ALMA MATER STUDIORUM – UNIVERSITA' DI BOLOGNA

CAMPUS DI CESENA

SCUOLA DI INGEGNERIA E ARCHITETTURA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

TITOLO DELLA TESI

ALGORITMI PER LA SOLUZIONE DEL PROBLEMA DELL'ASSEGNAMENTO GENERALIZZATO

Tesi in
Algorimi di Ottimizzazione LS

Relatore
Prof. Paolo Toth

Presentata da
Francesco Calore

Correlatori
Prof. Roberto Baldacci

Dott. Roberto Roberti

Sessione terza
Anno accademico 2012/2013

I. IL PROBLEMA CONSIDERATO

In questa tesi il problema preso in considerazione è il GAP (Generalized Assignment Problem). Si tratta di un problema di ottimizzazione combinatoria. E' una generalizzazione dell'Assignment Problem, in cui sia i lavori che le macchine hanno una capacità, la quale in generale varia da una macchina all'altra.

I.1.IL MODELLO MATEMATICO

Tale problema, originariamente formulato da Ross e Soland consiste nell'assegnare ogni lavoro dato ad una determinata macchina, con le seguenti precondizioni:

- Sono dati n lavori
- Sono fornite m macchine per svolgerli
- Si definisce $c_{i,j}$ il costo impiegato nell'assegnare alla macchina i il lavoro j (j=1,…,n e i=1,…,m)
- Si definisce $r_{i,,j}$ la quantità di risorsa utilizzata, corrispondente all'assegnamento del lavoro j alla macchina i (j=1,…,n e i=1,…,m)
- Si definisce b_i la quantità di risorsa disponibile per la macchina i (i=1,…,m)

L'obiettivo è **assegnare ogni lavoro ad una ed una sola macchina, in modo da minimizzare il costo globale**.

$$\min \quad z = \sum_{i=1}^{m}\sum_{J=1}^{n} c_{i,j}x_{i,j}$$

s.t.

$$\sum_{j=1}^{n} r_{i,j}x_{i,j} \leq b_i \qquad i=1,\ldots,m \qquad (1)$$

$$\sum_{i=1}^{m} x_{i,j} = 1 \qquad j=1,\ldots,n \qquad (2)$$

$$x_{i,j} \in \{0,1\} \quad i=1,\ldots,m \quad j=1,\ldots,n \qquad (3)$$

Alternativamente la funzione obiettivo può richiedere di massimizzare i profitti $p_{i,j}$ invece che di minimizzare i costi.

Nel caso speciale in cui tutte le risorse $r_{i,j}$ e b_i abbiano valore unitario, il problema si riduce al semplice Assignment Problem.

Il Problema dell'Assegnamento Generalizzato è un problema NP-hard.

I.2.LA LETTERATURA SUL PROBLEMA

Molti metodi di ricerca della soluzione esatta, presenti nella letteratura, sono metodi branch-and-bound che usano il rilassamento lagrangiano ottenuto dualizzando i vincoli di tipo (2). Se λ denota il vettore di moltiplicatori, allora questo rilassamento scompone il problema in m Problemi di Knapsack 0-1.

$$z(\lambda) = \sum_{J=1}^{n} \lambda_j + \min \sum_{i=1}^{m}\sum_{J=1}^{n} \tilde{c}_{i,j} x_{i,j}\,, \qquad dove \;\; \tilde{c}_{i,j} = c_{i,j} - \lambda_j$$

$$\text{s.t.} \qquad \sum_{j=1}^{n} a_{i,j} x_{i,j} \leq b_i \quad i=1,\ldots,m \qquad (1)$$

$$x_{i,j} \in \{0,1\} \; i=1,\ldots,m \quad j=1,\ldots,n \qquad (3)$$

Il bound lagrangiano duale $\max_\lambda z(\lambda)$ così ottenuto è forte almeno quanto il bound del rilassamento lineare, che equivale ai bound lagrangiani duali ottenuti dualizzando o i vincoli di Knapsack, o sia quelli di knapsack che quelli di assegnamento.

Ci sono poi il procedimento di Haddadi e Ouzia(2004), e quello di Nauss(2003), i quali dualizzano i vincoli di assegnamento, nei loro metodi lagrangiani di branch and bound. Haddadi e Ouzia offrono una procedura per perfezionare le soluzioni del rilassamento lagrangiano ottenuto durante la ricerca del subgradiente all'interno delle soluzioni ammissibili, in modo tale da avere la possibilità di migliorare l'upper bound. Nauss presenta un metodo ibrido lineare-lagrangiano che si serve della generazione di problemi di Knapsack. Savelsbergh(1997) introduce un metodo Branch and

Price, in cui le colonne generate corrispondono ad assegnamenti ammissibili per una macchina. Pigatti e altri(2005) accelerano la convergenza di questo metodo, proponendo uno schema per stabilizzare i moltiplicatori durante la fase di generazione delle colonne. Il metodo proposto da Avella ed altri(2010) è basato sul Branch and Cut λ su uno schema di generazione di Knapsack cover cut. Il metodo di Posta e altri (2011), che si propone il miglioramento delle prestazioni di quello di Avella ed altri, e che maggiormente approfondiremo in questo lavoro, riduce il problema di ottimizzazione in una sequenza di problemi decisionali. Si computa un valido lower bound per il GAP, e poi si ricercano soluzioni ammissibili con il valore del lower bound. Se non si trova una soluzione, si incrementa il lower bound e si itera il procedimento. Si va avanti fino a che non si trova una soluzione ammissibile, che è chiaramente quella ottima. Naturalmente è importante che la differenza tra il lower bound iniziale e la soluzione ottima sia sufficientemente piccola. Sono stati inoltre proposti diversi algoritmi euristici: Yagiura ed altri(2004) introducono un metodo metaeuristico, basato su una "catena di vicinanza" e Diaz e Fernandez(2001) propongono un semplice ed efficace metodo tabu search.

I.3 IL METODO DI HADDADI E OUZIA(2004)

Il metodo di Haddadi e Ouzia si basa su un algoritmo branch-and-bound per risolvere il problema dell'assegnamento generalizzato. Viene fornito un Lower Bound con un classico metodo subgradiente, usato ad ogni nodo dell'albero decisionale per risolvere il duale lagrangiano. Viene poi applicata una nuova euristica ad ogni iterazione del metodo subgradiente, che prova a sfruttare la soluzione del problema rilassato, risolvendo un GAP più piccolo. La soluzione ammissibile così trovata viene data in input ad una euristica di miglioramento. Il nodo radice viene processato con un'euristica lagrangiana.

I.4 IL METODO DI NAUSS(2003)

Nauss fornisce un metodo specializzato di branch-and-bound che utilizza tagli ottenuti con la programmazione lineare, generatori di soluzioni ammissibili, rilassamenti lagrangiani e ottimizzazione subgradiente. Un beneficio di questo algoritmo è quello di generare buone soluzioni

ammissibili in meno tempo dei precedenti, di qualità nettamente superiore rispetto alle precedenti euristiche formulate. Inoltre, il tempo di computazione richiesto è spesso meno del tempo impiegato dalle suddette euristiche. Il risultato è quindi un algoritmo di ottimizzazione che può essere usato effettivamente come un'euristica, quando la prova dell'ottimalità non è la richiesta più importante.

I.5 IL METODO DI SAVELSBERGH (1997)

Savelsbergh presenta un algoritmo che impiega sia la generazione di colonne che il branch-and-bound per ottenere soluzioni ottime intere per una formulazione Set Partitioning del Problema. Vengono definite varie strategie di branching, che permettono la generazione di colonne ad ogni nodo dell'albero branch-and-bound. L'algoritmo può essere visto come un branch-and-price, simile agli algoritmi branch-and-cut che consentono la generazione di righe ad ogni nodo dell'albero.

I.6 IL METODO DI PIGATTI ED ALTRI (2005)

Il metodo di Pigatti è un miglioramento di quello di Savelsbergh, basato su un meccanismo di stabilizzazione che accelera la convergenza della generazione di colonne. Propone inoltre tagli ellissoidali, un nuovo metodo per trasformare l'algoritmo esatto in una potente euristica. La soluzione trovata da questa euristica facilità il ritrovamento della soluzione esatta. Questo metodo ha ottime prestazioni ed ha risolto alcune tra le istanze che erano ancora aperte nella libreria della ricerca operativa.

I.7 IL METODO DI AVELLA ED ALTRI (2010)

Avella formula un metodo basato sui piani di taglio, che lavora nello spazio di variabili della formulazione di base, che si basa sulla procedura esatta di separazione dei politopi di Knapsack, indotta dai vincoli di capacità. Una efficiente implementazione di questa separazione permette di risolvere istanze su larga scala, e di risolvere diverse istanze precedentemente irrisolte. La procedura di separazione è integrata nello schema branch-and-cut. Un grande vantaggio di questo metodo è quello di

lavorare con la naturale formulazione del problema, contenendo solo le variabili iniziali. Questo facilita l'applicazione dell'algoritmo dei piani di taglio e permette di usare i classici branch-and-cut.

I.8 IL METODO EURISTICO DI YAGIURA ED ALTRI (2004)

Questo algoritmo combina un approccio a path relinking con una programmazione a memoria adattativa (tabu search); viene fornito un meccanismo "evoluzionistico" per generare nuove soluzioni a partire da due o più soluzioni di riferimento. Il path relinking è un'estensione della ricerca locale. La ricerca locale parte da una soluzione data e ne ricerca una migliore nelle sue vicinanze. Il path relinking utilizza invece un approccio a catena di espulsioni, associata ad una tabu search, in cui il rilassamento lagrangiano fornisce informazioni sui costi ridotti per guidare la ricerca di soluzioni vantaggiose nelle vicinanze. Inoltre viene fornito un meccanismo automatico per migliorare i parametri di ricerca, per mantenere un bilanciamento delle visite delle regioni ammissibili e non ammissibili.

I.9 IL METODO EURISTICO DI DIAZ E FERNANDEZ (2001)

Il metodo introduce un meccanismo di λ-generazione e, dopo un'investigazione su differenti strategie di ricerca e impostazioni dei parametri, viene applicata una ricerca tabu-search. Il metodo sviluppato racchiude un numero di caratteristiche che si sono dimostrate utili per ottenere soluzioni ottime e vicine all'ottimo. L'efficacia di questo approccio sta nel paragonare le prestazioni ottenute a quelle degli altri alberi di ricerca specializzati branch-and-bound, delle euristiche set partitioning e altre, su molti dei problemi standard della letteratura.

II. L'ALGORITMO DI POSTA ED ALTRI(2011)

Posta propone un algoritmo esatto per risolvere il Problema dell'Assegnamento Generalizzato. Il contributo è duplice: viene riformulato il problema di ottimizzazione in una serie di problemi decisionali; dopo di che vengono fissate le variabili, applicando le regole di seguito formulate, per risolvere questi problemi. I problemi decisionali sono risolti con un semplice metodo branch and bound lagrangiano depth-first. Questo metodo viene migliorato mediante le regole per fissare le variabili, per ridurre l'albero di ricerca.

Con l'algoritmo di Posta ed altri viene computato un Lower Bound valido per il GAP (nel modo descritto precedentemente), dopo di che si va alla ricerca di una soluzione ammissibile col valore del Lower Bound. Se non si riesce a trovare tale soluzione, viene incrementato il Lower Bound e ripetuta la procedura. Si procede fino a quando non si è trovata una soluzione ammissibile. La soluzione data è ottima per costruzione; naturalmente, l'algoritmo è efficiente solo se c'è una piccola differenza tra il Lower Bound iniziale e la soluzione ottima.

II.1 PANORAMICA SULL'ALGORITMO

L'approccio di questo algoritmo consiste nel, dato z* Lower Bound sulla soluzione ottima del problema, risolvere il problema decisionale sull'esistenza di una soluzione ammissibile di costo z* o inferiore. Se questa soluzione esiste, abbiamo trovato la soluzione ottima, altrimenti dobbiamo ripetere il procedimento, incrementando di 1 la nostra z*. E' un algoritmo molto semplice che si basa sul fatto che la soluzione è a valori interi, e questo ci permette di trasformare il problema in uno di tipo decisionale. Viene usato il rilassamento lagrangiano visto in precedenza per computare il Lower Bound iniziale, ma anche per computare i Lower Bound per i nodi del metodo branch and bound, per risolvere i problemi decisionali

Il metodo branch and bound per trovare la soluzione ammissibile:

- Inizializza la coda dei nodi attivi con il nodo radice

- Se la coda è vuota, il problema decisionale ha risposta negativa, altrimenti viene selezionato e rimosso un nodo dalla coda
- Si risolve il lagrangiano duale corrispondente al nodo per computare il Lower Bound del nodo stesso. Se si ottiene una soluzione duale ammissibile e inferiore o uguale a z*, la ricerca è terminata e il problema decisionale ha risposta affermativa
- Se il Lower Bound supera l'upper bound z*, si prende dalla coda il nodo successivo
- Si chiama la procedura per il fissaggio delle variabili
- Viene selezionato il nodo j con il massimo moltiplicatore λ_j tale che non tutti gli $x_{i,j}$ siano fissati, e si creano tanti branch quante sono le macchine non fissate, così da costruire fino a m nodi figli. Vengono aggiunti i figli alla coda di nodi attivi, per poi selezionare il primo nodo della coda

E' un approccio che nella pratica funziona molto bene. Secondo gli autori, questo è dovuto innanzitutto alla bontà del Lower Bound iniziale, che permette di rendere molto breve la sequenza di problemi decisionali, ma anche alla procedura del fissaggio delle variabili, una componente chiave dell'algoritmo che velocizza molto significativamente la risoluzione degli stessi problemi decisionali. Infatti viene utilizzata l'implicazione logica per fissare le variabili al valore ottimo, nel corrente nodo per il corrente problema, dato l'upper bound globale z*e con l'utilizzo delle informazioni fornite dal rilassamento lagrangiano. A questo punto sarà fondamentale un'adeguata implementazione di questa procedura, per accelerare la risoluzione dei successivi problemi. Questo a maggior ragione qualora il gap con la soluzione ottima sia esiguo, perché la nostra sequenza di problemi decisionali sfrutta questa caratteristica con vantaggio.

II.2 IL FISSAGGIO DELLE VARIABILI

Le procedure per il fissaggio delle variabili vengono utilizzate in molti algoritmi risolutori basati su rilassamenti lineari. La procedura di Posta ed altri sfrutta principi simili adattati al nostro rilassamento lagrangiano per il GAP. Per ogni nodo dell'albero branch and bound si denota λ^* il vettore dei migliori moltiplicatori trovati nella ricerca di $\max_\lambda z(\lambda)$, Δ il gap locale

con il lower bound ($\Delta = z^* - z(\lambda^*)$), $x(\lambda^*) \in \{0, 1\}^{mn}$ il valore ottimo per il rilassamento lagrangiano utilizzando i moltiplicatori λ^*. A ciascun nodo, alcuni dei componentin del vettore x devono essere fissati a 0 o a 1, attraverso il branching oppure attraverso il fissaggio delle variabili. Per ogni termine x_{ij} non fissato, se fissiamo x_{ij} al suo valore $x(\lambda^*)_{ij}$, allora $z(\lambda^*)$ non cambia; al contrario, se x_{ij} viene fissato al suo valore complementare $1 - x(\lambda^*)_{ij}$, allora $z(\lambda^*)$ può aumentare. Se supera z^*, possiamo concludere che non ci sono soluzioni ottime con x_{ij} fissato al valore complementare, e di conseguenza possiamo fissare con certezza x_{ij} al valore $x(\lambda^*)_{ij}$.

Computare nuovamente $z(\lambda^*)$ dopo aver fissato ogni singolo componente x_{ij} può sembrare irragionevole dal punto di vista computazionale. Bisogna comunque tenere presente che abbiamo un vettore di costi ridotti $c(\lambda^*)$ associato a $x(\lambda^*)$. Un costo ridotto $c(\lambda^*)_{ij}$ associato alla variabile $x(\lambda^*)_{ij}$ è un Lower Bound sull'aumento del valore obiettivo del rilassamento, quando x_{ij} è fissato al suo valore complementare. Di conseguenza, per ogni variabile x_{ij} non fissata, se $x(\lambda^*)_{ij} = 0$ e $c(\lambda^*)_{ij} > \Delta$, xpuò essere fissato a 0. Se invece $x(\lambda^*)_{ij} = 1$ e $c(\lambda^*)_{ij} > \Delta$, allora $x(\lambda^*)_{ij}$ può essere fissato a 1 e x_{kj} può essere fissato a 0 per ogni $k \neq i$. Queste regole possono essere viste come semplici principi per il fissaggio delle variabili. Si possono usare regole più potenti sfruttando il fatto che ogni lavoro deve essere assegnato ad uno ed una sola macchina, e che il rilassamento lagrangiano semplifica il problema in m problemi di Knapsack indipendenti. Considerando ogni variabile x_{ij} non fissata:

- Supponendo che $x(\lambda^*)_{ij} = 0$ e $\sum_{k=1}^{m} x(\lambda*)_{kj} \geq 1$, se x_{ij} dovesse diventare 1 in una soluzione ammissibile, a quel punto tutti gli altri x_{kj} dovrebbero essere posti a 0; pertanto se $c(\lambda^*)_{ij} + \sum_{k=1}^{m} c(\lambda*)_{kj}$, allora x_{ij} può essere fissato a 0.
- Supponendo invece che $x(\lambda^*)_{ij} = 1$ e $x(\lambda^*)_{kj} = 0$ per ogni $k \neq i$, se x_{ij} dovesse diventare 0 in una soluzione ammissibile, allora qualche x_{kj} ($k \neq i$), tra quelli non ancora fissati a 0, dovrebbe prendere il valore 1; e quindi se $c(\lambda^*)_{ij} + \min\{ c(\lambda^*)_{ij} \mid x_{ij}$ non è fissato, $i \in \{1, ..., m\} \} > \Delta$, allora il nodo corrente può essere superato.
- Supponendo che x_{kj} sia fissato a 0 per ogni $k \neq i$, ovviamente x_{ij} può essere fissato a 1, e a quel punto se $x(\lambda^*)_{ij} = 0$ e $c(\lambda^*)_{ij} > \Delta$, il nodo corrente può essere superato.

E’ possibile inoltre rendere il lower bound tanto forte da esplorare il nodo corrente. Prendendo ora in considerazione ogni lavoro j:

- se $\sum_{k=1}^{m} x(\lambda *)_{ij} = 0$, almeno una di queste variabili x_{ij} dovrà prendere il valore 1; pertanto se $\min\{ c(\lambda^*)_{ij} \mid x_{ij}$ non è fissato, $i \in \{1, ..., m\} \} > \Delta$, allora il nodo corrente può essere superato.
- se x_{ij} è fissato a 0 per ogni i, ovviamente il nodo corrente può essere superato .

Con queste regole l’algoritmo lavora molto bene: ridurre il campo alle regole che fissano le variabili al loro corrente valore $x(\lambda^*)$ permette di applicarle iterativamente a tutte le variabili non fissate, senza dover ricalcolare $x(\lambda^*)$ e $c(\lambda^*)$.

II.3 I COSTI RIDOTTI

L’algoritmo che computa i costi ridotti si basa su quello introdotto da Karabakal, ma mentre in quell’algoritmo i risultati venivano usati all’interno di un’euristica molto calante per risolvere il lagrangiano duale, questo algoritmo computa i costi ridotti per fissare le variabili nella ricerca branch-and-bound. Ecco come funziona l’algoritmo di Karakabal, partendo per ipotesi dal nodo radice, e con nessuna variabile ancora fissata: innanzitutto quando risolviamo il rilassamento lagrangiano per un vettore di moltiplicatori λ, possiamo scomporre il problema in m sottoproblemi di Knapsack indipendenti tra loro. Chiamando $k^i(\lambda)$ la soluzione ottima dell’i-esimo problema di Knapsack, associato ai moltiplicatori λ:

$$z(\lambda) = \sum_{J=1}^{n} \lambda_j + \sum_{i=1}^{m} k(\lambda)$$

$$k^i(\lambda) = \min \sum_{J=1}^{n} (c_{ij} - \lambda_j) x_{ij}$$

$$\text{s.t.} \qquad \sum_{j=1}^{n} a_{i,j} x_{i,j} \leq b_i \qquad i=1,\ldots,m$$

$$x_{i,j} \in \{0,1\} \quad i=1,\ldots,m \quad j=1,\ldots,n$$

Si definisce $x(\lambda)$ la soluzione ottima del rilassamento lagrangiano, $k^i(\lambda, x_{i,j} = 0)$ il valore ottimo del sottoproblema di Knapsack, quando una variabile libera $x_{i,j}$ viene fissata a 0, e $k^i(\lambda, x_{i,j} = 1)$ il valore ottimo quando viene fissata a 1. Il fissaggio della variabile $x_{i,j}$ al suo valore in $x(\lambda)$ non cambia $z(\lambda)$ (è la soluzione ottima del rilassamento lagrangiano), mentre fissarla al valore

complementare potrebbe incrementare la funzione obiettivo del problema di Knapsack i-esimo da $k^i(\lambda)$ a $k^i(\lambda, x_{i,j} = 1 - x(\lambda)_{i,j})$. E' di conseguenza possibile definire il vettore dei costi lagrangiani ridotti come il vettore di queste variazioni:

$$c(\lambda)_{ij} = \begin{cases} k^i(\lambda, x_{ij} = 0) - k^i(\lambda) & se\ x_{ij} = 1 \\ k^i(\lambda, x_{ij} = 1) - k^i(\lambda) & se\ x_{ij} = 0 \end{cases}$$

A questo punto i vari $k^i(\lambda)$ per ogni i sono già noti, per cui computare $c(\lambda)$ richiede solo la computazione dei k^i dei valori complementari, per ogni i e j. Il metodo di Posta ed altri lo fa in modo efficiente con l'utilizzo della programmazione dinamica.

II.4 LA PROGRAMMAZIONE DINAMICA

Dato che i coefficienti a_{ij} sono tutti interi, per risolvere ogni sottoproblema di Knapsack associato all'i-esima macchina, è possibile utilizzare un approccio a programmazione dinamica. Per ogni macchina i, viene definito uno stato iniziale s^i ed uno stato finale t^i e gli stati $v_b^{i,j}$ associati ad ogni lavoro j e ad ogni possibile utilizzo della risorsa b. A quel punto si ha il seguente meccanismo di transizione, dato un vettore di moltiplicatori λ. Considerato ogni singolo stato $v_b^{i,j}$:

- sono permesse le transizioni allo stato $v_b^{i,j+1}$ a costo 0
- sono permesse le transizioni allo stato $v_{b+a_{ij}}^{i,j+1}$ a costo $c_{ij} - \lambda_j$, se $b+a_{ij} < b_i$

Sono inoltre permesse tutte le transizioni uscenti da s^i e quelle entranti in t^i a costo zero. Lo spazio degli stati per il problema di Knapsack relativo alla macchina i-esima, parametrizzato dai moltiplicatori λ, può essere rappresentato come un grafo aciclico diretto e pesato. La rappresentazione di un sottoproblema di Knapsack come uno shortest-path problem dimostra che il valore ottimo $k^i(\lambda)$ è la lunghezza del cammino più breve da s^i a t^i. Definiti $f^i(\lambda, j, b)$ la lunghezza del cammino minimo da s^i a $v_b^{i,j}$ e $g^i(\lambda, j, b)$ la lunghezza del cammino minimo da $v_b^{i,j}$ a t^i, viene seguito il principio di ottimalità di Bellman al fine di esprimere i valori ricorsivamente:

$$f^i(\lambda,j,b)= \begin{cases} 0 & se\ j = 1 \\ fi(\lambda, j-1, b) & se\ b < a_{ij} \\ \min \begin{cases} fi(\lambda, j-1, b) \\ fi(\lambda, j-1, b-a_{ij}) + c_{ij} - \lambda_j \end{cases} & altrimenti \end{cases}$$

$$g^i(\lambda,j,b)= \begin{cases} 0 & se\ j = n+1 \\ gi(\lambda, j+1, b) & se\ b > b_i - a_{ij} \\ \min \begin{cases} gi(\lambda, j+1, b) \\ gi(\lambda, j+1, b+a_{ij}) + c_{ij} - \lambda_j \end{cases} & altrimenti \end{cases}$$

Si nota facilmente come si può ottenere $k^i(\lambda)$ mediante la computazione di f^i o g^i con l'utilizzo dell'approccio a programmazione dinamica. Questo approccio conduce automaticamente alla soluzione del problema di Knapsack per ogni i, dati i moltiplicatori λ con i quali una variabile x_{ij} è stata fissata ad uno specifico valore (ad esempio $k^i(\lambda, x_{ij} = 0)$). Fissare una variabile a 1 consiste nel rimuovere un sottoinsieme degli archi dallo spazio degli stati, in particolare gli archi ($v_b^{i,j}$, $v_b^{i,j+1}$) per ogni b. Fissare una variabile a 0 consiste invece nel rimuovere gli archi ($v_b^{i,j}$, $v_{b+a_{ij}}^{i,j+1}$) per ogni b. Il grafo è topologicamente ordinato, ed è relativamente semplice computare la lunghezza del nuovo cammino minimo. Infatti, secondo il principio di ottimalità di Bellmann, abbiamo:

$\kappa^i(\lambda, x_{ij} = 0) = \min_{0 \le b \le bi} f^i(\lambda, j,b) + g^i(\lambda, j+1,b)$

$\kappa^i(\lambda, x_{ij} = 1) = c_{ij} - \lambda_j + \min_{0 \le b \le bi-aij} f^i(\lambda, j,b) + g^i(\lambda, j+1,b+a_{ij})$.

Di conseguenza i costi ridotti possono essere trovati in $O(m\ n\ b_i)$.

III IL PROGRAMMA

Un problema di grande importanza in questo algoritmo è la risoluzione efficiente dei problemi di Knapsack, i quali vengono fuori dalla scomposizione del rilassamento lagrangiano del GAP. Il rilassamento è utilizzato in due occasioni: nella risoluzione del lagrangiano duale, e nella computazione dei costi ridotti. Nel secondo caso viene utilizzata la programmazione dinamica vista in precedenza, mentre nel primo si utilizza un algoritmo più sofisticato, MINKNAP, sviluppato da Pisinger(1997). E' una scelta fondamentale per riuscire a risolvere le istanze più difficili in cui i coefficienti sono strettamente correlati. Nella letteratura precedente tutti gli algoritmi di risoluzione utilizzavano invece un semplice branch and bound o la programmazione dinamica, ad eccezione dell'algoritmo di Avella.

Inoltre questo algoritmo risolve il duale lagrangiano con un metodo di bundle, in particolare quello di Frangioni(1996), mentre tutti gli algoritmi precedenti utilizzavano la procedura subgradiente. Il metodo bundle è molto robusto e richiede meno aggiustamenti di parametri rispetto al metodo subgradiente.

Viene utilizzata una strategia depth first per selezionare i nodi, in modo da trovare una soluzione ammissibile il prima possibile. Quando si valuta il LB in tutti i nodi, il metodo bundle è inizializzato con i migliori moltiplicatori trovati per il nodo genitore, e viene iterato per un numero fissato di volte; per il nodo radice si utilizzano i moltiplicatori ottenuti nella computazione del lower bound iniziale, evitando di fare la stessa operazione due volte.

Il programma computa il lower bound iniziale risolvendo il rilassamento lineare del modello MIP dell'istanza con CPLEX. A quel punto inizializza il metodo bundle con i moltiplicatori migliori di Knapsack del rilassamento lineare, e viene fatto iterare con un limite di iterazioni anche superiore al milione.

L'algoritmo è scritto in C ma utilizza il codice C++ del bundle e quindi va compilato con un compilatore C++. L'algoritmo utilizza delle librerie per il tempo e per la gestione delle risorse che non sono compatibili con Windows, quindi e necessario compilarlo ed eseguirlo su un sistema di

tipo UNIX; inoltre l'algoritmo utilizza al suo interno le librerie BLAS e LAPACK per le operazioni tra matrici e la risoluzione di sistemi lineari. Sarà pertanto necessario scaricare ed installare in precedenza queste due librerie, e installare un compilatore FORTRAN (gfortran o g77) necessario per compilare queste librerie scritte parzialmente anche in codice FORTRAN. Il programma richiede in input a linea di comando il percorso da cui prendere il file contenente l'istanza, l'eventuale upper bound noto (-1 se non presente), il massimo numero di iterazioni per trovare il lower bound del nodo (in questa tesi è stato usato il valore 1000), il time limit in secondi (in questa tesi è stato usato il valore 3600, ovvero un'ora) e un parametro che indica il dettaglio dei messaggi di log del programma (0 per avere un log completo).

Il programma fornisce in output il LB al nodo radice, con il tempo impiegato per calcolarlo e il numero di iterazioni fatte, la soluzione ottima al termine dell'esecuzione, se trovata, con il tempo totale di calcolo, il numero di nodi esplorati e di iterazioni totali effettuate, la soluzione migliore trovata al time limit, se questo scatta, con il gap% tra upper bound e lower bound (100*LB/UB), che nelle tabelle viene trasformato in (UB-LB)/UB per semplicità di valutazione.

Il programma si suddivide nei file bb.c, bundle.c, instance.c, knapsack.c, lagrelax.c, main.c, minknap.c; ognuno dei file C, ad eccezione del main, ha il relativo header file .h.

III.1 IL MAIN

Il main carica l'istanza presa dal file nel percorso dato in input, richiamando la funzione di caricamento del file instance.c, crea il rilassamento lagrangiano richiamando le funzioni di creazione del file lagrelax.c, richiama la funzione per risolvere il rilassamento lagrangiano duale, e calcolare di conseguenza il LB al nodo radice, tenendo traccia del tempo di calcolo, del numero di iterazioni e di quello di nodi esplorati (che sarà necessariamente 1). A questo punto passa all'iterazione attraverso le funzioni che si occupano del branch and bound, nel file bb.c, fornendo ad ogni passo dettagliati messaggi di log su tutti i parametri che ci interessano

del problema. Al ritrovamento della soluzione ottima, o allo scattare del time limit, si occupa di liberare tutte le risorse occupate.

III.2 INSTANCE.C

Contiene il metodo gap_instance_load che carica l'istanza a partire dal percorso del file, occupandosi di conseguenza della gestione degli errori, si occupa di aprire il file contenente i moltiplicatori ottenuto con CPLEX.

III.3 LAGRELAX.C

La funzione lagrelax_create produce la chiamata della funzione minknap_prepare del file minknap.c, e la chiamata della creazione di m knapsack del file knapsack.c con la funzione knapsack_create chiamata ricorsivamente. Successivamente viene chiamato il metodo bundle_create del file bundle.c.

La funzione lagrelax_state_create si occupa di allocare tutte le impostazioni iniziali per lo stato del rilassamento lagrangiano (lower bound, moltiplicatori, ecc.)

La funzione lagrangian_dual_solve chiama il metodo bundle_solve del file bundle.c per risolvere il rilassamento lagrangiano duale e ne ritorna il risultato per riferimento.

III.4 MINKNAP.C

Contiene una routine che può essere chiamata per risolvere un problema di Knapsack 0-1, basata sulla programmazione dinamica (infatti bisogna dare un numero massimo di iterazioni).

III.5 KNAPSACK.C

Contiene la funzione knapsack_solve che risolve il problema di Knapsack i-esimo utilizzando la routine del file minknap.c.

III.6 BUNDLE.C

Contiene la funzione getutime utilizzata per tenere traccia del tempo di calcolo.

Richiama e incapsula (con dei wrapper) le funzioni di operazioni tra vettori e matrici, e risoluzione di sistemi lineari, utilizzate nelle librerie BLAS e LAPACK.

Contiene il metodo bundle_qp_solve, che richiama il metodo bundle_qp_solve_mask ordinando i suoi parametri per semplificare la ricerca. Il metodo mask si occupa della ricerca della migliore soluzione per il rilassamento lagrangiano duale.

III.7 BB.C

E’ il file che contiene le funzioni per il branch-and-bound. Le sue funzioni permettono di effettuare la ricerca branch and bound, creare, valutare ed eliminare un nodo e liberare la memoria al termine della ricerca.

IV DISAMINA DEI RISULTATI OTTENUTI SULLE ISTANZE

Per valutare l'efficienza dell'algoritmo di Posta ed altri, si utilizzano le Istanze del Beasley set, che sono normalmente adoperate nella valutazione degli algoritmi per questo tipo di problema. Non vengono considerate le istanze di tipo A e B, perché considerate troppo semplici da risolvere, già per i precedenti algoritmi.
Si considerano e analizzano dettagliatamente, invece, le istanze di tipo C, D ed E. Ogni istanza è definita dal suo tipo, dal numero di macchine e dal numero di lavori da assegnare (es. c05100, istanza di tipo C, 5 lavori e 100 macchine).
Il programma è stato testato su un Intel Core i5 2.6 GHz con 4 GB di RAM DDR3, con un sistema operativo Ubuntu 12, utilizzando un compilatore g++ per il codice di Posta, e uno gfortran per le librerie di base in FORTRAN.

IV.1 ISTANZE DI TIPO C

Sono istanze in cui gli $r_{i,j}$ sono interi casuali compresi tra 5 e 25, i $c_{i,j}$ sono interi casuali compresi tra 10 e 50 e ogni b_i è calcolato come 0.8 $\sum_{j=1}^{n} r_{i,j}$ / m. Di conseguenza i termini c e r sono indipendenti tra loro, mentre i b sono vincolati agli r.

L'istanza c05100 è un'istanza di tipo C, con 5 macchine e 100 lavori da eseguire. Viene individuato il LB al nodo radice di 1929 dopo 0.73 secondi, mediante 8604 iterazioni. La soluzione ottima di 1931 viene calcolata dopo 1.18 secondi, attraverso l'esplorazione di 6 nodi con 13112 iterazioni.

L'istanza c05200 è un'istanza di tipo C, con 200 lavori e 5 macchine per farli. Abbiamo quindi lo stesso numero di macchine della c05100, con il doppio dei lavori da assegnare. Troviamo il LB al nodo radice dopo 1.58 secondi, valore 3454, dopo aver eseguito 9396 iterazioni. Il procedimento iterativo trova l'ottimo dopo 2.11 secondi, nel valore 3456, analizzando 10 nodi con 14079 iterazioni.

L'istanza c10100 è un'istanza di tipo C caratterizzata da 100 lavori da compiere e 10 macchine per compierli. Ha di conseguenza il doppio delle

macchine dell'istanza c05100, per compiere lo stesso numero di lavori. Il nostro algoritmo trova il lower bound al nodo radice in 0.65 secondi; il valore di questo è 1399, per trovarlo compie 7080 iterazioni. A questo punto viene attivato l'algoritmo incrementale, che dopo 1.26 secondi trova l'ottimo in 1402, dopo aver esplorato 19 nodi e compiuto 15545 iterazioni.

L'istanza c10200 è un'istanza di tipo C che ha 10 macchine e 200 lavori da assegnare, ovvero lo stesso numero di macchine della c10100 ma il doppio dei lavori da assegnare. Viene trovato il LB di 2803 al nodo radice. Per trovarlo l'algoritmo impiega 1.89 secondi e 9562 iterazioni. La ricerca della soluzione ottima va a buon fine dopo 8.91 secondi. La soluzione ottima è 2806, trovata dopo 67297 iterazioni, e mediante l'esplorazione di 122 nodi.

L'istanza c10400 è un'istanza di tipo C con 10 macchine e 400 lavori da assegnare, stesso numero di macchine della c10200 ma ancora una volta raddoppiamo il numero dei lavori. Ci aspettiamo pertanto che aumenti ancora la complessità della risoluzione. Il LB al nodo radice viene trovato dopo 3.53 secondi, impiegando 9431 iterazioni; il suo valore è 5595. Dopo 18.37 secondi il programma trova la soluzione ottima di 5597, esplorando 132 nodi e facendo 80433 iterazioni

L'istanza c15900 è un'istanza di tipo C caratterizzata da 15 macchine e 900 lavori da assegnare, il triplo delle macchine della c05100, ma ben nove volte il suo numero di lavori. E' chiaro che cominciamo ad addentrarci in istanze sempre più complesse da risolvere. Infatti il LB al nodo radice viene trovato dopo 9.64 secondi, il suo valore è 11338, e per trovarlo occorrono 9899 iterazioni. Per trovare l'ottimo occorrono 513.75 secondi, ovvero oltre otto minuti e mezzo, viene impiegata l'esplorazione di 1707 nodi, per ben 955669 iterazioni. Il suo valore è 11340.

L'istanza c20100 è un'istanza di tipo C che ha 20 macchine e 100 lavori da assegnare. Aumentiamo quindi il numero di macchine rispetto all'istanza precedente ma caliamo sensibilmente il numero di lavori. Analizziamo come si comporta l'algoritmo in questo caso: il LB al nodo radice viene trovato dopo 1.09 secondi e 8341 iterazioni, valore 1241. Dopo 2.28 secondi viene trovato l'ottimo di 1243; per farlo abbiamo esplorato 24 nodi, facendo 20350 iterazioni.

L'istanza c20200 è un'istanza di tipo C, 20 macchine per 200 lavori da assegnare. Rispetto alla c20100 abbiamo lo stesso numero di macchine ma raddoppiamo il numero di lavori. Troviamo il Lower Bound al nodo radice 2390, in 2.43 secondi, impiegando 9675 iterazioni. Occorrono 20.80 secondi per trovare l'ottimo di 2392, utilizzando 109202 iterazioni per l'esplorazione di 201 nodi.

L'istanza c20400 è un'istanza di tipo C che consta di 20 macchine e 400 lavori. Ancora una volta rispetto alla precedente teniamo fisso il numero di macchine, raddoppiando quello di lavori. Il LB al nodo radice è 4780, questo valore viene trovato dopo 4.86 secondi, mediante 9808 iterazioni. Troviamo la soluzione ottima di 4782 dopo 136.53 secondi. Nella ricerca vengono fatte 369279 iterazioni utilizzando 708 nodi branch and bound.

L'istanza c201600 è un'istanza di tipo C, caratterizzata da 20 macchine e 1600 lavori. In questo caso, rispetto all'istanza c20400, teniamo lo stesso numero delle macchine per addirittura quadruplicare il numero dei lavori. Ci si aspetta che i tempi si dilatino notevolmente, e in effetti è proprio così. Per trovare il LB vengono impiegati 18.44 secondi e 9982 iterazioni; il suo valore è 18801. Per quanto riguarda la ricerca della soluzione ottima, ci si trova di fronte alla prima istanza che va in time limit (3600 secondi) prima di trovarla. Il miglior valore trovato è 18802, con un gap del 0.01%.

L'istanza c30900 è un'istanza di tipo C, caratterizzata da 30 macchine e 900 lavori. Aumentano le macchine, mentre diminuisce il numero di lavori da eseguire. Il Lower Bound al nodo radice di 9981 viene trovato dopo 13.59 secondi, con 9957 iterazioni. Il ritrovamento della soluzione ottima va a buon fine, dopo 1348.54 secondi (oltre 22 minuti). Il valore trovato è 9982, si sono resi necessari 2585 nodi branch and bound per oltre 1,460,000 iterazioni.

L'istanza c40400 è un'istanza di tipo C, con 40 macchine e 40 lavori da eseguire. Ancora una volta si hanno più macchine e meno lavori rispetto all'istanza precedente. Il LB al nodo radice 4243 viene calcolato in 6.15 secondi, utilizzando 9896 iterazioni. Per trovare la soluzione ottima di 4244 occorrono 28.29 secondi, 69 nodi esplorati e 51443 iterazioni. Si può notare una notevole diminuizione di complessità in presenza di questo

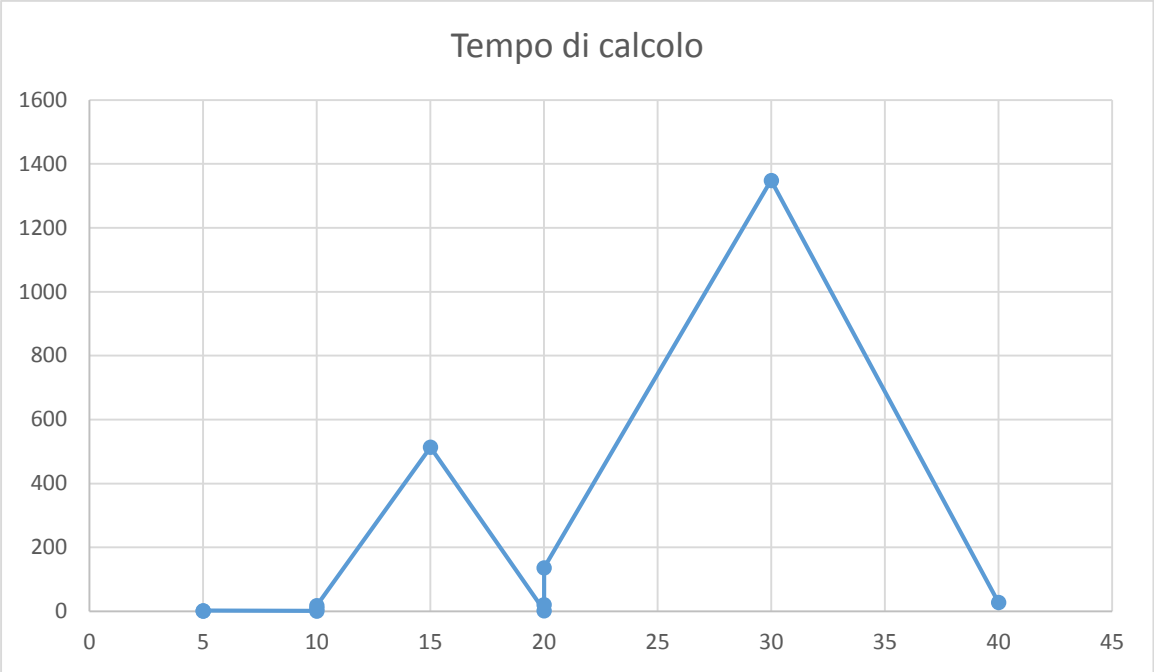
rapporto tra numero di macchine e numero di lavori.

L'istanza c401600 è un'istanza di tipo C, che ha 40 macchine per 1600 lavori da eseguire. Rispetto alla c40400 abbiamo il quadruplo del numero dei lavori da assegnare allo stesso numero di macchine. Essendo andata in time limit la c201600 è facile aspettarsi che lo faccia anche questa. Intanto il LB di 17143 viene ritrovato dopo 26.80 secondi e 9982 iterazioni. Come si era facilmente preventivato, il programma va in time limit con la soluzione di 17144, con un gap dello 0.01%

L'istanza c60900 è un'istanza di tipo C, che consta di 60 macchine per compiere 900 lavori. Questa volta viene aumentato il numero di macchine, per la diminuizione di quello dei lavori. Il LB al nodo radice si trova dopo 19.20 secondi e 9964 iterazioni; tale valore è 9324. Anche questa istanza va in time limit, con il valore di 9326 ed un gap di 0.02%.

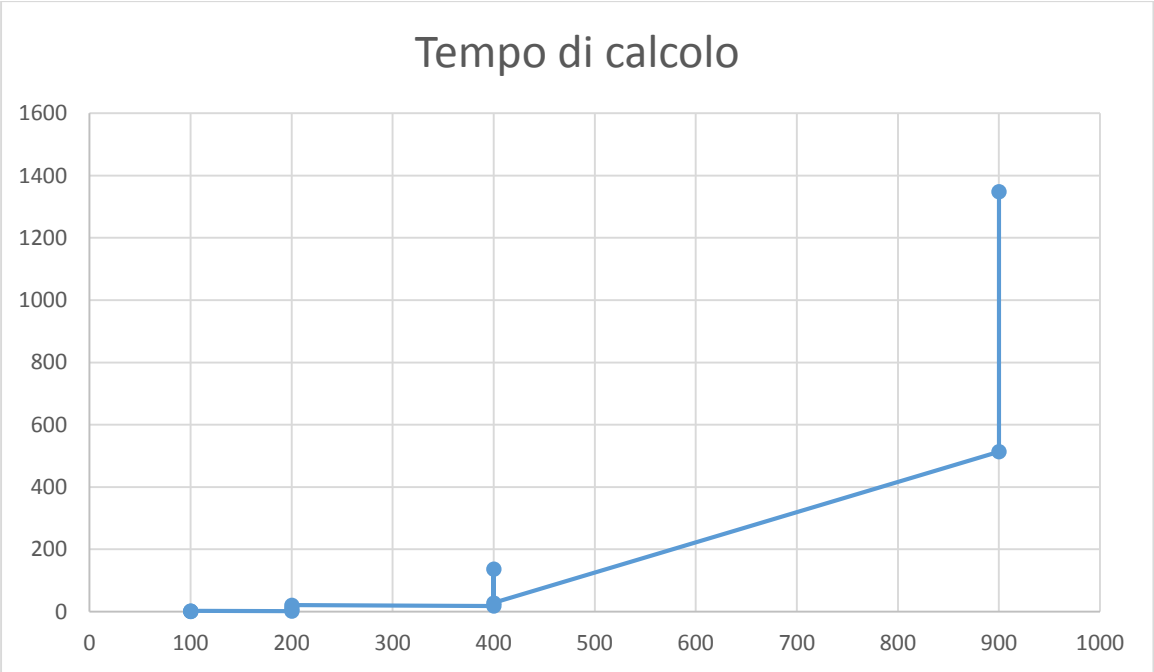
L'istanza c801600 è un'istanza di tipo C, 80 macchine per 1600 lavori, è quella che ha di dimensioni massime. Dopo 38.20 secondi viene individuato il LB di 16282, impiegando 9975 iterazioni. Il programma va in time limit nella ricerca della soluzione ottima, con il miglior valore trovato di 16284, con un gap inferiore a 0.01%.

	LB al nodo radice			Soluzione ottima				Soluzione al time limit	
Nome istanza	LB	t calcolo	iterazioni	soluzione ottima	t calcolo	iterazioni	nodi espl.	Soluzione migliore	gap %
c05100	1929	0,73	8604	1931	1,18	13112	6		
c05200	3454	1,58	9396	3456	2,11	14079	10		
c10100	1399	0,65	7080	1402	1,26	15545	19		
c10200	2803	1,89	9562	2806	8,91	67297	122		
c10400	5595	3,53	9431	5597	18,37	80433	132		
c15900	11338	9,64	9899	11340	513,75	955669	1707		
c20100	1241	1,09	8341	1243	2,28	20350	24		
c20200	2390	2,43	9675	2392	20,8	109202	201		
c20400	4780	4,86	9808	4782	136,53	369279	708		
c201600	18801	18,44	9882					18802	0,01%
c30900	9981	13,59	9957	9982	1348,54	1460000	2585		
c40400	4243	6,15	9896	4244	28,29	51443	69		
c401600	17143	16,8	9982					17144	0,01%
c60900	9324	19,2	9964					9326	0,02%
c801600	16282	38,2	9975					16284	0,01%

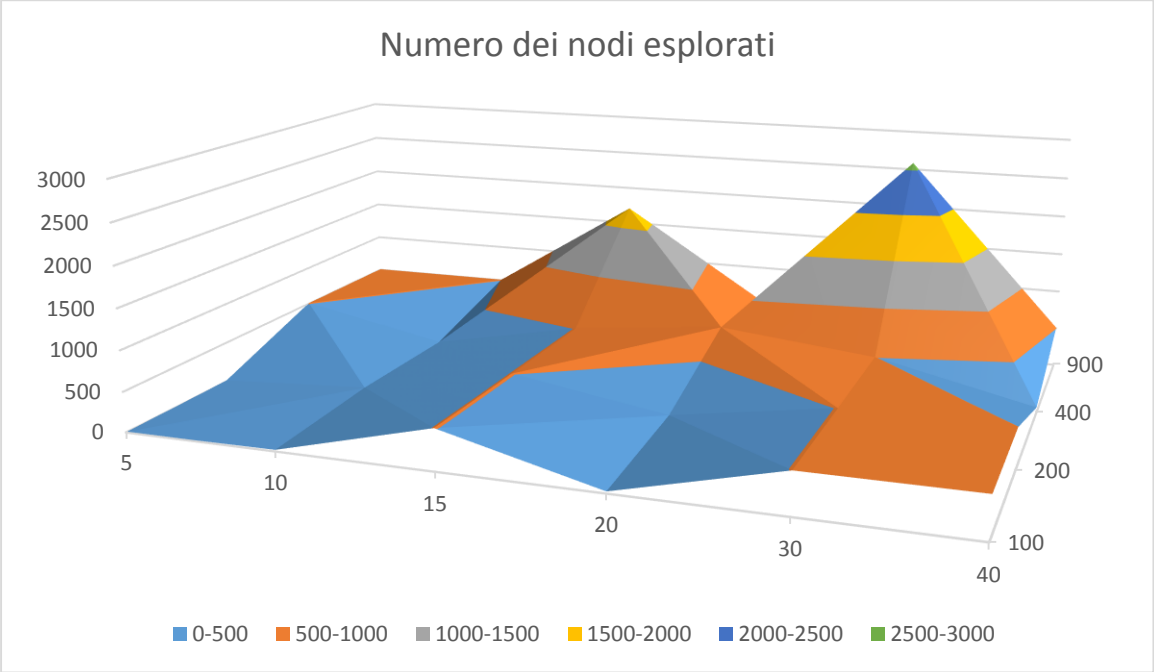


Tempo di calcolo delle istanze di tipo C all'aumentare del numero delle macchine (solo le istanze che stanno dentro il time limit)

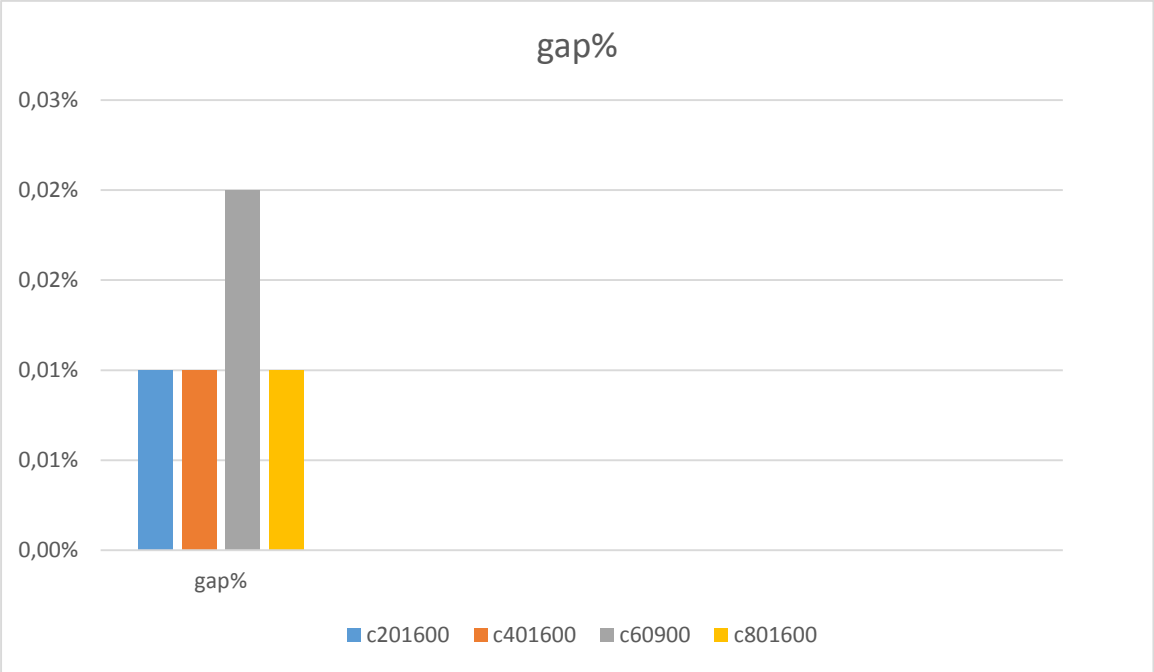
Come si può vedere delle 11 istanze risolte, solo quella con 15 macchine (c15900) e quella con 30 macchine (c30900) superano i 200 secondi di tempo di calcolo.



Tempo di calcolo delle istanze di tipo C all'aumentare del numero dei lavori (solo quelle che non vanno in time limit. L'algoritmo di Posta ed altri lavora molto bene con tutte le istanze che hanno fino a 400 lavori, va in difficoltà con 900 e 1600.



Il numero dei nodi esplorati dall'algoritmo per le istanze di tipo C, in funzione del numero di macchine(ascissa) e del numero di lavori (ordinata), per le istanze risolte nel time limit.



Percentuale di discostamento della miglior soluzione trovata rispetto al LB al nodo radice, nelle istanze di tipo C.

IV.2 ISTANZE DI TIPO D

Questo tipo di istanze viene generato assegnando agli $r_{i,j}$ un intero casuale compreso tra 1 e 100, mentre $c_{i,j} = 111 - r_{i,j} + \varepsilon_i$, ε_i intero casuale compreso tra -10 e 10. I b_i vengono sempre calcolati come $0.8 \sum_{j=1}^{n} r_{i,j} / m$. Abbiamo quindi una proporzionalità tra i $c_{i,j}$ e gli $r_{i,j}$.

L'istanza d05100 è un'istanza di tipo D, caratterizzata da 5 macchine e 100 lavori da eseguire. Rispetto alla sua "sorella" di tipo C ci si aspettano dei tempi più lunghi per il ritrovamento della soluzione. Il Lower Bound di 6349 si trova dopo 1.5 secondi e 8086 iterazioni. Per il ritrovamento della soluzione ottima occorrono 37.87 secondi. La soluzione 6354 è ottenuta dopo l'esplorazione di 926 nodi branch and bound, per 382012 iterazioni.

L'istanza d05200 è un'istanza di tipo D con 5 macchine e 200 lavori da eseguire. Rispetto alla precedente le macchine sono rimaste le stesse come numero, mentre i lavori sono raddoppiati. Il LB al nodo radice viene trovato dopo 3.53 secondi e 9210 iterazioni; il suo valore è 12740. La

soluzione ottima 12742 viene invece trovata dopo 19.21 secondi, attraverso 224 nodi e 112398 iterazioni. Contrariamente a quanto ci si poteva aspettare, il tempo impiegato per trovare la soluzione ottima è molto inferiore a quello dell'istanza precedente. Ciò è sicuramente dovuto alla qualità del LB trovato inizialmente.

L'istanza d10100 è un'istanza di tipo D con 10 macchine e 100 lavori. Rispetto alla d05200 raddoppiano le macchine mentre si dimezzano il numero di lavori. Si trova il LB di 6341 dopo 2.38 secondi con l'impiego di 9139 iterazioni. Per il calcolo della soluzione ottima di 6347 sono necessari 159.30 secondi (oltre due minuti e mezzo), utilizzando 2435 nodi e poco più di un milione di iterazioni.

L'istanza d10200 è un'istanza di tipo D, che ha 10 macchine e 200 lavori da eseguire. Raddoppia quindi il numero dei lavori tenendo fisso il numero delle macchine. Il LB viene trovato in 4.73 secondi, con 9397 iterazioni, valore 12425. Il ritrovamento della soluzione ottima non viene portato a buon fine prima del time limit di 3600 secondi. La migliore soluzione trovata è 12430, con un gap dello 0.04%.

L'istanza d10400 è un'istanza di tipo D, 10 macchine per 400 lavori. Viene ulteriormente raddoppiato il numero dei lavori, a fronte dello stesso numero di macchine, di conseguenza è facile aspettarsi che anche quest'istanza vada in time limit. Il LB al nodo radice di 24958 viene ritrovato dopo 7.99 secondi e 9425 iterazioni. Il sistema va in time limit come pronosticato, con il miglior valore trovato di 24961 e un gap dello 0.01%.

L'istanza d15900 è un'istanza di tipo D, dotata di 15 macchine e 900 lavori da eseguire. Rispetto alla precedente aumentano del 50% le macchine, - mentre il numero di lavori aumenta del 125%. Il Lower bound necessita di 20.81 secondi e 9548 iterazioni per essere trovato; il suo valore è 55402. Anche con questa istanza l'algoritmo va in time limit, con la miglior soluzione trovata 55403 per un gap del 0.002%.

L'istanza d20100 è un'istanza di tipo D, caratterizzata da 20 macchine e 100 lavori. Sono aumentate le macchine rispetto alla d15900, mentre si è ridotto notevolmente il numero di lavori da eseguire. Viene trovato il LB di

6176, impiegando 3.33 secondi per 9318 iterazioni. Anche in questo caso l'algoritmo di Posta ed altri va in time limit, con la migliore soluzione trovata di 6183, e un gap di 0.11%.

L'istanza d20200 è un'istanza di tipo D, con 20 macchine e 200 lavori. Quindi ci sono lo stesso numero di macchine della precedente per il doppio dei lavori da svolgere. Il LB al nodo radice di 12229 viene trovato dopo 6.84 secondi e 9452 iterazioni. Superiamo il time limit anche con questa istanza, senza trovare una soluzione ottima. La migliore soluzione è 12233, per un gap del 0.03%.

L'istanza d20400 è un'istanza di tipo D che consta di 20 macchine e 400 lavori. Rispetto alla d20200 quindi si hanno lo stesso numero di macchine per il doppio dei lavori. Il Lower Bound di 24560 necessita di 12.27 per essere ritrovato, con 9540 iterazioni compiute. Viene superato il time limit con la migliore soluzione trovata di 24562, per un 0.01% di gap.

L'istanza d201600 è un'istanza di tipo D, 20 macchine per 1600 lavori. Quindi vengono ancora aumentati i lavori, in particolare quadruplicati rispetto alla d20400. Dopo 40.12 secondi viene trovato il LB di 97822, impiegando 9648 iterazioni. Al raggiungimento del time limit la migliore soluzione è 97825, con un gap pari a 0.003%

L'istanza d30900 è un'istanza di tipo D, dotata di 30 macchine e con 900 lavori da svolgere. Rispetto alla d201600 abbiamo il 50% in più di macchine per poco più della metà dei lavori. Il calcolo del LB al nodo radice richiede 31.21 secondi e 9683 iterazioni; il valore di questo è 54832. Allo scattare del time limit il miglior valore trovato è 54833 per un gap dello 0.002%.

L'istanza d40400 è un'istanza di tipo D, caratterizzata da 40 macchine e 400 lavori da eseguire. Rispetto alla precedente abbiamo quindi incrementato il numero delle macchine e decrementato quello dei lavori. Il Lower Bound di 24349 viene ritrovato dopo 14.19 secondi, e 9396 iterazioni. Anche in questo caso scatta il time limit, con la miglior soluzione trovata di 24350 e un gap di 0.004%

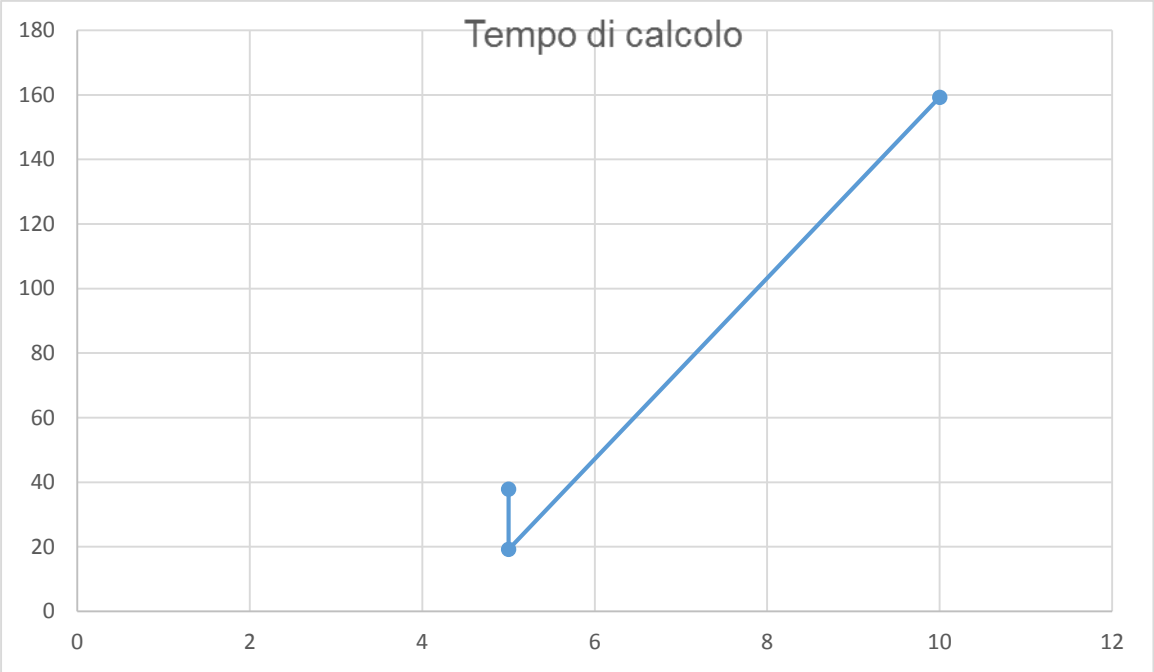
L'istanza d401600 è un'istanza di tipo D, con 40 macchine e 1600 lavori.

Tenendo fisso il numero delle macchine, si è quindi quadruplicato il numero di lavori da compiere. Occorrono 43.92 secondi e 9246 iterazioni per calcolare il LB di 97105. Superato il time limit, la migliore soluzione è 97106, con un gap di 0.001%.

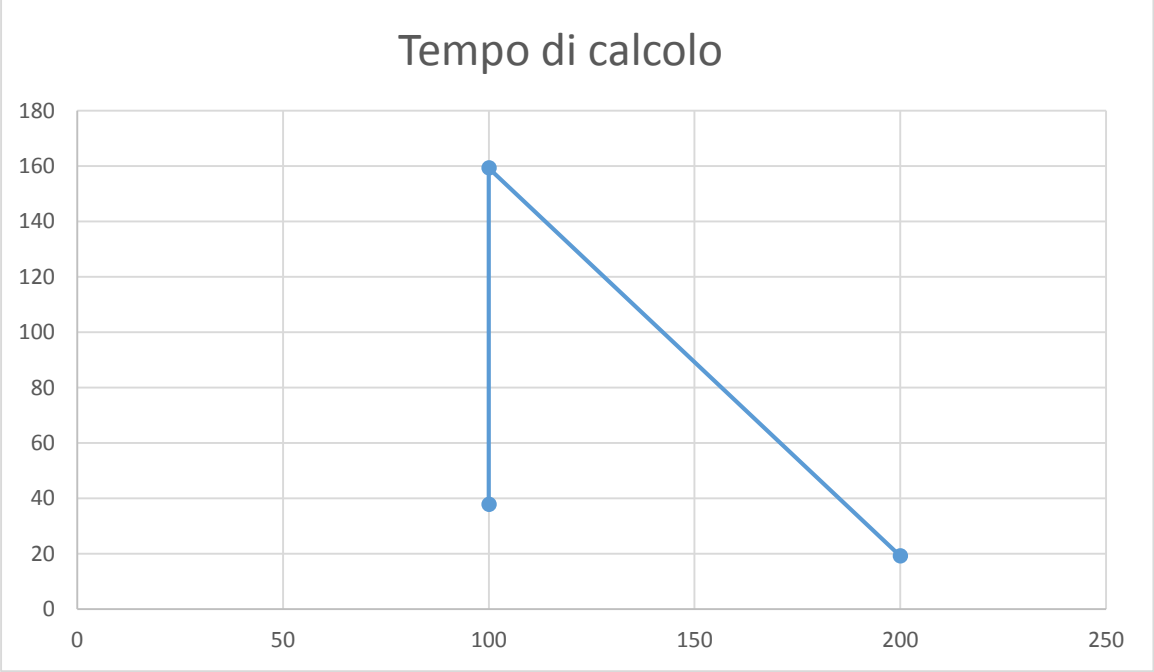
L'istanza d60900 è un'istanza di tipo D, 60 macchine per 900 lavori, quindi un aumento di ½ delle macchine e di 5/4 del numero di lavori. Il LB al nodo radice è 54551, calcolato dopo 36.15 secondi e 9265 iterazioni. Naturalmente viene superato il time limit anche in questo caso, con la soluzione migliore di 54552 ed un gap del 0.002%.

L'istanza d801600 è un'istanza di tipo D, caratterizzata da 80 macchine e 1600 lavori; rispetto alla d60900 le macchine sono aumentate del 33% mentre i lavori sono quasi raddoppiati. Viene ritrovato il LB di 97034 dopo 85.95 secondi (oltre un minuto), facendo 9194 iterazioni. Allo scattare del time limit la miglior soluzione trovata è 97035, per un gap del 0.001%

	LB al nodo radice			Soluzione ottima				Soluzione al time limit	
Nome istanza	LB	t calcolo	iterazioni	soluzione ottima	t calcolo	iterazioni	nodi espl.	Soluzione migliore	gap %
d05100	6349	1,5	8086	6354	37,87	382012	926		
d05200	12740	3,53	9210	12742	19,21	112398	224		
d10100	6341	2,38	9139	6347	159,3	>1000000	2435		
d10200	12425	4,73	9397					12430	0,04%
d10400	24958	7,99	9425					24961	0,01%
d15900	55402	20,81	9548					55403	0,002%
d20100	6176	3,33	9318					6183	0,11%
d20200	12229	6,84	9452					12233	0,03%
d20400	24560	12,27	9540					24562	0,01%
d201600	97822	40,12	9648					97825	0,003%
d30900	54832	31,21	9683					54833	0,002%
d40400	24349	14,19	9396					24350	0,004%
d401600	97105	43,92	9246					97106	0,001%
d60900	54551	36,15	9265					54552	0,002%
d801600	97034	85,95	9194					97035	0,001%

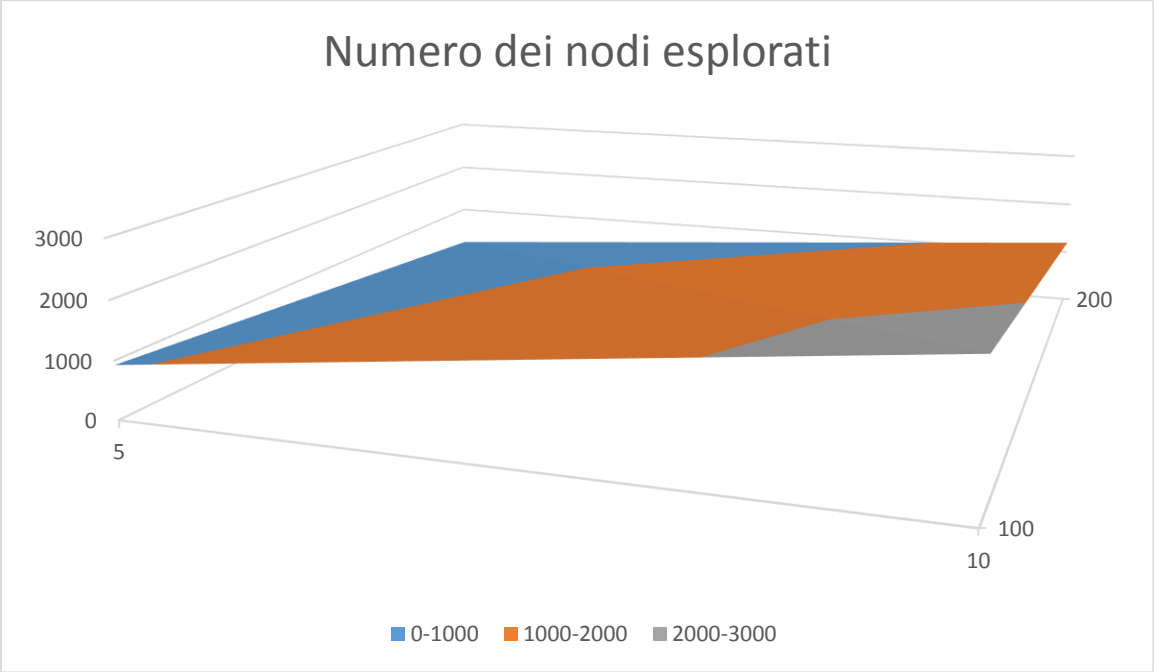


Tempo di calcolo delle istanze di tipo D all'aumentare del numero delle macchine (solo quelle che non vanno in time limit. Solo tre istanze di tipo D vengono risolte all'interno del time limit. Queste istanze vengono comunque risolte con un tempo di calcolo ragionevole, che aumenta all'aumentare del numero delle macchine.

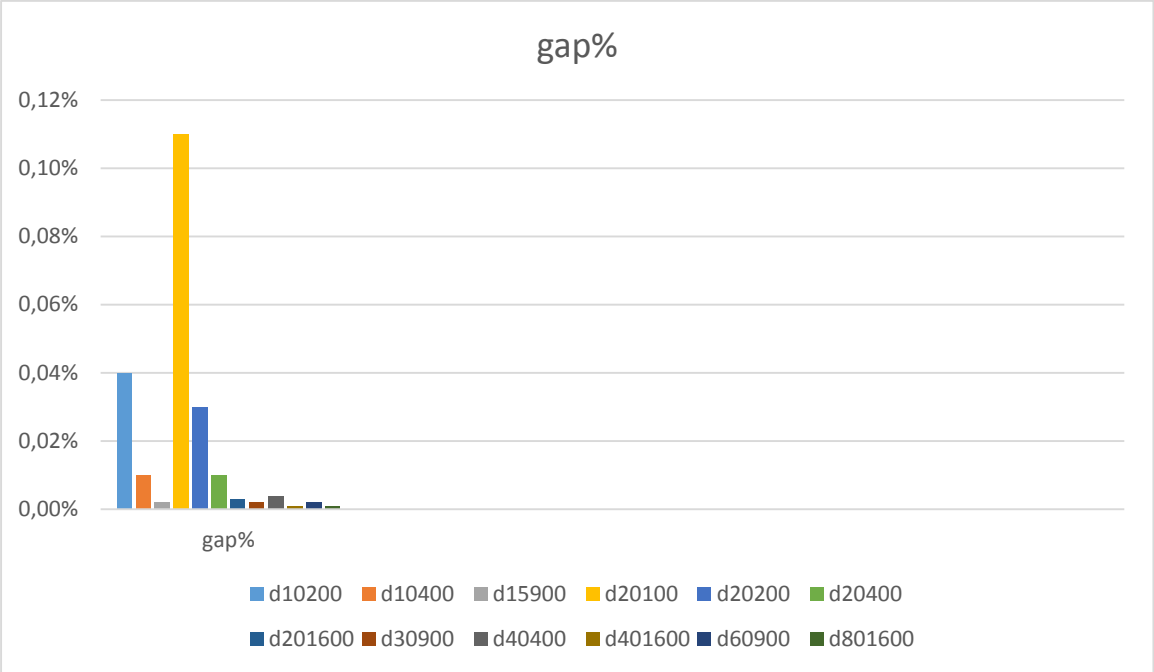


Tempo di calcolo delle istanze di tipo D all'aumentare del numero dei lavori (solo quelle che non vanno in time limit. In questo caso il tempo di calcolo diminuisce all'aumentare del numero dei lavori, prendendo in

esame le sole tre istanze che terminano all'interno del time limit.



Il numero dei nodi esplorati dall'algoritmo per le istanze di tipo D, in funzione del numero di macchine (ascissa) e del numero di lavori (ordinata), per le istanze risolte nel time limit.



Percentuale di discostamento della miglior soluzione trovata rispetto al LB al nodo radice, nelle istanze di tipo D.

IV.3 ISTANZE DI TIPO E

Le istanze di tipo E hanno $r_{i,j} = 1 - 10\ ln\ \varepsilon$, ε reale casuale maggiore di 0 e non superiore a 1, $c_{i,j} = \frac{1000}{r_{i,j}} - 10\ n$, n reale casuale compreso tra 0 e 1. I b_i vengono calcolati invece nel solito modo. In questo caso i termini c ed r sono inversamente proporzionali tra loro.

L'istanza e05100 è la più semplice istanza di tipo E, dotata di 5 macchine e 100 lavori da eseguire. La ricerca del LB al nodo radice richiede 0.74 secondi per 8873 iterazioni; il suo valore è 12673. Una soluzione ottima viene ritrovata dopo 4.46 secondi. Per trovarla sono necessarie 87594 iterazioni, per 174 nodi esplorati. Il suo valore è 12681.

L'istanza e05200 è un'istanza di tipo E, con 5 macchine e 200 lavori; rispetto alla e05100 sono raddoppiati i lavori, tenendo costante il numero delle macchine. Il LB al nodo radice di 24926 viene ritrovato dopo 1.31 secondi, con 8805 iterazioni. La soluzione ottima di 24930 è individuata dopo l'esplorazione di 40 nodi e 25473 iterazioni, per un tempo di calcolo

di 2.62 secondi.

L'istanza e10100 è un'istanza di tipo E, 10 macchine per 100 lavori. Si raddoppiano quindi le macchine mentre viene dimezzato il numero di lavori. Dopo 1.21 secondi, mediante 8869 iterazioni, si arriva ad un LB di 11568. La soluzione ottima di 11579 viene individuata dopo 31.63 secondi. Per la sua ricerca vengono esplorati 821 nodi, per un totale di 359662 iterazioni.

L'istanza e10200 è un'istanza di tipo E, 10 macchine per 200 lavori. Si ha quindi lo stesso numero di macchine della precedente, per il doppio dei lavori da compiere. Il ritrovamento del LB al nodo radice va a buon fine dopo 1.95 secondi e 8734 iterazioni, valore 23301. Dopo 45.27 secondi viene invece individuata la soluzione ottima di 23308; sono stati esplorati 695 nodi per un totale di 350009 iterazioni.

L'istanza e10400 è un'istanza di tipo E, caratterizzata da 10 macchine e 400 lavori. Ancora una volta si è in presenza di un raddoppio del numero dei lavori a fronte dello stesso numero di macchine. Occorrono 3.98 secondi e 9065 iterazioni per ritrovare il LB di 45744. L'ottimo di 45746 viene raggiunto invece dopo 17.35 secondi, con 120 nodi esplorati e 69778 iterazioni.

L'istanza e15900 è un'istanza di tipo E, dotata di 15 macchine e 900 lavori. Rispetto alla e10400 aumentano il numero di macchine e quello dei lavori. Il LB di 102419 viene individuato dopo 11.05 secondi e 9059 iterazioni. Per ritrovare la soluzione ottima di 102421 sono necessari 42.86, per l'esplorazione di 100 nodi e 63431 iterazioni.

L'istanza e20100 è un'istanza di tipo E, che consta di 20 macchine e 100 lavori. Rispetto alla precedente, a fronte di un aumento del numero delle macchine, abbiamo una notevole diminuizione del numero dei lavori. Il ritrovamento del LB al nodo radice richiede 2.25 secondi e 9237 iterazioni; il suo valore è 8431. La soluzione ottima di 8437 viene individuata dopo 14.25 secondi, grazie all'esplorazione di 174 nodi e 80446 iterazioni.

L'istanza e20200 è un'istanza di tipo E, con 20 macchine e 200 lavori. Si hanno quindi lo stesso numero di macchine della e20100 per il doppio dei

lavori. Dopo 3.51 secondi e 9042 iterazioni si ritrova il LB di 22376. Dopo 7.68 secondi si arriva alla soluzione ottima di 22379, esplorando 37 nodi per 25770 iterazioni.

L'istanza e20400 è un'istanza di tipo E, 20 macchine per 400 lavori. Ancora una volta viene raddoppiato il numero di lavori, per lo stesso numero di macchine. Il LB di 44875 è calcolato dopo 6.78 secondi, mediante 9116 iterazioni. Occorrono 25.62 secondi per trovare invece la soluzione ottima di 44877. Per calcolarla sono stati esplorati 96 nodi, per un totale di 59252 iterazioni.

L'istanza e201600 è un'istanza di tipo E, caratterizzata da 20 macchine e 1600 lavori. Rispetto alla precedente si ha lo stesso numero di macchine, per il quadruplo dei lavori. 180643 è il LB al nodo radice; viene individuato dopo 24.52 secondi, con 9355 iterazioni. La soluzione ottima viene ritrovata dopo 1072.16 secondi (quasi 18 minuti), mediante l'esplorazione di 1507 nodi, per 854087 iterazioni. Il suo valore è 180645.

L'istanza e30900 è un'istanza di tipo E, che consta di 30 macchine e 900 lavori. Al confronto della e201600 abbiamo un aumento del numero di macchine, per un numero di lavori quasi dimezzato. Dopo 19.36 secondi, per mezzo di 9152 iterazioni, si ritrova il LB di 100426. La soluzione ottima viene ritrovata dopo 290.07 secondi (quasi 5 minuti), esplorando 460 nodi e compiendo 275611 iterazioni; il valore dell'ottimo è 100428.

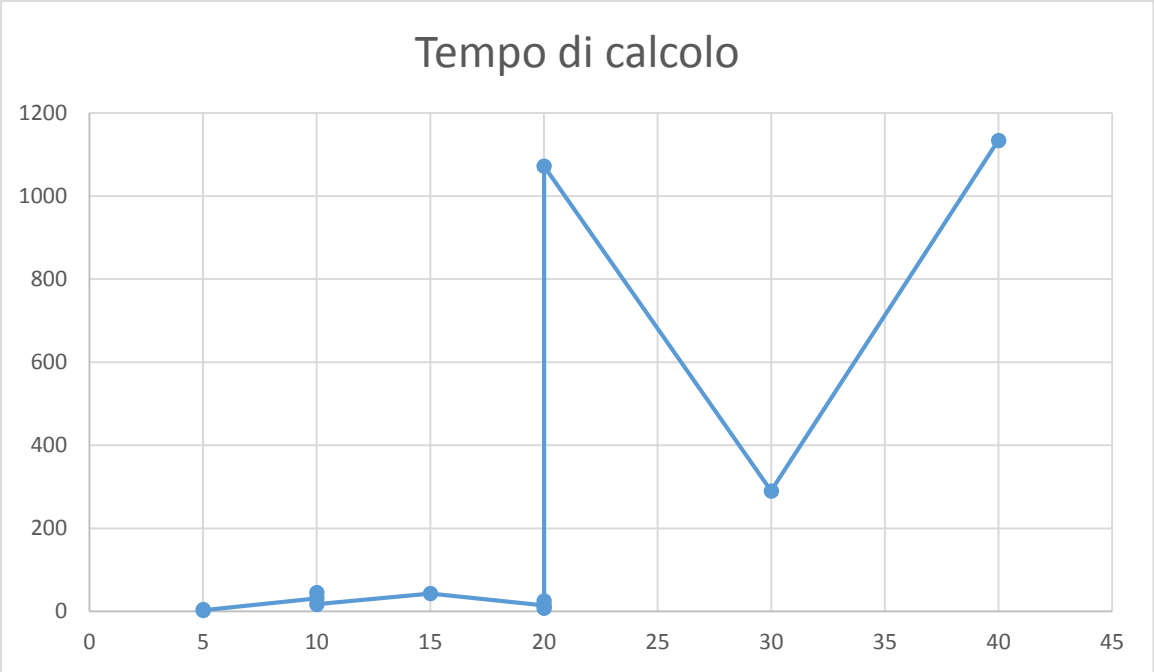
L'istanza e40400 è un'istanza di tipo E, dotata di 40 macchine e 400 lavori da eseguire. Ancora una volta si vede un aumento del numero delle macchine a fronte di una diminuizione del numero di lavori. Il LB di 44556 viene individuato dopo 12.21 secondi e 9190 iterazioni. Occorrono 1134.23 secondi (più di 18 minuti) per ritrovare la soluzione ottima di 44561, mediante l'esplorazione di 3270 nodi, per 1559802 iterazioni.

L'istanza e401600 è un'istanza di tipo E, con 40 macchine e 1600 lavori. Rispetto alla precedente ci sono quindi le stesse macchine per il quadruplo dei lavori da eseguire. Il LB al nodo radice è 178292, calcolato dopo 45.19 secondi con 9422 iterazioni. Per il calcolo della soluzione ottima il programma va in time limit, arrivando alla miglior soluzione di 178293, per un gap del 0.0006%.

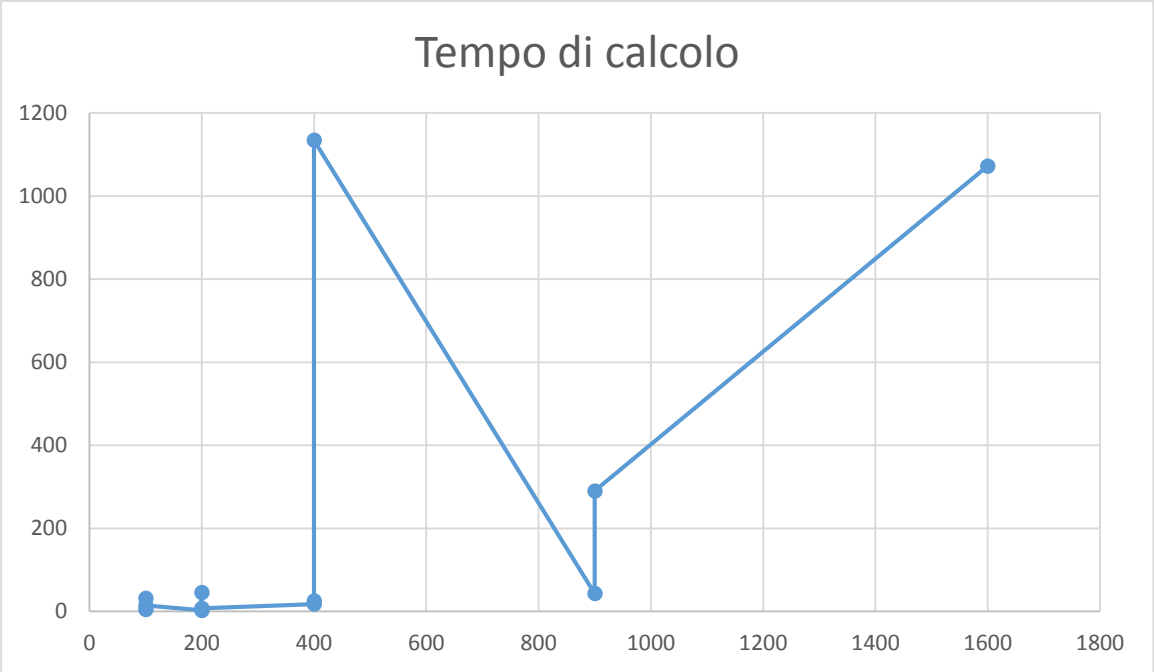
L'istanza e60900 è un'istanza di tipo E, 60 macchine per 900 lavori. Rispetto alla e401600 c'è una riduzione del numero dei lavori per un aumento del numero delle macchine. Il LB di 100146 viene individuato dopo 37.28 secondi, con 9420 iterazioni. Nel ritrovamento della soluzione l'algoritmo fa scattare il time limit, con il valore di 100149, 0.003% di gap.

L'istanza e801600 è un'istanza di tipo e, con 80 macchine che devono eseguire 1600 lavori. Aumentano sia il numero delle macchine che quello dei lavori. Il Lower Bound è 176818; per calcolarlo occorrono 81.81 secondi e 9318 iterazioni. Allo scattare del time limit la miglior soluzione trovata è 176820, con un gap del 0.001 %.

	LB al nodo radice			Soluzione ottima				Soluzione al time limit	
Nome istanza	LB	t calcolo	iterazioni	soluzione ottima	t calcolo	iterazioni	nodi espl.	Soluzione migliore	gap %
e05100	12673	0,74	8873	12681	4,46	87594	174		
e05200	24926	1,31	8805	24930	2,62	25473	40		
e10100	11568	1,21	8869	11579	31,63	359662	821		
e10200	23301	1,95	8734	23308	45,27	350009	695		
e10400	45744	3,98	9065	45746	17,35	69778	120		
e15900	102419	11,05	9059	102421	42,86	63431	100		
e20100	8431	2,25	9237	8437	14,25	80446	174		
e20200	22376	3,51	9042	22379	7,68	25770	37		
e20400	44875	6,78	9116	44877	25,62	59252	96		
e201600	180643	24,52	9355	180645	1072,16	854087	1507		
e30900	100426	19,36	9152	100428	290,07	275611	460		
e40400	44556	12,21	9190	44561	1134,23	1559802	3270		
e401600	178292	45,19	9422					178293	0,0006
e60900	100146	37,28	9420					100149	0,003
e801600	176818	81,81	9318					176820	0,001

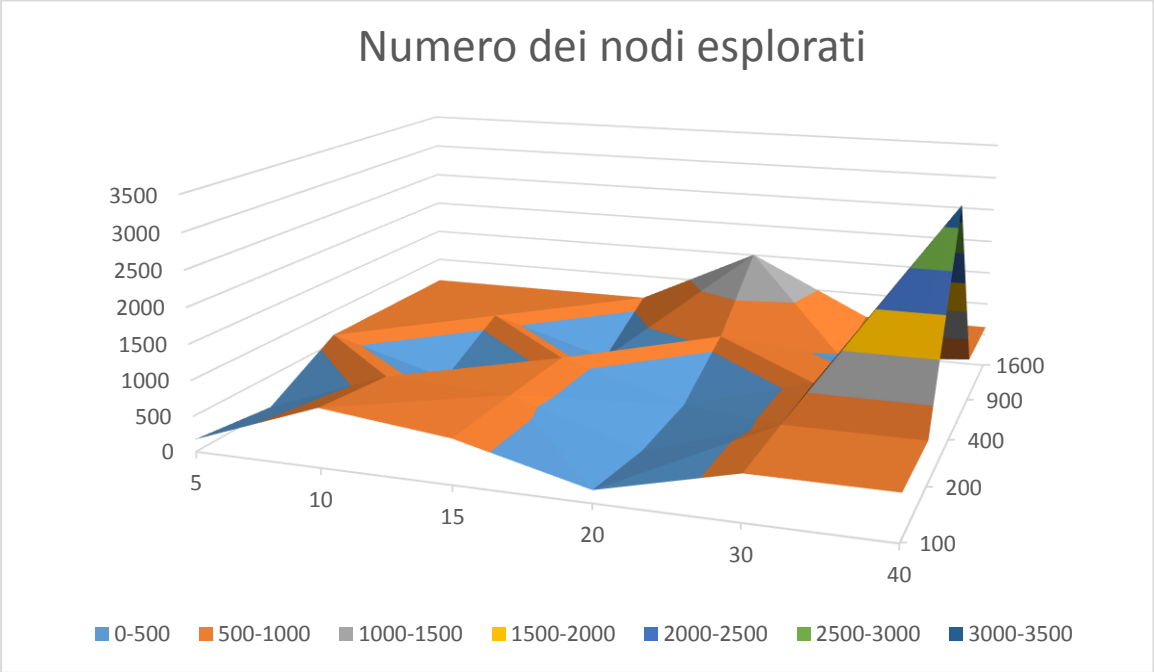


Tempo di calcolo delle istanze di tipo E all'aumentare del numero delle macchine (solo quelle che non vanno in time limit. E' interessante notare come il metodo di Posta ed altri si comporti molto bene con le istanze di tipo E, stando all'interno del minuto di tempo di calcolo con il 60% delle istanze.

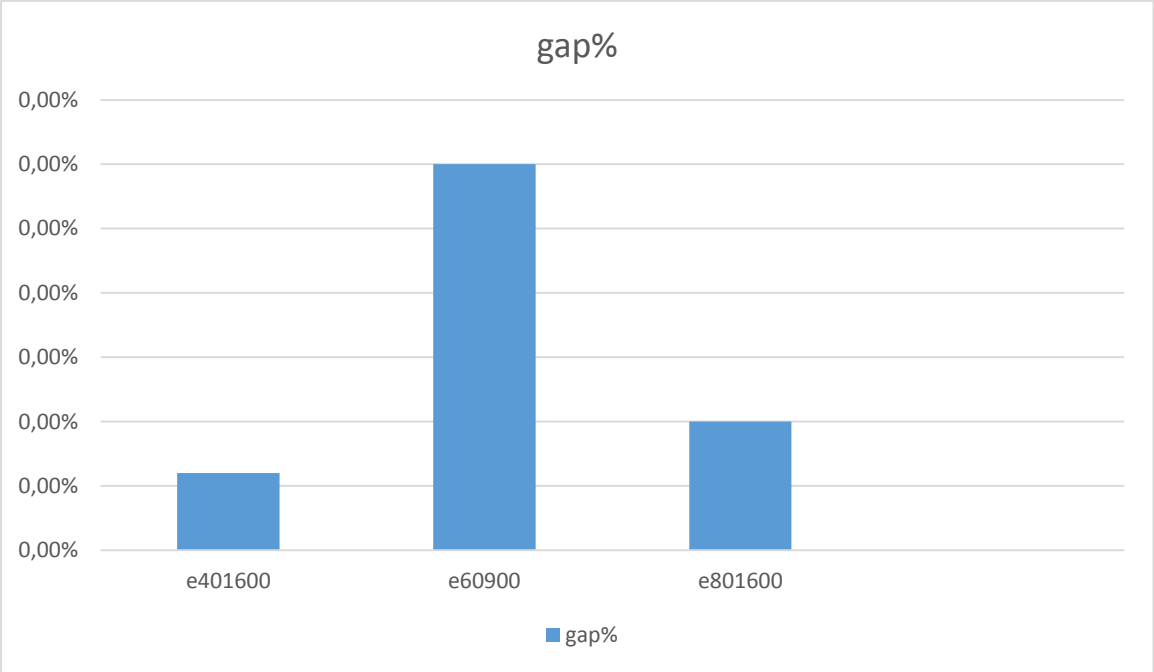


Tempo di calcolo delle istanze di tipo E all'aumentare del numero dei lavori (solo quelle che non vanno in time limit. Tutte le istanze con n fino

a 200 vengono risolte con un brevissimo tempo di calcolo. Solo due istanze superano i 400 secondi.



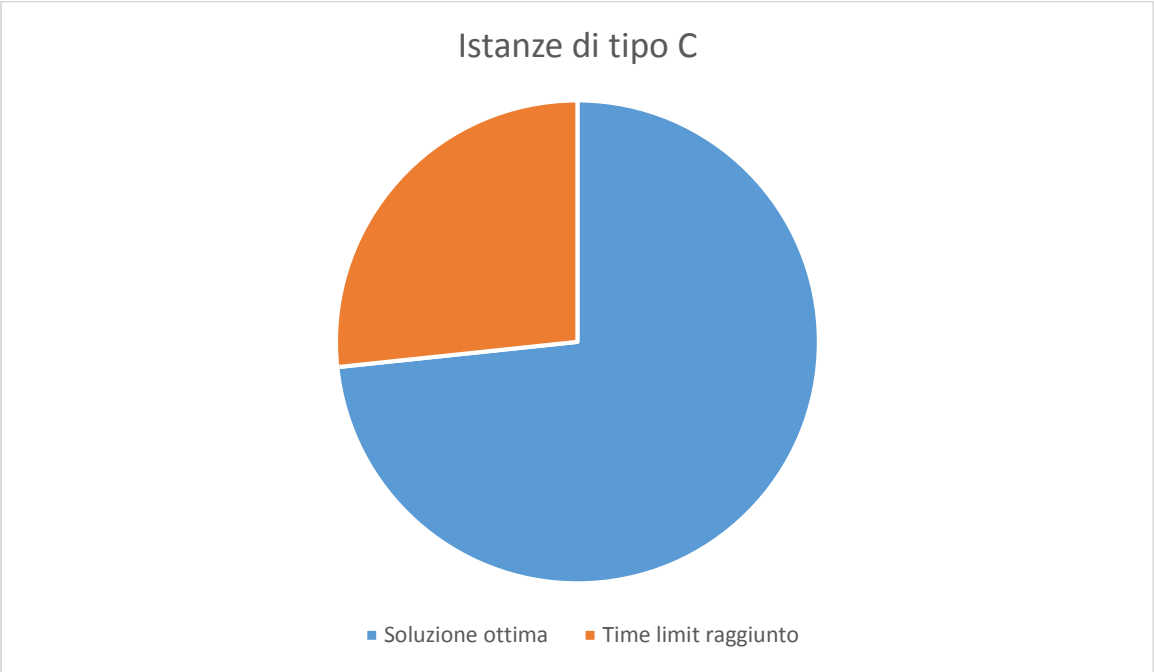
Il numero dei nodi esplorati dall'algoritmo per le istanze di tipo D, in funzione del numero di macchine (ascissa) e del numero di lavori (ordinata), per le istanze risolte nel time limit. Dai tre grafici relativi ai nodi esplorati si può notare come il loro numero aumenti maggiormente all'aumentare del numero delle macchine, rispetto a quello dei lavori.

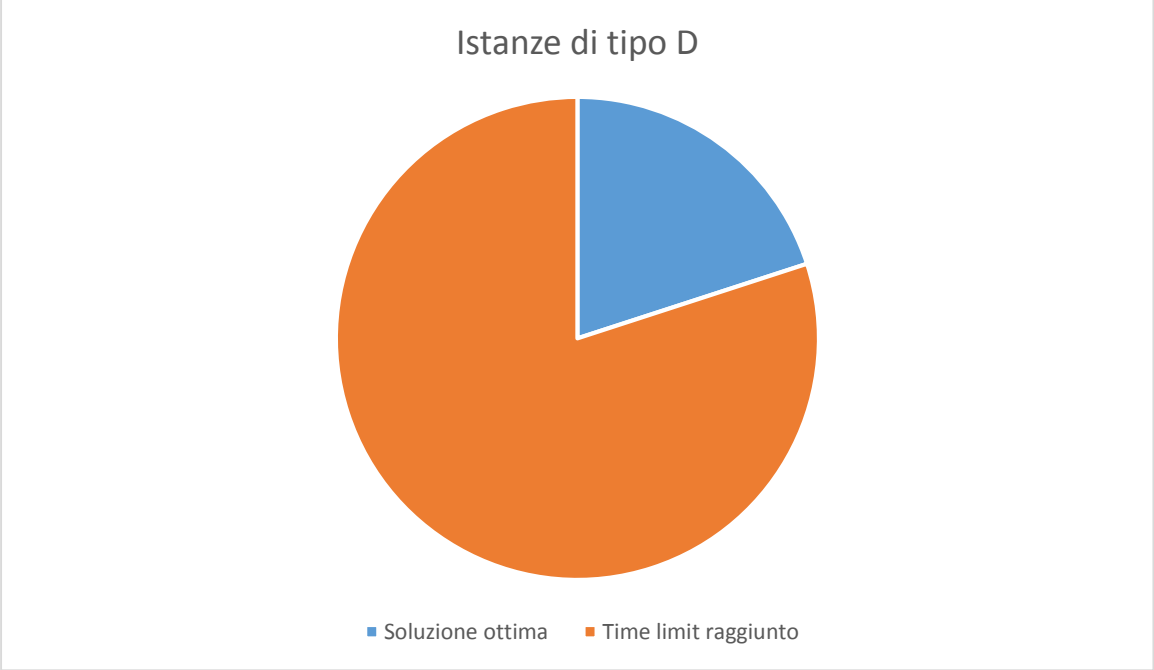


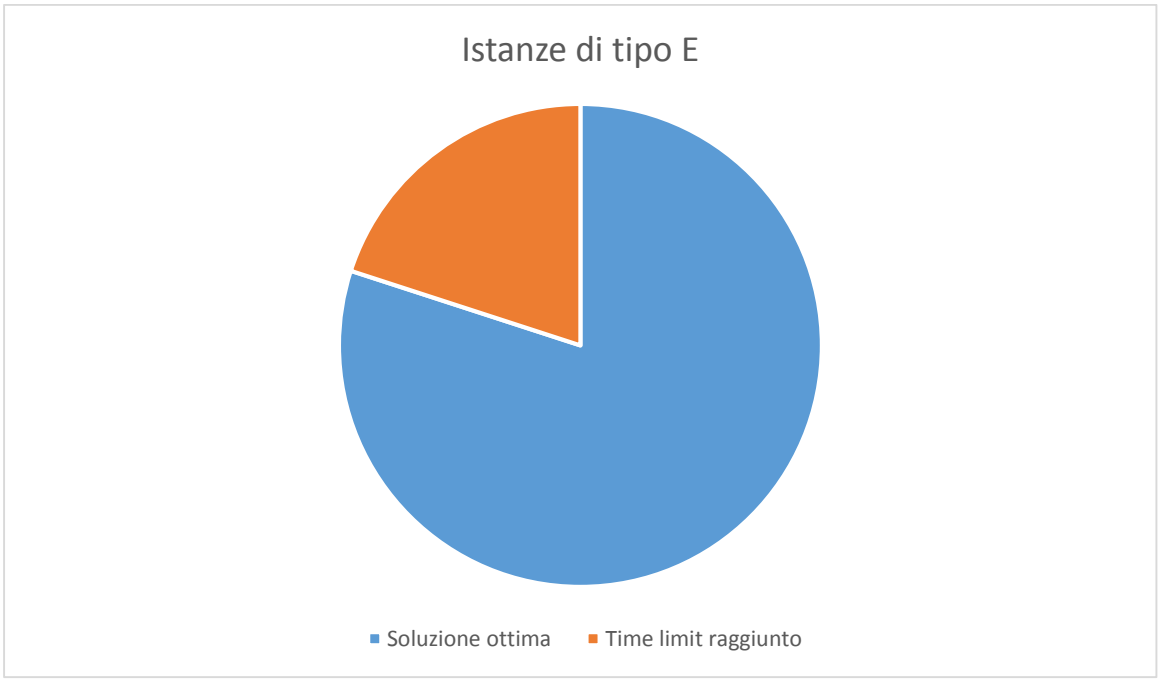
Percentuale di discostamento della miglior soluzione trovata rispetto al LB al nodo radice, nelle istanze di tipo E.

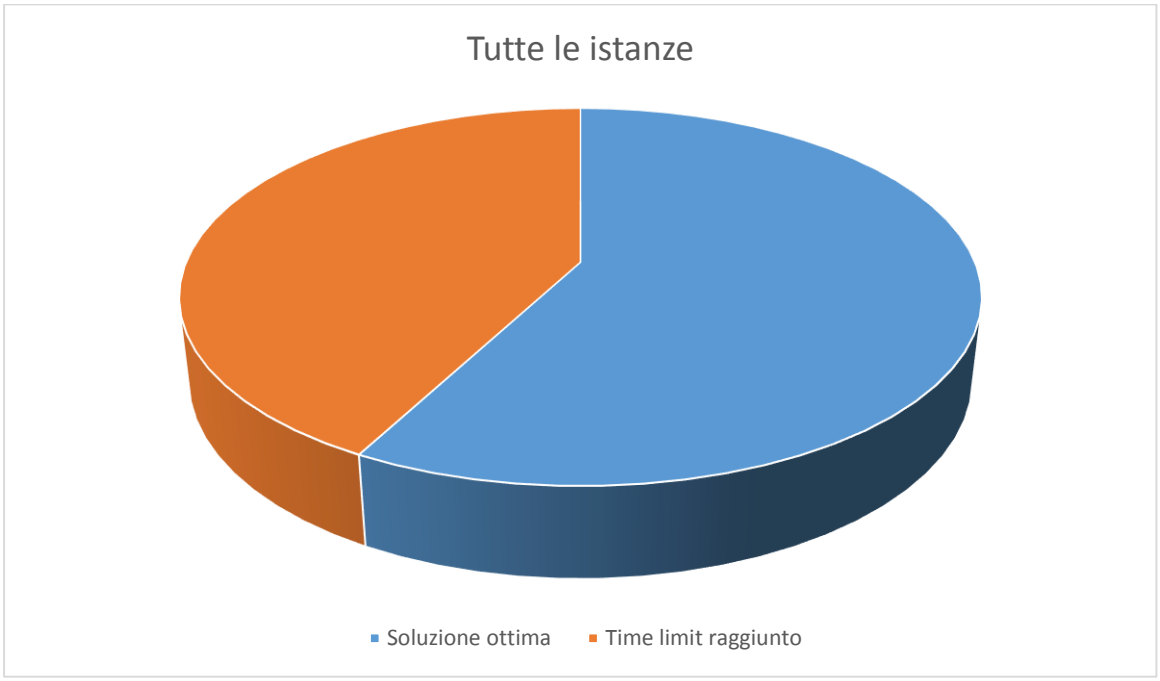
V. CONCLUSIONI

L’algoritmo di Posta ed altri arriva alla soluzione ottima prima del time-limit per il 73.3% delle istanze di tipo C, il 20% delle istanze di tipo D e l’80% delle istanze di tipo E, per un totale del 57.8%. Considerando la frequenza del processore non altissima (2.6 GHz) e il time limit ridotto di 3600 secondi, è una prestazione più che accettabile.






Istanze di tipo E
Soluzione ottima
Time limit raggiunto


Tutte le istanze
Soluzione ottima
Time limit raggiunto

VI. RIFERIMENTI

1. Atamtürk, A., Savelsbergh, M.W.P.: Integer-programming software systems. Ann. Oper. Res. 140(1), 67–124 (2005)

2. Avella, P., Boccia, M., Vasilyev, I.: A computational study of exact knapsack separation for the generalized assignment problem. Comput. Optim. Appl. 45(3), 543–555 (2010)

3. Beasley, J.E.: Generalised assignment problem test data sets. http://people.brunel.ac.uk/~mastjjb/jeb/orlib/gapinfo.html

4. Diaz, J.A., Fernandez, E.: A tabu search heuristic for the generalized assignment problem. Eur. J. Oper. Res. 132(1), 22–38 (2001)

5. Frangioni, A.: Solving semidefinite quadratic problems within nonsmooth optimization algorithms. Comput. Oper. Res. 23(11), 1099–1118 (1996)

6. Haddadi, S., Ouzia, H.: Effective algorithm and heuristic for the generalized assignment problem. Eur. J. Oper. Res. 153(1), 184–190 (2004)

7. Karabakal, N., Bean, J.C., Lohmann, J.R.: A steepest descent multiplier adjustment method for the generalized assignment problem. Technical report, University of Michigan (1992)

8. Nauss, R.M.: Solving the generalized assignment problem: an optimizing and heuristic approach. INFORMS J. Comput. 15(3), 249–266 (2003)

9. Pigatti, A., de Aragao, M.P., Uchoa, E.: Stabilized branch-and-cut-and-price for the generalized assignment problem. Electron. Notes Discrete Math. 5, 389–395 (2005)

10. Pisinger, D.: A minimal algorithm for the 0-1 knapsack problem. Oper. Res. 45(5), 758–767 (1997)

11. Posta M., Ferland J. A., Michelon P.: An exact method with variable fixing for solvingthe generalized assignment problem. Comput. Optim. Appl. 45(3), 543–558 (2011)

12. Ross, G.T., Soland, R.M.: A branch and bound algorithm for the generalized assignment problem. Math. Program. 8(1), 91–103 (1975)

13. Savelsbergh, M.: A branch-and-price algorithm for the generalized assignment problem. Oper. Res. 45(6), 831–841 (1997)

14. Wolsey, L.A.: Integer Programming. Wiley, New York (1998)

15. Yagiura, M., Ibaraki, T., Glover, F.: An ejection chain approach for the generalized assignment problem. INFORMS J. Comput. 16(2), 133–151 (2004)

16. Yagiura, M., Ibaraki, T., Glover, F.: A path relinking approach with ejection chains for the generalized assignment problem. Eur. J. Oper. Res. 169(2), 548–569 (2006)

Sommario