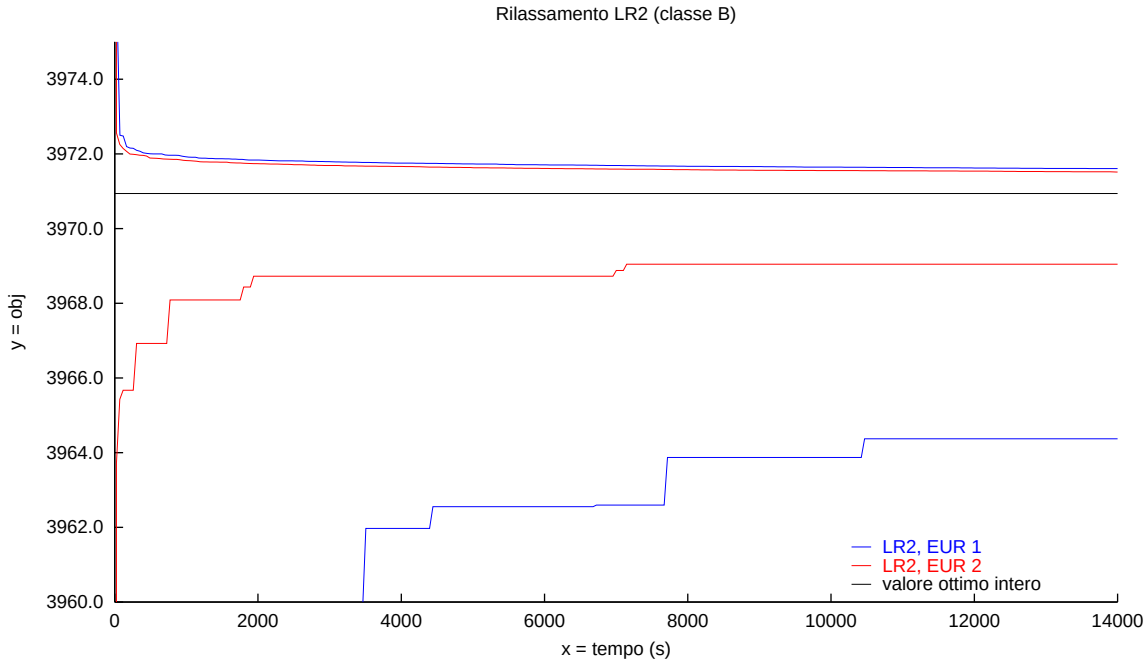
$$y_i = \begin{cases} 1 & \text{se } \sum_{j \in O_S(i)} \alpha l_i x_j^* > \frac{(1-\alpha)}{l_i} \\ 0 & \text{altrimenti} \end{cases}$$

e per le stringhe in cui si è fissato $y_i = 0$ fissare anche tutte le relative occorrenze a $x_j = 0 \quad \forall j \in O_S(i)$. Questa euristica è indicata nei test come $\{$EUR 1$\}$.

Un'euristica aggiuntiva consiste nell'ottimizzare il risultato di $\{$EUR 1$\}$ risolvendo MPP sulle sole stringhe scelte nel passo precedente; inoltre, poiché generalmente le soluzioni prodotte hanno troppe stringhe, si ottiene un ulteriore miglioramento ponendo

$$y_i = \begin{cases} 1 & \text{se } \sum_{j \in O_S(i)} \alpha l_i x_j^* > \frac{(1-\alpha)}{l_i} + \epsilon \\ 0 & \text{altrimenti} \end{cases}$$

con ϵ piccolo; nei test ho utilizzato $\epsilon = \alpha$. Questa seconda euristica è indicata nei test come $\{$EUR 2$\}$.



In questo grafico viene mostrato il rilassamento LR2 con l'euristica $\{\text{EUR } 1\}$ e con l'euristica $\{\text{EUR } 2\}$, su un input di classe B. Si nota un notevole incremento del *lower bound*, su cui l'euristica lavora, e un miglioramento molto meno sensibile dell'*upper bound*; il leggero miglioramento dell'*upper bound* è dovuto all'influenza del gap sulla modifica dei moltiplicatori. Nonostante la singola iterazione sia computazionalmente più complessa, l'euristica $\{\text{EUR } 2\}$ risulta decisamente più efficiente.

4.6.2 Riduzione del problema

Data la soluzione del rilassamento Lagrangeano (x^*, y^*), con moltiplicatori ottimi μ^*, se provando a invertire il valore di una variabile libera si ottiene un *upper bound* più basso del miglior *lower bound* trovato, si può ridurre il problema.

Sia $z^*_{LR2}(\mu^*)$ il valore ottimo del rilassamento Lagrangeano. Fissando $y_i \leftarrow (1 - y^*_i)$, l'ottimo del rilassamento Lagrangeano, dati gli stessi moltiplicatori, diventa:

$$z'_{LR2}(\mu^*) = z^*_{LR2}(\mu^*) - \left| \sum_{j \in O_S(i)} (\mu^*_j) - \frac{1-\alpha}{l_i} \right|$$

Se $z'_{LR2}(\mu^*) < z_{LB}$ si può fissare $y_i \leftarrow y^*_i$.

4.6.3 Strategia di *branching*

Data la soluzione del rilassamento Lagrangeano (x^*, y^*), con moltiplicatori ottimi μ^*, in generale esiste un certo insieme di variabili che violano i vincoli $x_j - y_{u(j)} \leqslant 0$. Per il *branching* viene scelta la variabile y_i che, fissata a 0 e a 1 nei due nodi generati, massimizza il minimo decremento di $z^*_{LR2}(\mu^*)$.

Si definisca $V(i) = \left\{ j \in O_S(i) \mid x^*_j > y^*_i \right\}$, ovvero l'insieme delle occorrenze della stringa i che violano il vincolo rilassato.

Fissando una variabile y_i si danno i seguenti due casi:

- se si fissa $y_i \leftarrow 0$ si devono fissare a 0 anche tutte le sue occorrenze. Il valore ottimo di LR2y non cambia, poiché vale già $y_i^* = 0$, mentre l'ottimo di LR2x diminuisce del valore di tutte le occorrenze eliminate; pertanto nel nodo figlio, con i moltiplicatori μ^*, l'ottimo di LR2 diminuisce di

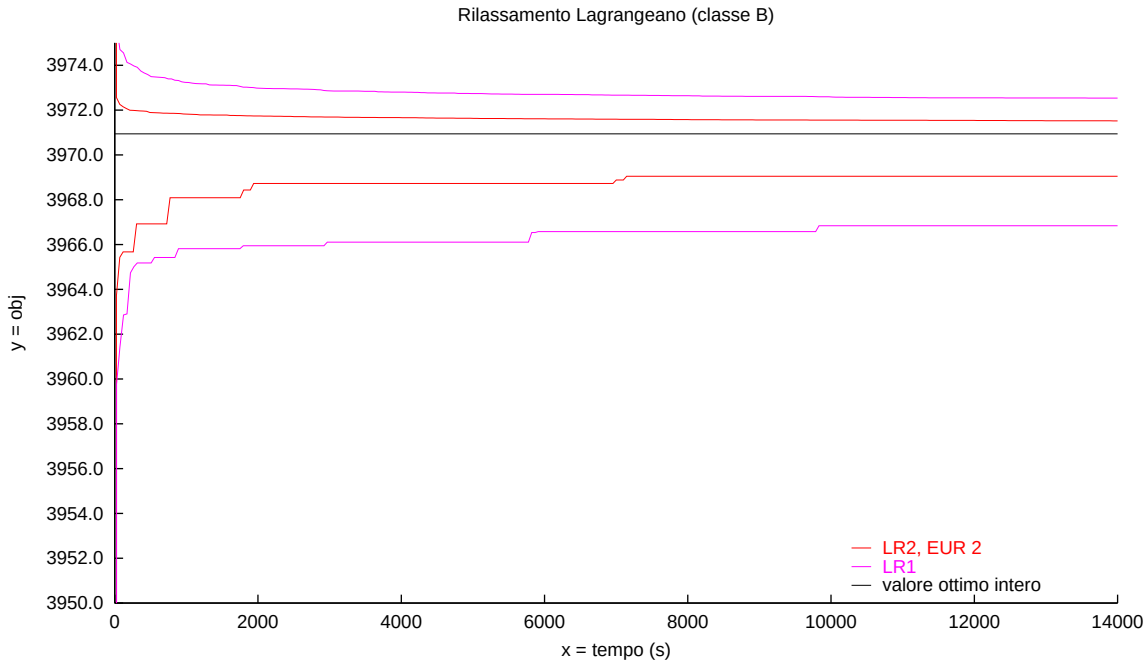
$$\sigma_i^0 = \sum_{j \in V(i)} (\alpha l_i - \mu_j^*)$$

- se si fissa $y_i \leftarrow 1$ non si modifica l'ottimo di LR2x, poiché vengono scelte le stesse occorrenze, mentre il valore ottimo di LR2y aumenta del coefficiente di costo ridotto (negativo) della stringa i; pertanto nel nodo figlio, con i moltiplicatori μ^*, l'ottimo di LR2 diminuisce di

$$\sigma_i^1 = \frac{1-\alpha}{l_i} - \sum_{j \in O_S(i)} (\mu_j^*)$$

La variabile di *branching* scelta è quella che massimizza il minimo tra i due decrementi, in modo da ottenere un *upper bound* bilanciato sui due nodi; quindi viene scelta la stringa $i = \arg\min_k(\max(\sigma_k^0, \sigma_k^1))$.

4.7 Risultati



In questo grafico vengono messi a confronto i due rilassamenti, su un input di classe B. LR2 produce sia *upper bound* migliori che soluzioni ammissibili migliori. Nonostante abbia un maggior numero di moltiplicatori ($|O| > |W|$), il rilassamento che mantiene i vincoli di non sovrapposizione si rivela più efficace nel mantenere l'informazione sulla struttura combinatoria del problema.

Capitolo 5

Algoritmi di ricerca locale

Gli algoritmi di ricerca locale si basano sul concetto di intorno, o *neighborhood*. L'intorno di una soluzione s è l'insieme $N(s)$ dei suoi *vicini*, cioè l'insieme delle soluzioni raggiungibili da s in un "passo".

La ricerca locale consiste in una procedura iterativa in cui la soluzione corrente viene sostituita da uno dei suoi vicini, scelto in modi diversi, fino al soddisfacimento di un criterio di terminazione.

5.1 Strategie di ricerca locale

Esistono diverse strategie di ricerca locale:

- *best improve*: ad ogni iterazione si cerca tra tutti i vicini $n \in N(s)$ quello con $z(n)$ maggiore; se $z(n) > z(s)$ il vicino rimpiazza la soluzione s, altrimenti è stato raggiunto il minimo locale e l'algoritmo termina.
- *first improve*: ad ogni iterazione si sceglie un solo vicino $n \in N(s)$ e se $z(n) > z(s)$ questo rimpiazza la soluzione corrente s e si itera, altrimenti viene cercato un altro vicino. Questo processo può proseguire finché non sono stati valutati tutti i vicini in $N(s)$ senza trovarne uno migliorante.
- strategie ibride: ad ogni iterazione si valutano un certo numero di vicini e si sceglie quello con $z(n)$ maggiore; se $z(n) > z(s)$ questo rimpiazza la soluzione corrente s e si itera, altrimenti si procede a valutare un altro gruppo di vicini. Anche in questo caso si procede finché sono stati valutati tutti i vicini senza trovarne uno migliorante.

Se l'intorno non contiene un numero di soluzioni troppo elevato e se la valutazione di un vicino non è dispendiosa in termini di tempo è possibile utilizzare la strategia *best improve*. In caso contrario è necessario valutare un minor numero di vicini per volta, eventualmente valutandone altri se non se ne è trovato uno migliorante, cioè utilizzare una delle altre strategie.

Con l'intorno che viene definito nel seguito, la valutazione di una soluzione, sebbene venga fatta in tempo polinomiale, ha un costo elevato, perché implica la risoluzione di MPP. Inoltre il numero di vicini è pari al numero delle stringhe, e perciò è molto alto. Ho quindi deciso di valutare ad ogni iterazione un numero di vicini limitato, definito da un parametro NBR. Per NBR $= 1$ la strategia corrisponde a *first improve*, mentre per $NBR = |S|$ corrisponde a *best improve*.

Per dare un corretto valore al parametro NBR occorre considerare che maggiore è il numero di vicini che si analizzano, migliore e più rapida (come numero di iterazioni) è la convergenza; tuttavia la singola iterazione diventa più lenta.

Per i metodi di ricerca locale implementati si è fatto riferimento a [8].

5.2 *Threshold Accepting*

Threshold Accepting è un algoritmo di ricerca locale a *threshold*, o soglia, in cui una nuova soluzione viene accettata sempre se migliorante, e solo entro una soglia se peggiorante.

Uno pseudocodice per questo algoritmo, in versione di massimizzazione, è:

```
sol := GenerateStartingSolution()
k := 0
while ( not ShouldStop() )
    t_k := ThresholdAtStep(k)
    neighbor := SelectNeighbor(sol)
    if ( z(sol) - z(neighbor) < t_k )
        sol := neighbor
    else
        ModifyNeighborhood(sol)
    k := k+1
```

Non è previsto un particolare schema di variazione della soglia di accettazione; in letteratura vengono date alcune indicazioni, ovvero che si abbia $t_k \geqslant t_{k+1} \geqslant 0 \quad \forall k$, e $\lim_{k \to \infty} t_k = 0$, cioè che la serie sia monotona non crescente e con limite 0, in modo da avere la garanzia di convergenza ad un minimo locale.

5.2.1 Applicazione di *Threshold Accepting* a TCSS

L'applicazione di *Threshold Accepting* a TCSS consiste nel definire i seguenti componenti dell'algoritmo:

- Spazio delle soluzioni: una soluzione s consiste nel valore y^s a cui sono fissate le variabili relative alle stringhe; l'algoritmo prende in considerazione solo tali variabili, ottenendo una soluzione completa per il problema originario, necessaria per valutare la soluzione, dalla soluzione di MPP.
- `GenerateStartingSolution`: sono state provate diverse soluzioni:
  - partire da un insieme di stringhe scelte vuoto.
  - partire da un insieme generato casualmente.
  - partire dall'insieme di tutte le stringhe.

  La soluzione che si è rivelata più efficace è stata quella di partire con tutte le stringhe scelte, cioè $y_i = 1 \quad \forall i = 1..|S|$.

- `SelectNeighbor`, `ModifyNeighborhood`: data una soluzione s, $N(s)$ è l'insieme delle soluzioni che si differenziano da s per il valore di una sola variabile y_i; per ogni soluzione esistono quindi esattamente $|S|$ vicini. La strategia di ricerca utilizzata consiste nell'analizzare NBR vicini per volta, scegliendo il migliore e fermandosi al primo accettato.

  Per l'ordine con cui i vicini vengono scelti da $N(s)$ sono state utilizzate due modalità:

  - considerare ciclicamente le stringhe in un ordine fisso, partendo dalla stringa y_{i+1}, dove y_i è l'ultima stringa invertita, e proseguendo in avanti.
  - estrarre a caso una variabile con probabilità uniforme $\frac{1}{|S|}$. Si è scelto di non modificare la distribuzione di probabilità dopo ogni estrazione.

  Poiché l'ordine con cui vengono valutate le stringhe incide sul risultato dell'algoritmo, per poter confrontare questo metodo risolutivo con gli altri viene fatta una media del risultato di dieci esecuzioni dello stesso con una diversa permutazione iniziale delle stringhe nel primo caso, e con una diversa inizializzazione del seme dei numeri pseudocasuali utilizzati per il sorteggio nel secondo caso.

- `ThresholdAtStep`: il valore di soglia all'iterazione k è dato da

$$t_k = \frac{(1-\alpha)}{1 + k \ \textbf{div} \ |S|}$$

  dove l'utilizzo della divisione intera permette di mantenere costante la soglia di accettazione per un numero di iterazioni pari a $|S|$. Per valori di α diversi le stringhe hanno un costo medio diverso; perciò la presenza del fattore $(1-\alpha)$ nella funzione t_k rende l'algoritmo più robusto rispetto alla variazione di α.

- test di accettazione: sono state implementate due diverse versioni del test di accettazione:

  - il vicino viene confrontato con la soluzione attuale; è il test presentato in letteratura.
  - il vicino viene confrontato con la soluzione migliore trovata fino a quel momento; dal punto di vista teorico questa modalità sembrerebbe peggiore, dal momento che se la soglia è troppo bassa è impossibile uscire da un minimo locale, tuttavia i risultati sperimentali mostrano un buon comportamento.

- `ShouldStop`: l'algoritmo termina quando sono stati provati tutti i vicini di una soluzione s senza accettarne nessuno; tuttavia per un utilizzo pratico risulta ragionevole fermare l'algoritmo dopo un numero massimo di iterazioni non miglioranti, dell'ordine di $|S|$. I test di confronto tra i diversi parametri per la taratura dell'algoritmo utilizzano anche un limite di tempo massimo.

Lo pseudocodice dell'algoritmo ottenuto è il seguente:

for $\langle\ i \in S\ \rangle$
    y_i^{sol} := 1
    y_i^{best} := 1
k := 0
while (**not** ShouldStop())
    // VALUTA IL VALORE DI SOGLIA PER QUESTA ITERAZIONE
    $t_k := \frac{1-\alpha}{1+k \ \textbf{div}\ |S|}$
    // SCEGLIE UN VICINO, CON INTORNO PROBABILISTICO O SEQUENZIALE
    y^n := SelectNeighbor()
    // VERIFICA SE IL VALORE È ENTRO LA SOGLIA
    if (Accept($z(y^n)$, t_k))
        $y^{sol} := y^n$
        if ($z(y^n) > z(y^{best})$)
            $y^{best} := y^n$
    $k := k+1$

dove la funzione che valuta una soluzione è:

$$z(y) = \alpha \cdot \text{SolveMPP}(y) - (1-\alpha) \sum_{i=1}^{|S|} \frac{1}{l_i} y_i$$

e la funzione di accettazione è nei due casi:

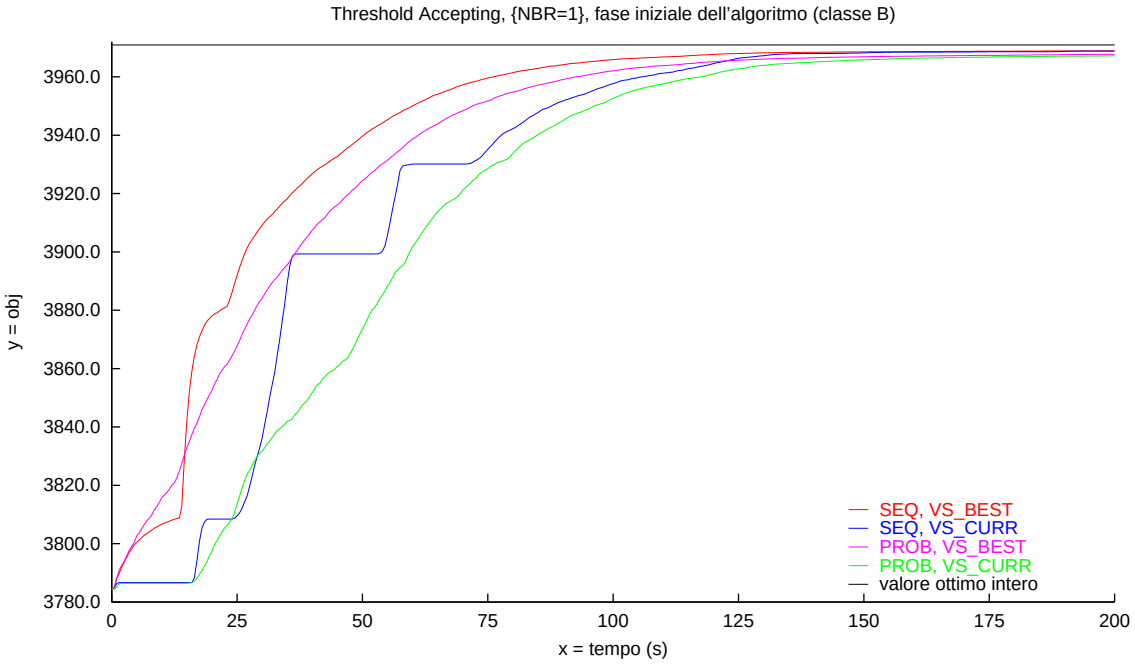
function Accept(z, t_k)
    return $\begin{cases} z > z(y^{sol}) - t_k & \text{// CONFRONTO CON LA SOLUZIONE CORRENTE} \\ z > z(y^{best}) - t_k & \text{// CONFRONTO CON LA SOLUZIONE MIGLIORE} \end{cases}$

L'elenco dei parametri che controllano il comportamento delle diverse *subroutine*, con cui verranno denominati i test, è il seguente:

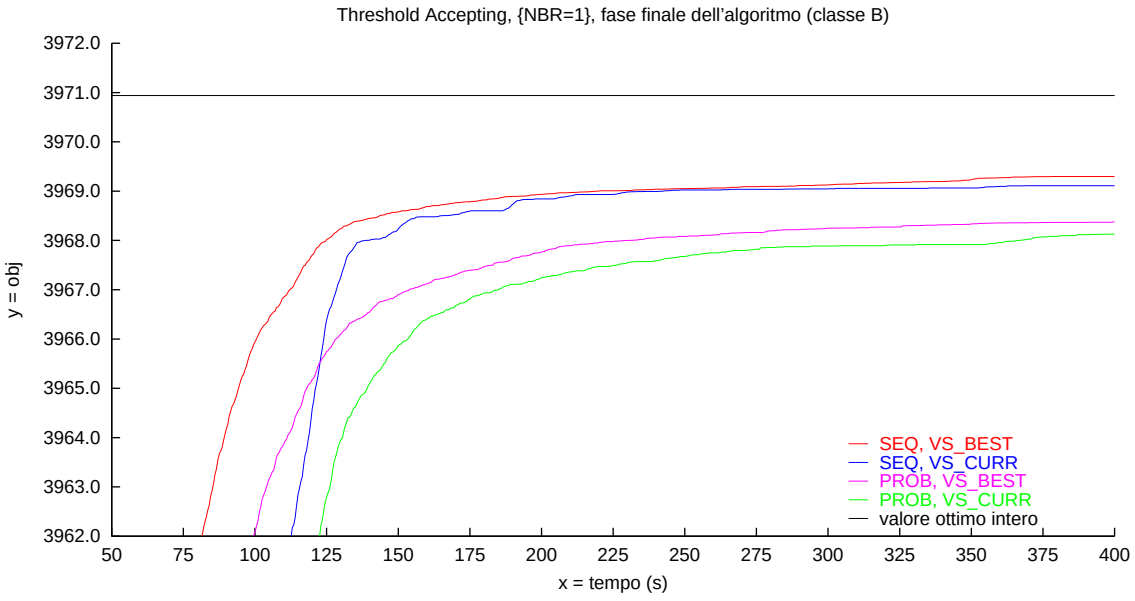
- {NBR}, numero di vicini valutati ad ogni iterazione (funzione `GetNextNeighbor`).
- {PROB | SEQ} indica il tipo di intorno usato, probabilistico o sequenziale (funzione `GetNextNeighbor`).
- {VS_CURR | VS_BEST} indica rispetto a quale soluzione viene fatto il confronto, se la soluzione corrente o quella ottima (funzione `Accept`).

5.2.2 Risultati

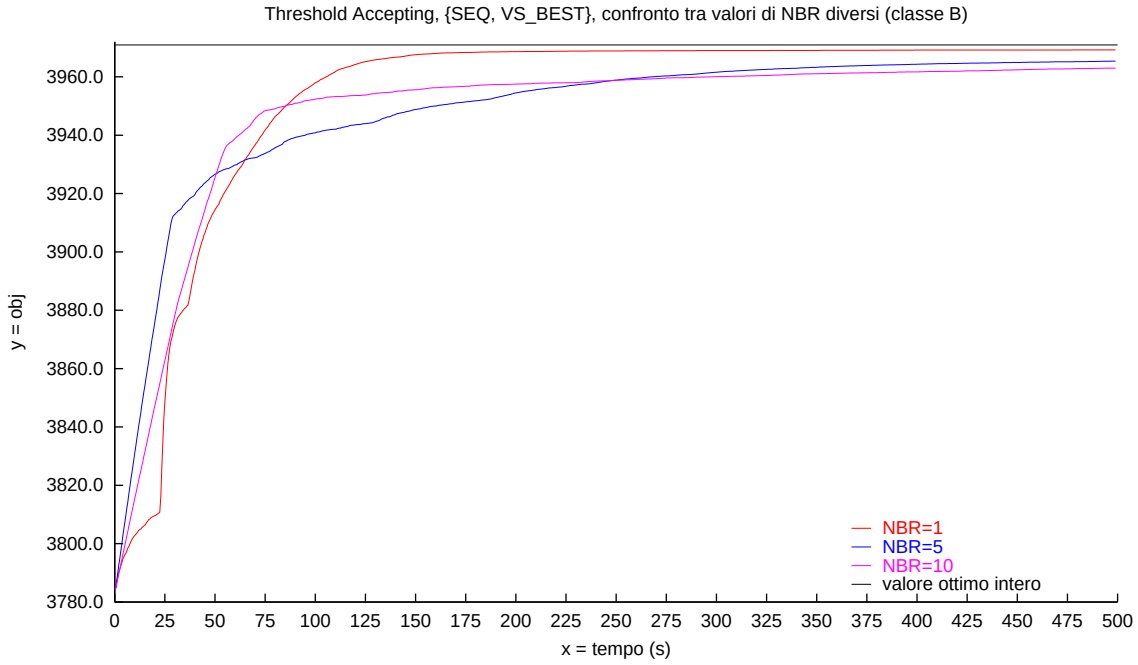
Le diverse scelte relative ai parametri dell'algoritmo vengono confrontate su un input di classe B. Il valore di *upper bound* è dato dall'ottimo intero del problema calcolato con CPLEX.



Come si può notare dal primo grafico, a parità di intorno il confronto {VS_BEST} porta a risultati medi più elevati rispetto al confronto {VS_CURR}; un possibile motivo è che cercare di mantenere le soluzioni vicine alla soluzione migliore porta a miglioramenti più frequenti; si nota che inizialmente, quando la soglia ha un valore abbastanza elevato, il confronto {VS_CURR} porta a lunghi periodi di non miglioramento (comportameno a scalini della curva {SEQ, VS_CURR}). L'intorno probabilistico {PROB} ha il vantaggio di avere un'andamento molto graduale, tuttavia in nessuno dei casi risulta migliore del relativo intorno sequenziale {SEQ}.



Nella fase finale dell'algoritmo il confronto {VS_CURR} tende allo stesso valore di {VS_BEST}. Si nota anche come entrambi gli algoritmi che utilizzano il metodo {SEQ} superino gli algoritmi che utilizzano l'intorno probabilistico {PROB}.



In questo grafico viene mostrato il comportamento dell'algoritmo *Threshold Accepting* al variare del parametro {NBR}, utilizzando l'intorno sequenziale ed il confronto con la miglior soluzione. Lo stesso test è stato ripetuto anche con i parametri {PROB} e {VS_BEST}, ottenendo risultati molto simili a quelli presentati nel grafico.

Si nota come aumentare il numero di vicini sia utile in fase iniziale, quando l'algoritmo effettua buone scelte nonostante la soglia alta, mentre in fase avanzata porti ad un comportamento meno buono, con una crescita molto ridotta. In generale si è notato che più è lenta e graduale la convergenza, migliore è il risultato finale; fare scelte troppo buone in fase iniziale porta ad entrare in minimi locali da cui è più difficile uscire, e di conseguenza ad esplorare meno bene lo spazio delle soluzioni.

Il superamento della curva con {NBR 1} da parte delle curve con {NBR} più alti, se avviene, avviene oltre i limiti di tempo utilizzati in questo grafico.

5.3 *Simulated Annealing*

Simulated Annealing è un noto algoritmo di ricerca locale appartenente. Al contrario degli algoritmi a soglia del tipo *Threshold Accepting*, in cui l'accettazione di una nuova soluzione è deterministica, in *Simulated Annealing* l'accettazione di una soluzione peggiorante è probabilistica, con probabilità che dipende da quanto è peggiorante e da un parametro T, la *temperatura*, che dà una misura della dinamicità del sistema.

Lo pseudocodice differisce da *Threshold Accepting* solo per la funzione di accettazione:

```
sol := GenerateStartingSolution()
k := 0
while ( not ShouldStop() )
    T_k := TemperatureAtStep(k)
    neighbor := SelectNeighbor(sol)
    if (  Accept(neighbor, sol, T_k)  )
        sol := neighbor
    else
        ModifyNeighborhood(sol)
    k := k+1
```

Tale funzione è la seguente:

```
function Accept(neighbor, sol, T_k)
    if ( z(neighbor) ⩾ z(sol) )
        return true
    else
        p_k = e^((z(neighbor) - z(sol)) / T_k)
        if ( ExtractUniform(0,1) ⩽ p_k )
            return true
        else
            return false
```

ovvero se la soluzione è non peggiorante viene accettata con certezza, altrimenti con probabilità p_k. Al decrescere della temperatura la probabilità di accettazione scende, mentre a temperatura fissa soluzioni più vicine alla soluzione attuale sono più probabili.

Anche in questo caso non è previsto un unico schema di variazione della temperatura, e anche in questo caso le indicazioni date in letteratura sono che si abbia $T_k \geqslant T_{k+1} \geqslant 0 \quad \forall k$, e $\lim_{k\to\infty} T_k = 0$, in modo da garantire $\lim_{k\to\infty} p_k = 0$ e quindi ottenere la garanzia di convergenza ad un minimo locale.

5.3.1 Applicazione di *Simulated Annealing* a TCSS

Rispetto all'applicazione di *Threshold Accepting* sono state fatte le seguenti modifiche:

- `SelectNeighbor`, `ModifyNeighborhood`: viene proposta la stessa politica di ricerca utilizzata nel metodo precedente ma con una modifica. Con l'accettazione deterministica è inutile riconsiderare i

vicini già valutati e scartati, perché se il migliore tra loro è stato scartato ad una iterazione lo sarà anche in quelle successive. Con l'accettazione probabilistica ha invece senso tentare il test di accettazione con il migliore tra tutti i vicini valutati dall'inizio della ricerca, poiché il vicino con valore maggiore è quello che ha più probabilità di essere accettato. Quindi l'intorno parziale valutato ad ogni iterazione non si sposta, in caso di non accettazione, ma viene ingrandito, senza ri-valutare i vicini già valutati. È quindi possibile che il test di accettazione venga ripetuto più volte sullo stesso vicino, e che eventualmente solo dopo un certo numero di tentativi questo venga accettato.

- `TemperatureAtStep`: l'algoritmo ha tre parametri da tarare:
  - la temperatura iniziale $T_{start} \in \mathcal{R}$.
  - il numero di iterazioni a temperatura costante $T_{const} \in \mathcal{N}$.
  - il parametro di *raffreddamento* $T_{cool} \in (0, 1)$.

  La temperatura ad ogni iterazione è data da

  $$T_k = T_{start} \cdot (T_{cool})^{k \textbf{ div } T_{const}}$$

  ovvero la temperatura cala ogni T_{const} iterazioni; poiché $T_{cool} < 1$ vale $\lim_{k \to \infty} T_k = 0$.

- `ShouldStop()`: per il criterio di terminazione valgono le considerazioni fatte per l'algoritmo *Threshold Accepting*, con in più l'osservazione che, essendo l'accettazione probabilistica, per terminare l'algoritmo è bene richiedere un numero di iterazioni senza miglioramento leggermente maggiore.

Anche in questo caso per ogni test viene fatta una media del risultato di dieci esecuzioni, variando il seme di inizializzazione dei numeri pseudocasuali.

Sono stati provati sia l'intorno probabilistico {PROB} che il confronto con la soluzione ottima {VS_BEST}, tuttavia anche a causa della natura probabilistica dell'algoritmo non hanno portato nessun tipo di vantaggio.

Per i test con questo metodo sono stati utilizzati i parametri {SEQ} e {VS_CURR}, provando diversi valori di {NBR}.

Sono stati fatti diversi test con differenti valori di temperatura iniziale, parametro di raffreddamento e numero di iterazioni a temperatura costante. Dal risultato di tali test ho derivato dei criteri di taratura parametrici nell'input, che vengono presentati nel paragrafo seguente.

5.3.2 Taratura dei parametri

Il corretto valore dei parametri che controllano la temperatura dipende dall'input del problema: il numero di iterazioni necessarie alla convergenza dipende dal numero di stringhe, mentre per un dato valore di temperatura la probabilità di accettazione dipende dalla differenza di valore tra due soluzioni, e quindi dal parametro α.

Si è deciso di utilizzare $T_{const} = 1$; infatti se T_{cool} è sufficientemente vicino ad 1 la temperatura varia molto poco tra un'iterazione e la successiva e non c'è la necessità di rallentarla ulteriormente.

Dal momento che la probabilità dipende dalla differenza di valore tra due soluzioni, per decidere che valore assegnare ai parametri T_{start} e T_{cool} bisogna definire quantitativamente cosa significa un peggioramento accettabile.

In fase iniziale si vuole che ogni scambio abbia una buona probabilità di essere accettato, quindi che T_{start} sia tale da avere $p_0 \geqslant \frac{1}{2}$. La soluzione di partenza è $s_0 = \{y_i = 1 \quad \forall i\}$, quindi in fase iniziale l'algoritmo analizza soluzioni in cui viene fissato $y_i = 0$ per una certa stringa i. In questa situazione un peggioramento consiste nel perdere copertura per un valore maggiore del costo della stringa tolta. La copertura di un carattere nella funzione obiettivo vale α (valore di un'occorrenza diviso la lunghezza della stringa corrispondente); quindi possiamo considerare un peggioramento accettabile $L_{MAX}\alpha$, che corrisponde a perdere copertura su tanti caratteri quanti ne copre un'occorrenza della stringa più lunga. Poniamo quindi:

$$T_{start} = \frac{-L_{MAX}\alpha}{\ln(0.5)}$$

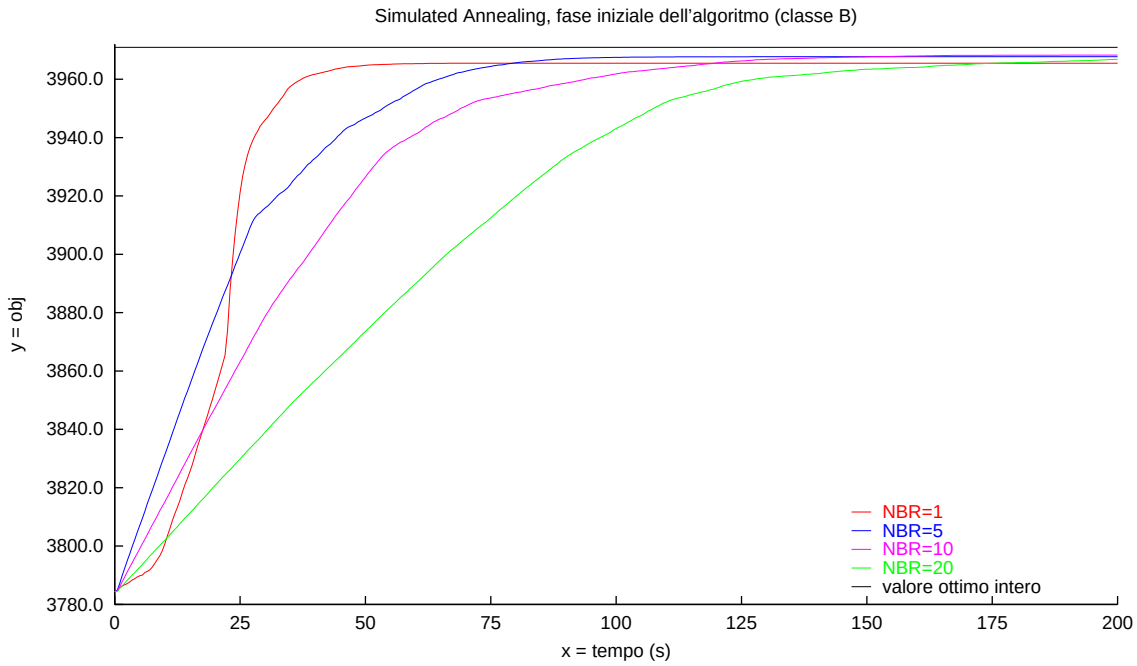
L'algoritmo dopo aver visitato una volta tutto l'insieme delle stringhe, quindi dopo $|S|$ iterazioni, si suppone abbia trovato una discreta soluzione; da questo punto in poi si può considerare di essere vicini alla convergenza. Trascorse $|S|$ iterazioni vogliamo quindi che la temperatura sia calata a sufficienza da avere, per una soluzione non molto peggiorante, una bassa probabilità di accettazione. Un piccolo peggioramento consiste nell'avere un carattere coperto in meno a parità di costo, quindi una differenza di valore di $-\alpha$, e come probabilità bassa poniamo $p_F = \frac{1}{100}$. Abbiamo quindi $T_F = \frac{-\alpha}{\ln(0.01)}$, e poiché $T_F = T_{start} \cdot (T_{cool})^{|S|}$ abbiamo $T_{cool} = \sqrt[|S|]{\frac{-\alpha}{\ln(0.01)T_{start}}}$ e sostituendo otteniamo:

$$T_{cool} = \sqrt[|S|]{\frac{\ln(0.5)}{L_{MAX}\ln(0.01)}}$$

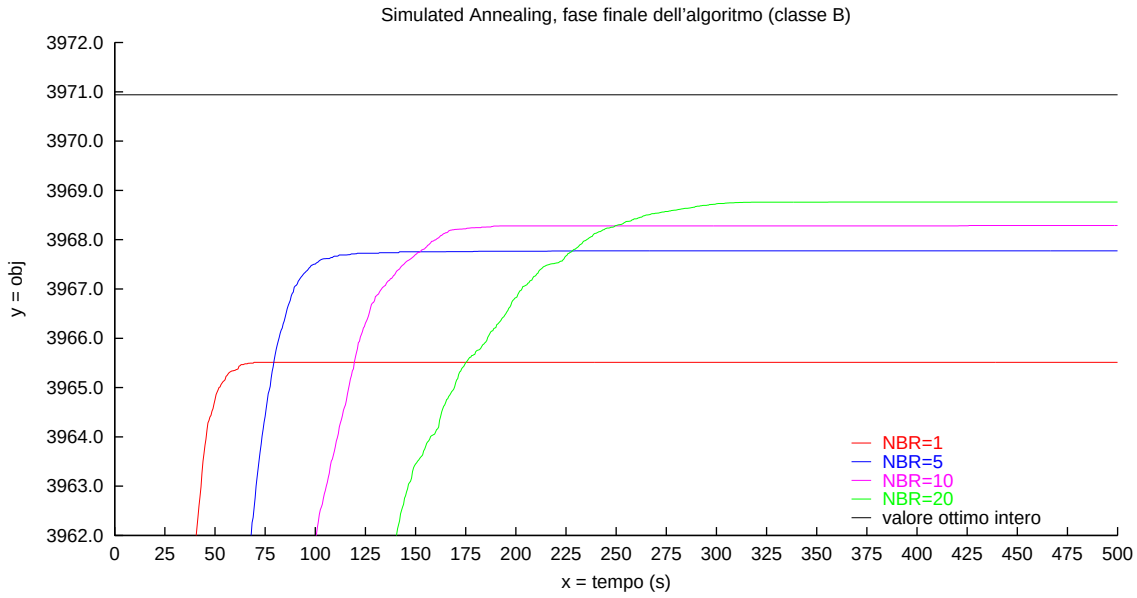
Nei nostri test $\alpha = 0.1$ e $L_{MAX} = 6$, quindi $T_{start} \approx 0.86562$. Per i test che seguono ho utilizzato un input di classe B, con 2203 stringhe, quindi $T_{cool} \approx 0.99833$. Per i test sugli input di altre classi vengono utilizzati valori di T_{cool} appropriati alla loro dimensione.

5.3.3 Risultati

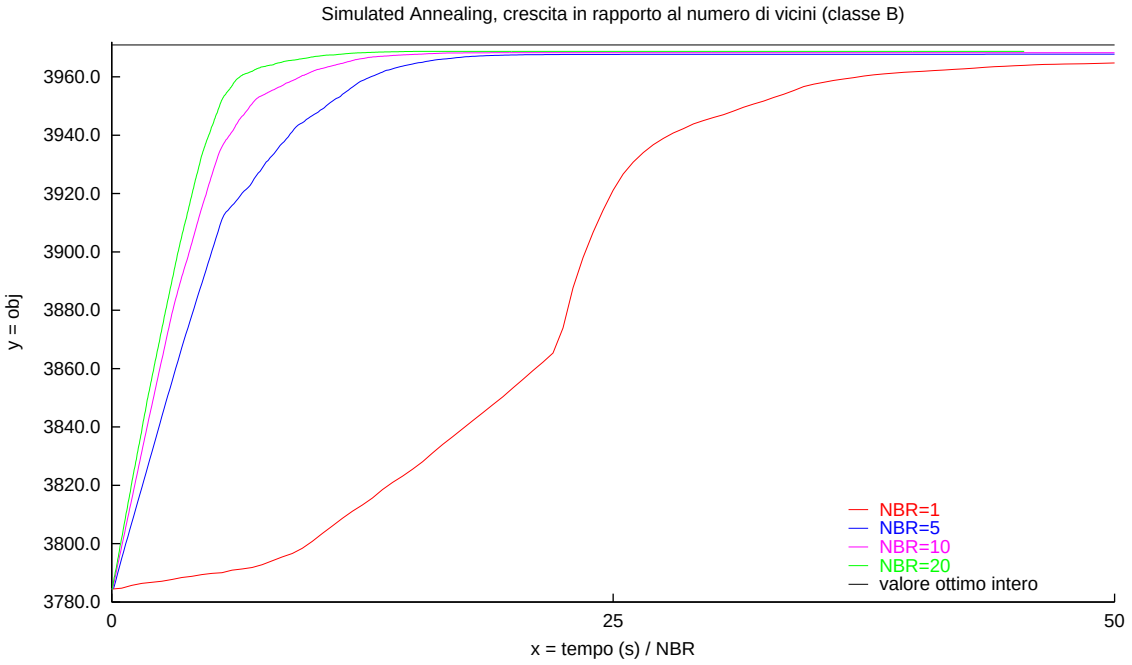
L'input di questi test è lo stesso input di classe B usato nei capitoli precedenti.



In questo grafico si può notare come il parametro {NBR 1} abbia una curva con crescita più rapida, nonostante nelle prime fasi l'alta probabilità di accettazione porti a scelte non ottimali. Scegliendo tra più vicini il migliore, come accade con gli altri parametri, anche in fase iniziale c'è una crescita costante poiché c'è una maggior probabilità di miglioramento ad ogni iterazione. Si può osservare come al crescere del parametro la crescita della curva sia più lenta.



Osservando il comportamento nella fase finale però si osserva che maggiore è il numero di vicini presi in considerazione e maggiore è il valore finale, benché con una crescita più lenta.



In questo grafico al posto del tempo si è usato il rapporto tra il tempo e NBR, per mostrare che maggiore è NBR e più veloce risulta la crescita in rapporto al numero di iterazioni.

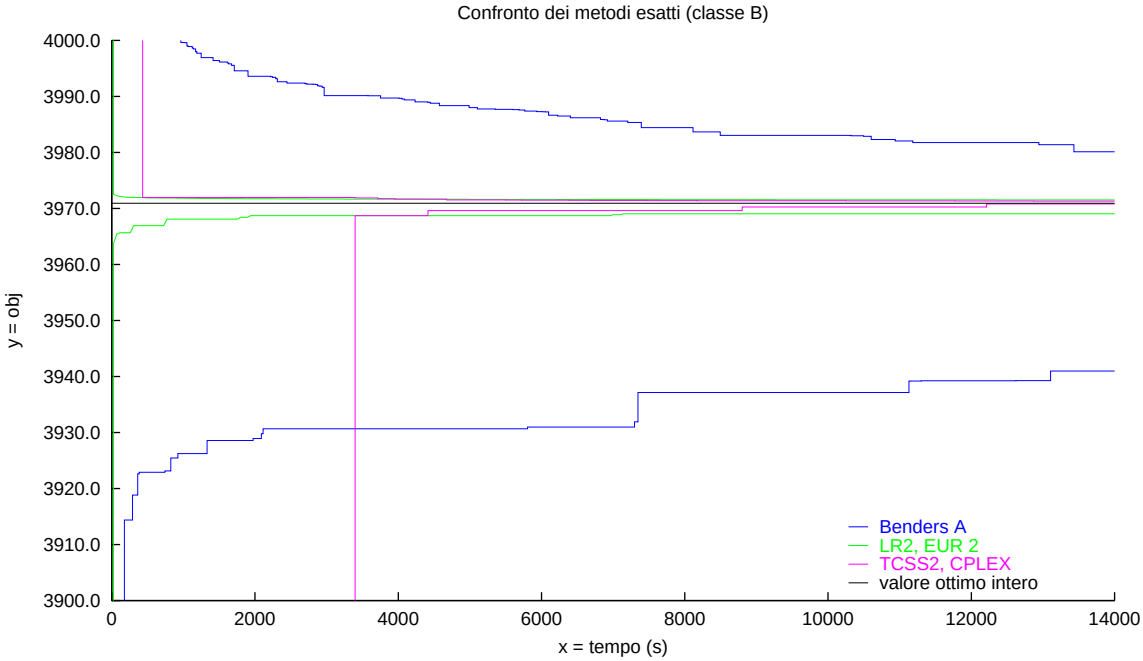
Capitolo 6

Confronto dei diversi metodi

In questo capitolo vengono presentati risultati comparativi per i diversi metodi, ciascuno con le scelte migliori dei parametri desunte dagli esperimenti riportati nei capitoli precedenti.

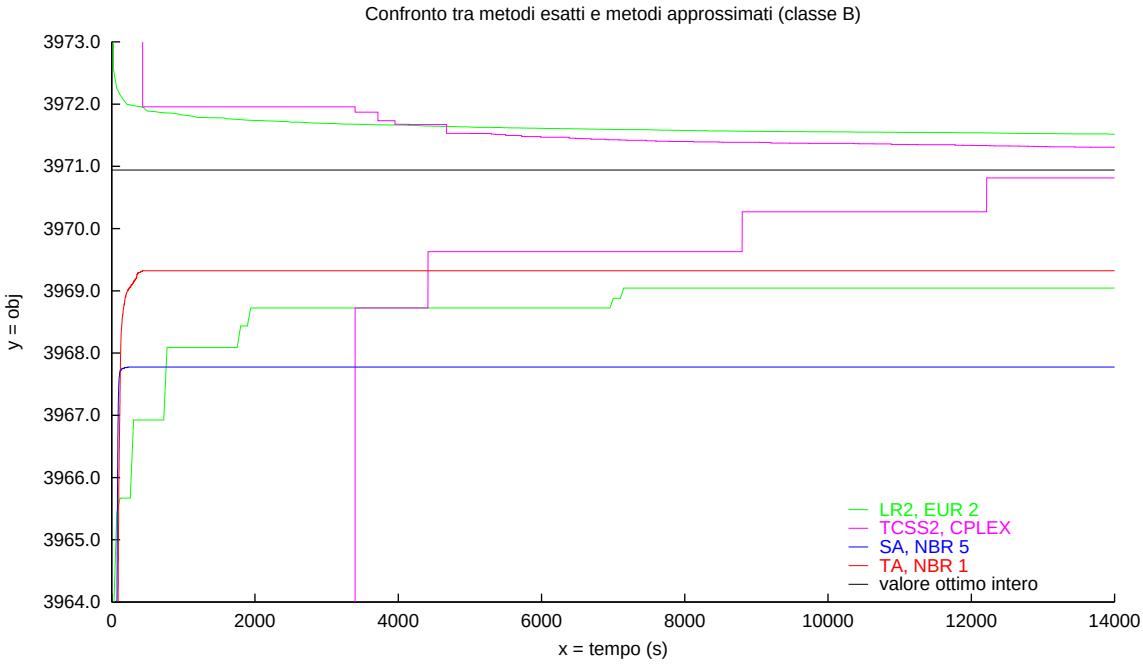
Il primo paragrafo riassume e confronta tutti i risultati degli esperimenti su un'istanza di classe B, mentre nel secondo paragrafo i diversi metodi vengono confrontati su un'istanza di classe C.

6.1 Confronto su un'istanza di classe B



In questo grafico vengono confrontati tutti i metodi applicati che risolvono il problema all'ottimo.

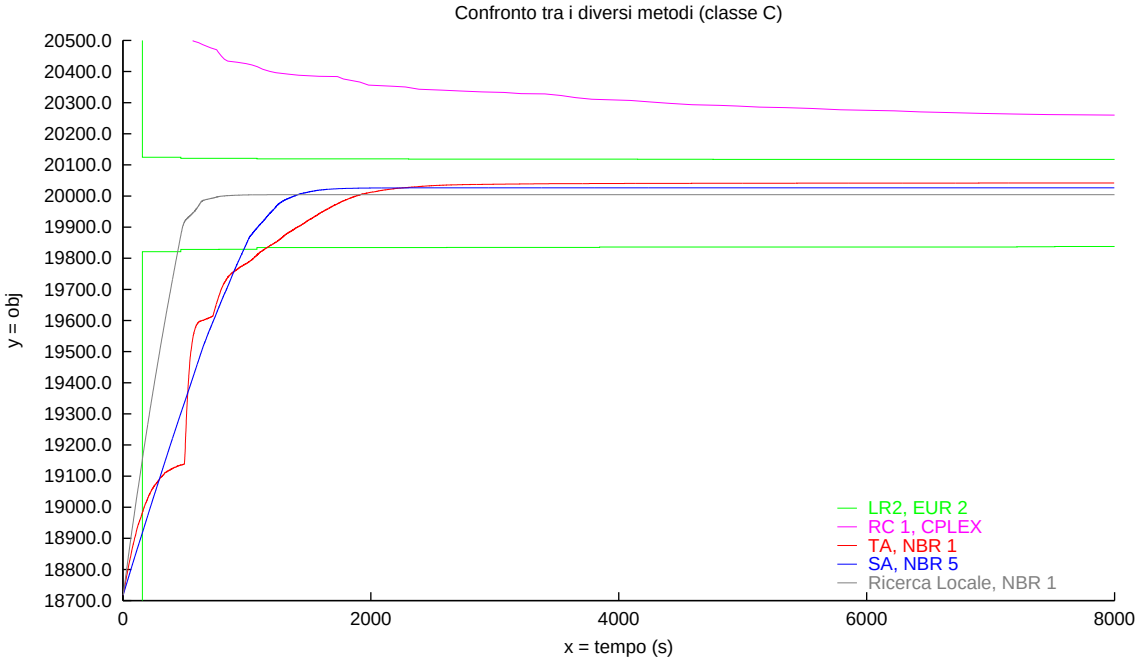
Per questa classe, il metodo esatto più efficace sul lungo periodo è la soluzione di TCSS_2 con CPLEX, anche se la prima soluzione intera viene trovata dopo più di un'ora. Il rilassamento Lagrangeano ha un buon comportamento fin dall'inizio, ma migliora lentamente il *lower bound* e di conseguenza viene superato da CPLEX. La riformulazione di Benders si rivela inefficace.



In questo grafico il metodo del rilassamento Lagrangeano e di CPLEX con il rilassamento $TCSS_2$ vengono confrontati con i metodi approssimati. La terminazione dei metodi approssimati avviene entro i dieci minuti di calcolo, ma la curva è stata prolungata per poter confrontare i risultati con i risultati dei metodi esatti. Il comportamento di *Threshold Accepting* è decisamente migliore di quello di *Simulated Annealing*, che viene superato entro 20 minuti di calcolo dal rilassamento Lagrangeano. Il valore migliore raggiunto da *Threshold Accepting* viene superato, entro i limiti di tempo del grafico, solo da CPLEX dopo più di un'ora di calcolo.

Anche rispetto ai metodi di approssimazione, per istanze di classe B o inferiore il metodo più efficace è la soluzione di $TCSS_2$ con CPLEX.

6.2 Confronto su un'istanza di classe C



Il grafico presentato per questa istanza è stato ottenuto dalla media di tre esecuzioni dei diversi algoritmi.

Il calcolo con CPLEX dell'ottimo del rilassamento continuo $\{\text{RC } 1\}$ su questa istanza non termina entro alcuni giorni di calcolo; di conseguenza non è possibile nemmeno iniziare l'ottimizzazione del problema intero. Su questa istanza, solo il rilassamento $\{\text{RC } 2\}$ riesce a terminare entro un giorno di calcolo, ma il *bound* prodotto è decisamente debole, all'incirca del 14%.

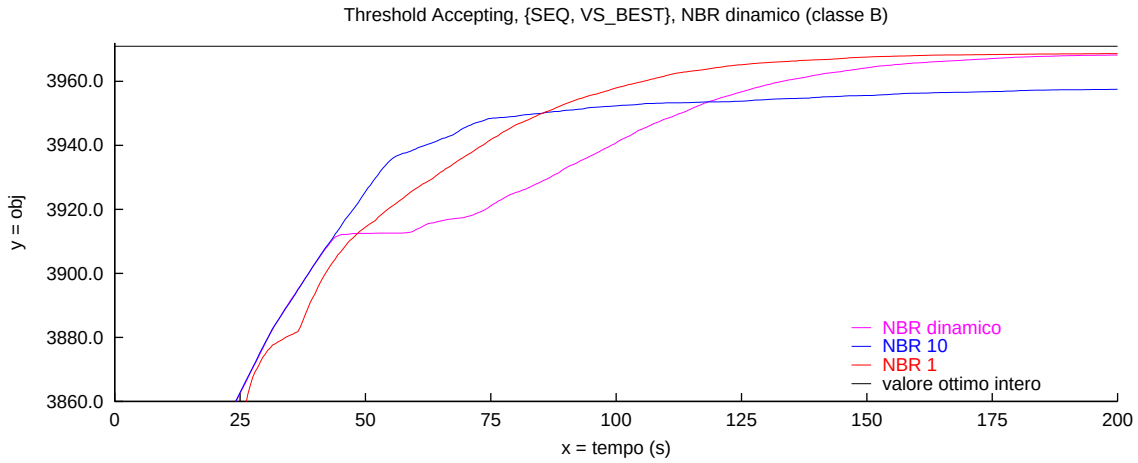
Il rilassamento Lagrangeano produce un discreto *upper bound* fin dal nodo radice, migliorandolo però molto lentamente; i *lower bound* prodotti non sono invece altrettanto buoni; l'errore di approssimazione che si ottiene è di poco inferiore all'1.4%.

Gli algoritmi di ricerca locale utilizzati riescono, in un tempo relativamente breve, a superare il miglior *lower bound* del rilassamento Lagrangeano e ad assestarsi su valori molto vicini all'ottimo, con un errore di approssimazione minore dello 0.4%. La curva $\{\text{Ricerca Locale, NBR } 1\}$ è una ricerca locale senza soglia; arriva molto rapidamente all'ottimo locale, che però ha un valore inferiore sia di quello trovato da *Simulated Annealing* che di quello trovato da *Threshold Accepting*.

Capitolo 7

Sviluppi futuri

Un primo possibile sviluppo del lavoro svolto nell'ambito di questa tesi riguarda l'inserimento degli algoritmi nel *framework* di classificazione di testi, in modo da valutare la reale utilità della formulazione studiata, al variare del parametro α. In caso di successo può essere interessante applicare la classificazione a testi in lingua italiana, su cui ci si aspettano risultati migliori di quelli su testi inglesi.



Come mostra il grafico, alcuni esperimenti preliminari in cui il valore del parametro NBR veniva fatto decrescere durante l'ottimizzazione (da 10 a 1), indicano come ulteriore direzione di sviluppo lo studio di meccanismi autoadattativi per variare dinamicamente NBR in base alla rapidità di convergenza.

Altri possibili sviluppi riguardano l'applicazione a TCSS di metodi risolutivi diversi; in particolare i metodi basati su *Column Generation* (Dantzig-Wolfe [3]) e su *Cross Decomposition* (Van Roy [4]) per la soluzione esatta del problema, e su *Tabu Search* per la soluzione approssimata.

Un ulteriore argomento di ricerca riguarda la dimostrazione di NP-completezza di TCSS.

Visto l'ottimo comportamento del rilassamento Lagrangeano rispetto all'*upper bound*, e alla capacità dei metodi di ricerca locale di ottenere buone soluzioni, potrebbe risultare interessante inserire i due metodi in un'unico algoritmo, utilizzando il *lower bound* del rilassamento Lagrangeano come soluzione iniziale per una ricerca locale, ottenendo un metodo *multistart* facilmente parallelizzabile.

Infine è possibile considerare le due formulazioni alternative presentate qui di seguito, che, trattandosi di un problema intero, possono consentire un'esplorazione più raffinata della regione paretiana di quanto non avvenga per TCSS.

Formulazione alternativa A Una prima possibile formulazione è la seguente:

$$F_A) \qquad \max z_A = \sum_{j=1}^{|O|} l_{u(j)} x_j$$

$$s.t. \begin{cases} \sum_{j=1}^{|O|} a_{tj} x_j \leqslant 1 & t = 1 \dots |W| \\ x_j - y_{u(j)} \leqslant 0 & j = 1 \dots |O| \qquad (7.1) \\ \sum_{i=1}^{|S|} \frac{1}{l_i} y_i \leqslant G \qquad (7.2) \\ x_j \in \{0,1\} & j = 1 \dots |O| \\ y_i \in \{0,1\} & i = 1 \dots |S| \end{cases}$$

In questa formulazione viene mantenuto come unico obiettivo la copertura massima del testo, imponendo un limite massimo G al costo delle stringhe (vincolo (7.2)). Applicando il rilassamento Lagrangeano ai vincoli (7.1) si ottengono due sottoproblemi distinti sui due insiemi di variabili.

Il primo è un problema di cammino massimo su grafo aciclico:

$$LR_{Ax}) \qquad \max z_{LRAx} = \sum_{j=1}^{|O|} (l_{u(j)} - \lambda_j) x_j$$

$$s.t. \begin{cases} \sum_{j=1}^{|O|} a_{tj} x_j \leqslant 1 & t = 1 \dots |W| \\ x_j \in \{0,1\} & j = 1 \dots |O| \end{cases}$$

mentre il secondo è un problema di *knapsack*:

$$LR_{Ay}) \qquad \max z_{LRAy} = \sum_{i=1}^{|S|} \left(\sum_{j \in O_S(i)} \lambda_j \right) y_i$$

$$s.t. \begin{cases} \sum_{i=1}^{|S|} \frac{1}{l_i} y_i \leqslant G \\ y_i \in \{0,1\} & i = 1 \dots |S| \end{cases}$$

Il problema di cammino massimo su grafo aciclico è risolvibile in tempo polinomiale, con l'algoritmo di programmazione dinamica presentato in 1.3.1. mentre il problema di *knapsack* è NP-completo, ma per la sua risoluzione esistono algoritmi estremamente efficaci [9] e [10].

Formulazione alternativa B Una seconda possibile formulazione è la seguente:

$$F_B) \qquad \min z_B = \sum_{i=1}^{|S|} \frac{1}{l_i} y_i$$

$$s.t. \begin{cases} \sum_{j=1}^{|O|} a_{tj} x_j \leqslant 1 & t = 1 \dots |W| \\ x_j - y_{u(j)} \leqslant 0 & j = 1 \dots |O| \qquad (7.3) \\ \sum_{j=1}^{|O|} l_{u(j)} x_j \geqslant F \qquad (7.4) \\ x_j \in \{0,1\} & j = 1 \dots |O| \\ y_i \in \{0,1\} & i = 1 \dots |S| \end{cases}$$

In questa formulazione viene mantenuto come unico obiettivo quello di minimizzare il costo associato alle stringhe, imponendo un valore minimo F alla copertura del testo (vincolo (7.4)). Applicando anche in questo caso il rilassamento Lagrangeano ai vincoli (7.3) si ottengono due sottoproblemi distinti.

Il primo è un problema di cammino ottimo su grafo aciclico con vincoli di risorsa:

$$LR_{Bx}) \qquad \min z_{LRBx} = \sum_{j=1}^{|O|} \lambda_j x_j$$

$$s.t. \begin{cases} \sum_{j=1}^{|O|} a_{tj} x_j \leqslant 1 & t = 1 \dots |W| \\ \sum_{j=1}^{|O|} l_{u(j)} x_j \geqslant F \\ x_j \in \{0,1\} & j = 1 \dots |O| \end{cases}$$

mentre il secondo è un sottoproblema banale:

$$LR_{By}) \qquad \min z_{LRBy} = \sum_{i=1}^{|S|} \left(\frac{1}{l_i} - \sum_{j \in O_S(i)} \lambda_j \right) y_i$$

$$s.t. \quad y_i \in \{0,1\} \qquad i = 1 \dots |S|$$

In questo caso tutta la complessità risiede nella soluzione del primo problema.

Bibliografia

[1] Y. Yang, X. Liu: “A re-examination of Text Categorization Methods”, *22nd Annual International SIGIR*, pp. 42-49, Berkley, August 1999

[2] D. Gusfield: “Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology”, Cambridge University Press, 1997

[3] G.B. Dantzig, P. Wolfe: Decomposition principle for linear programs, *Operations Research*, no. 8, pp. 101-111, 1960

[4] T.J. Van Roy: “Cross decomposition for Mixed Integer Programming”, *Mathematical Programming*, no. 25, pp. 46-63, North-Holland Publishing Company, 1983

[5] G.L. Nemhauser, L.A. Wolsey: “Integer and Combinatorial Optimization”, John Wiley & Sons, 1988

[6] L.A. Wolsey: “Integer Programming”, John Wiley & Sons, 1998

[7] J.E. Beasley: “Lagrangean Relaxation”, *The Management School Imperial College - Technical Report*, 1992

[8] E.H.L. Aarts, Jan K. Lenstra: “Local search in Combinatorial Optimization”, John Wiley & Sons, 1997

[9] S. Martello, P. Toth: “Knapsack Problems - Algorithms and Computer Implementations”, John Wiley & Sons, 1989

[10] D. Pisinger: “A minimal algorithm for the 0-1 knapsack problem”, *Operations Research*, 46, 5, pp. 758-767, 1995

[11] E. Balas: “An Additive Algorithm for Solving Linear Programs with Zero-one Variables”, *Operations Research*, 13, 4, pp. 517-546, 1965