

UNIVERSITÀ DEGLI STUDI DI MILANO

Facoltà di Scienze Matematiche, Fisiche e Naturali

Dipartimento di Tecnologie dell'Informazione

Corso di Laurea in Informatica



ALGORITMI DI PROGRAMMAZIONE  
MATEMATICA PER UN PROBLEMA DI  
TURNAZIONE DI SQUADRE DI  
PRONTO INTERVENTO

Relatore: Prof. Giovanni Righini

Tesi di Laurea di:

Daniele Gilberti

Matricole 641387

Anno accademico 2004/2005

# Ringraziamenti

Il primo ringraziamento è rivolto ai miei genitori, che mi sono stati sempre accanto, sostenendomi nei momenti difficili.

Ringrazio il Prof. Giovanni Righini per il suo aiuto avuto durante lo svolgimento di questa tesi, per la pazienza e la disponibilità avuta nei miei confronti senza le quali non sarei riuscito a concludere questa tesi.

Un altro ringraziamento è rivolto a tutti i gli amici che mi hanno sostenuto ed aiutato nel mio cammino universitario.

Ringrazio la mia fidanzata Deborah per il suo costante sostegno alle mie scelte e l'appoggio nei momenti difficili.

Grazie a tutte le persone che mi sono state accanto durante questi anni perché mi hanno permesso di raggiungere questo importante obiettivo.

# Capitolo 1

## Introduzione

## 1.1 Motivazione

Il progetto di questa tesi riguarda un algoritmo che sia in grado di risolvere all'ottimo il problema della dislocazione e della turnazione di squadre di pronto intervento. Ogni squadra è posizionata nella propria sede, ovvero la località in cui essa lavora, ognuna di esse deve sorvegliare un insieme di luoghi, che rappresentano delle località sensibili nel territorio, per ogni coppia di luogo-sede esiste un valore che ne rappresenta l'effettiva distanza tra il luogo e la sede in questione. Dato che una squadra non può essere attiva 24 ore su 24 è necessario introdurre i turni, ovvero delle suddivisioni temporali all'interno delle quali solo alcune squadre possono essere attive contemporaneamente. L'assegnare una squadra ad uno specifico turno determina il variare della distanza tra un determinato luogo e la sua squadra di copertura. Il fine è quello di ottenere una dislocazione e turnazione che renda minima la distanza massima tra un luogo e la sede. La motivazione dello studio effettuato sul problema è quello di realizzare uno strumento che sia in grado di aiutare la protezione civile, o qualsiasi associazione o azienda che abbia la necessità sorvegliare dei luoghi mediante l'ausilio di squadre da disporre su più turni. La soluzione ottima, che minimizza la distanza massima, serve per minimizzare il tempo massimo di risposta che una squadra ha nei confronti di un determinato luogo; in tal modo si assicura una copertura di tutti i luoghi sensibili del territorio in modo ottimale.

## 1.2 Descrizione informale

Il problema affrontato, come detto in precedenza, è quello di localizzare squadre di pronto intervento da allertare in caso di emergenza e di assegnare a

ciascuna squadra una zona di competenza. La zona di competenza identifica l'insieme dei luoghi da sorvegliare. I luoghi rappresentano delle località che necessitano di una sorveglianza da parte di squadre di pronto intervento, perché sono luoghi di interesse elevato quali ospedali, scuole, musei, centri commerciali o luoghi di culto, in cui si concentra un elevato numero di persone, oppure industrie ad elevata pericolosità od oggetto di possibili attentati. Sia i luoghi che le sedi delle squadre devono essere scelti sulla mappa. Ad ogni coppia di luogo-sede è assegnato un valore che è la misura della distanza tra il luogo e la sede, espressa in termini di lunghezza del percorso che separa il luogo dalla sede, oppure in termini di quantità di tempo necessario ad una squadra per raggiungere il luogo. Ogni luogo deve essere sempre tenuto sotto controllo da una squadra quindi bisogna assicurare che in ogni turno vi sia una squadra che sorvegli quel determinato luogo. Ogni squadra può essere scelta per un solo turno e di conseguenza un luogo può essere assegnato ad una squadra in un determinato turno solo se essa è utilizzabile in quel turno. Inoltre all'interno di ogni turno devono essere attive solo un determinato numero di squadre, bisogna quindi garantirne una equa distribuzione nei vari turni, in modo da non avere turni sovraccarichi, con numerose squadre attive aventi ciascuna un numero esiguo di luoghi da sorvegliare, e turni in cui vi sono poche sedi con un numero elevato di luoghi da controllare. Il criterio per valutare una soluzione è la distanza massima tra un qualsiasi luogo ed una qualsiasi squadra soggetti ad un assegnamento.

## 1.3 Formulazione matematica

Il problema descritto precedentemente può essere formulato matematicamente ed il modello che ne deriva è descritto in seguito. Per convenzione assumo che  $i$  è l'indice riferito ai luoghi, mentre  $j$  è l'indice delle sedi e  $k$  è l'indice dei turni, di conseguenza quando si parlerà di  $i$ ,  $j$  e  $k$  ci si riferisce rispettivamente a luoghi, sedi e turni del problema.

Dati del problema:

- $I$  luoghi da sorvegliare
- $J$  sedi delle squadre di pronto intervento
- $K$  turni
- $d_{ij}$  distanza tra il luogo  $i$  e la sede  $j$  misurata in termini di tempo necessario per percorrerla

Variabili:

- $x_{ijk}$  variabile per l'assegnamento del luogo  $i$  alla squadra  $j$  nel turno  $k$
- $y_{jk}$  variabile di assegnamento della squadra  $j$  al turno  $k$

Formulazione del problema:

$$\begin{aligned} \min \quad & Z \\ \text{s.t.} \quad & \sum_{k=1}^K y_{jk} = 1 \quad \forall j = 1, \dots, J \end{aligned} \quad (1.1)$$

$$\sum_{j=1}^J x_{ijk} = 1 \quad \forall i = 1, \dots, I \quad \forall k = 1, \dots, K \quad (1.2)$$

$$x_{ijk} \leq y_{jk} \quad \forall i = 1, \dots, I \quad \forall j = 1, \dots, J \quad \forall k = 1, \dots, K \quad (1.3)$$

$$\sum_{j=1}^J y_{jk} \geq \lfloor \frac{|J|}{|K|} \rfloor \quad \forall k = 1, \dots, K \quad (1.4)$$

$$\sum_{j=1}^J y_{jk} \leq \lceil \frac{|J|}{|K|} \rceil \quad \forall k = 1, \dots, K \quad (1.5)$$

$$d_{ij} x_{ijk} \leq Z \quad \forall i = 1, \dots, I \quad \forall j = 1, \dots, J \quad \forall k = 1, \dots, K \quad (1.6)$$

$$x_{ijk} \in \{0, 1\} \quad \forall i = 1, \dots, I \quad \forall j = 1, \dots, J \quad \forall k = 1, \dots, K \quad (1.7)$$

$$y_{jk} \in \{0, 1\} \quad \forall j = 1, \dots, J \quad \forall k = 1, \dots, K \quad (1.8)$$

- Il vincolo 1.1 impone che una squadra di pronto intervento sia attiva in un unico turno, dato che consideriamo attiva la sede  $j$  nel turno  $k$  se la variabile  $y_{jk}$  assume valore 1, per cui la sommatoria su  $k$  di tutte le variabili  $y_{jk}$  deve essere uguale a 1.
- Con il vincolo 1.2 si impone che ogni luogo in ogni turno deve essere sorvegliato da una sola squadra, quindi consideriamo il luogo  $i$  sorvegliato dalla squadra  $j$  nello specifico turno  $k$  se la variabile  $x_{ijk}$  assume il valore 1, per tanto il vincolo deve essere formulato come sommatoria

su  $j$  di  $x_{ijk}$  deve essere uguale a 1 da ripetere per ogni luogo  $i$  e ogni turno  $k$ .

- Il vincolo 1.3 ha lo scopo di vietare di assegnare un luogo  $i$  ad una squadra  $j$  in un turno  $k$  se la squadra non è di servizio in quel turno, per cui per ogni luogo, squadra e turno la relativa variabile  $x_{ijk}$  deve assumere un valore minore o al più uguale a quello della rispettiva variabile  $y_{jk}$ .
- Grazie al vincolo 1.4 è possibile limitare il numero minimo di squadre contemporaneamente attive in ogni turno, tale numero è l'intero inferiore del risultato della divisione tra il numero di squadre ed il numero di turni.
- Con il vincolo 1.5 si impone il limite massimo al numero di squadre contemporaneamente attive in ogni turno e tale valore è l'intero superiore del risultato della divisione tra il numero di sedi ed il numero di turni. Ponendo sia un limite inferiore che un limite superiore si distribuiscono equamente le squadre fra i vari turni, evitando così di avere turni con tante squadre in servizio e turni con poche squadre attive.
- Il vincolo 1.6 permette il calcolo della distanza massima, infatti questo vincolo serve per ottenere il valore di tale distanza, per ogni luogo, squadra e turno  $Z$  assume il valore massimo relativo ad ogni assegnamento  $x_{ijk}$ .
- Con il vincolo 1.7 impone che i valori delle variabili  $x_{ijk}$  siano unicamente binari in tal modo se una  $x_{ijk}$  vale 1 allora il luogo  $i$  è assegnato alla squadra  $j$  nel turno  $k$ , se vale 0 l'assegnamento non è avvenuto.



- Il vincolo 1.8 impone anch'esso che il valore delle variabili  $y_{jk}$  siano binari, si può attribuire così il seguente significato: se  $y_{jk} = 1$  la squadra  $j$  è di servizio nel turno  $k$ , se invece  $y_{jk} = 0$  significa che la squadra  $j$  non è allertata per il turno  $k$ .

Il modello del problema descritto precedentemente può essere classificato come un problema di programmazione lineare intera.

## 1.4 Lower-Bound del problema

Il lower-bound rappresenta un limite al valore della soluzione ottima del problema, in questo caso il problema consiste nel minimizzare una distanza massima per cui è possibile calcolare un valore di limite inferiore al di sotto del quale il valore della soluzione ottima non può scendere. Per ottenere tale valore è sufficiente calcolare la distanza massima che deriva dall'assegnare ogni luogo, in ogni turno, alla squadra con la distanza minima ed in ogni turno la squadra assegnata deve essere diversa. In tal modo si ottiene una soluzione inammissibile per il problema ma fornisce un valore di lower-bound molto valido.

Lo pseudo codice dell'algoritmo per il calcolo del lower-bound è il seguente:

$LB := 0$  valore del lower-bound  
 $d_{max} := 0$  valore della distanza massima  
 $d_{min} := +\infty$  valore della distanza minima  
 $j_{min}$  riferimento alla squadra con la distanza minima  
 $V$  = lista delle sedi utilizzate

```
for  $i := 1$  to  $I$   
     $V := \emptyset$   
    for  $k := 1$  to  $K$   
         $d_{min} := +\infty$   
        for  $j := 1$  to  $J$   
            if  $(d_{min} < d_{ij}) \ \& \ (j \notin V)$  then  
                 $d_{min} := d_{ij}$   
                 $j_{min} := j$   
            if  $(d_{min} > d_{max})$  then  $d_{min} := d_{max}$   
         $V \leftarrow j_{min}$   
 $LB := d_{max}$ 
```

Figura 1.1: Pseudo-codice dell'algoritmo per il calcolo del lower-bound

Il valore di lower-bound così calcolato permette all'algoritmo creato ad hoc per questo problema di essere più efficace ottenendo così la soluzione ottima in un tempo inferiore.

## Capitolo 2

### Solutori general-purpose

## 2.1 Programmazione lineare intera

Il problema che questa tesi si propone di risolvere è un problema di programmazione lineare intera [8]. Mediante l'algoritmo del simplesso possiamo ottenere la soluzione ottima del rilassamento continuo del problema. Il rilassamento continuo è il problema di PL che si ottiene dal problema di PLI trascurando i vincoli di integralità. Dal rilassamento continuo del problema è possibile ottenere la soluzione del problema originario mediante l'utilizzo di un algoritmo di tipo Branch-and-Bound, che consente di trovare la soluzione ottima di problemi di tipo NP-Hard. I sottoproblemi che l'algoritmo di Branch-and-Bound risolve sono il rilassamento continuo del problema originario con l'aggiunta di vincoli per l'assegnazione di valori alle variabili del problema, fissare il valore di una variabile in un determinato sottoproblema significa assegnare a quel sottoproblema un sottoinsieme delle soluzioni del problema originario in cui la variabile fissata assume quel determinato valore.

## 2.2 Solutori general-purpose

Esistono dei solutori di tipo general-purpose che sono in grado di risolvere problemi di svariati tipi, questi incorporano al loro interno l'algoritmo del simplesso e un algoritmo di Branch-and-Bound potendo così risolvere all'ottimo sia problemi di programmazione lineare che di programmazione lineare intera. Utilizzare dei solutori general-purpose può essere il primo passo per risolvere un problema di cui non si conosce un algoritmo specifico per la sua risoluzione, questi solutori sono in grado di risolvere in modo ottimo i problemi, ma peccano in fatto di efficienza nei tempi di calcolo. Tra i numerosi solutori general-purpose ho deciso di utilizzare la libreria "GNU Linear Pro-

gramming Kit” [3] della GNU : è una libreria di libero utilizzo che può essere integrata in programmi C/C++, ed è in grado di risolvere problemi di programmazione lineare e di programmazione lineare intera, perciò è adatta alla risoluzione del problema. L’algoritmo creato con la libreria GLPK è utilizzato come pietra di paragone per l’algoritmo ad hoc illustrato in seguito.

## 2.3 Risultati sperimentali

L’algoritmo è stato testato con numerosi problemi di varie dimensioni allo scopo di valutarne le prestazioni riguardanti il tempo impiegato per la risoluzione ottima del problema. In base ai test effettuati ho riscontrato che la risoluzione del rilassamento continuo del problema è ottenibile in tempi ragionevoli anche con problemi di elevate dimensioni; mentre l’esecuzione del Branch-and-Bound richiede un’enorme quantità di tempo. Infatti sottoponendo all’algoritmo problemi di “grandi” dimensioni, sull’ordine di 150 luoghi, 9 sedi e 3 turni, la soluzione ottima è fornita in un tempo di 23390 secondi, circa 6 ore e 30 minuti, date le dimensioni del problema non è certo un tempo di risoluzione soddisfacente. Analizzando i risultati ottenuti con la libreria GLPK, rappresentati nella tabella in figura 2.1, si può notare che le dimensioni dei problemi risolti sono considerevolmente piccole. Se pensiamo alle dimensioni di un problema nella realtà che coinvolge una città intera possiamo stimare un numero di luoghi da sorvegliare dell’ordine di parecchie centinaia di unità, un numero di squadre di pronto intervento di qualche decina, mentre il numero di turni su cui distribuire le squadre può dipendere dal numero di squadre disponibili; se pensiamo di dover sorvegliare i luoghi 24 ore al giorno è plausibile avere un numero di turni compreso tra 3 e 6. Quindi un esempio reale è : 500/600 luoghi, 10/100 squadre, 3/6 turni. Da

queste considerazioni si può notare che l'algoritmo, che utilizza la libreria GLPK, non riuscirebbe a risolvere un problema di queste dimensioni in un tempo accettabile. Le istanze dei problemi utilizzati per le prove sono istanze di problemi della OR-Library opportunamente adattate per il problema in questione.

File	Luoghi	Sedi	Turni	Numero righe	Numero colonne	Numero elementi	Numero Yjk	Numero Xijk	Soluzione continua	Soluzione intera	Durata Totale
gap15-5-2.txt	15	5	2	337	161	54257	10	150	4,38	20,00	2
mdmkp20-6-2.txt	20	6	2	528	253	133584	12	240	101,69	612,00	3
csp20-6-3.txt	20	6	3	789	379	299031	18	360	105,40	567,00	3
csp20-9-6.txt	20	9	6	2295	1135	2604825	54	1080	151,93	1362,00	18
assign30-9-5.txt	30	9	5	2864	1396	3998144	45	1350	7,69	93,00	915
gap35-7-4.txt	35	7	4	2111	1009	2129999	28	980	3,29	27,00	8
pmed37-10-3.txt	37	10	3	2344	1141	2674504	30	1110	9,27	93,00	36
mdmkp40-6-2.txt	40	6	2	1048	493	516664	12	480	107,00	670,00	9
pmed40-8-4.txt	40	8	4	2732	1313	3587116	32	1280	10,47	105,00	17
csp41-9-5.txt	41	9	5	3909	1891	7391919	45	18454	161,04	1447,00	491
pmed50-8-4.txt	50	8	4	3412	1633	5571796	32	1600	14,66	131,00	29
dea50.txt	50	10	5	5265	2551	13431015	50	2500	3,58	37,00	1968
assign50-12-7.txt	50	12	7	8769	4285	37575164	84	4200	5,42	98,00	18032
cap50-16-4.txt	50	16	4	6620	3265	21614300	64	3200	12529,41	229188,00	341
cap50-16-5.txt	50	16	5	8271	4081	33753951	80	4000	272314,00	1361570,00	46
pmed60-8-4.txt	60	8	4	4092	1953	7991676	32	1920	15,75	158,00	65
estein80.txt	80	15	5	12420	6076	75463920	75	6000	443019,42	nessuna	14526
mdmkp100-10-3.txt	100	10	3	6313	3031	19134704	30	3000	62,85	698,00	9543
assign150-9-3.txt	150	9	3	8562	4078	34915836	27	4050	7,49	77,00	23409

Figura 2.1: Tabella dei risultati sperimentali

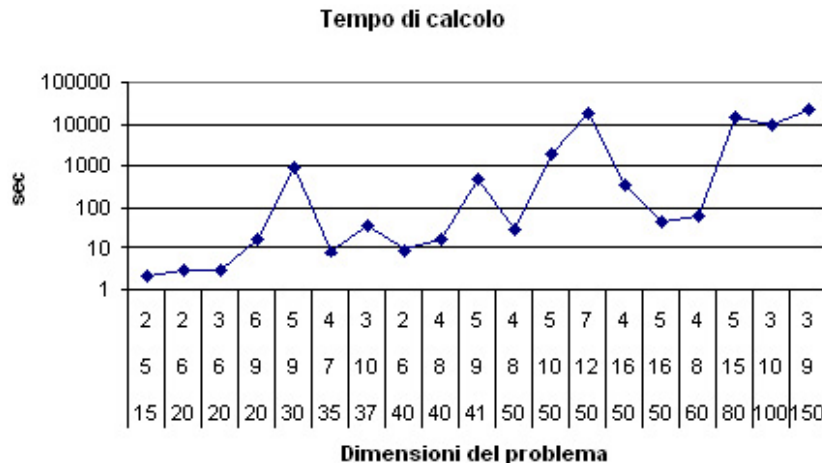


Figura 2.2: Grafico dei tempi di calcolo

Osservando il grafico è possibile notare un notevole incremento del tempo di calcolo al crescere delle dimensioni del problema, per problemi di piccole dimensioni l'algoritmo si comporta abbastanza bene ottenendo soluzioni ottime in tempi accettabili; tuttavia aumentando anche di poco le dimensioni del problema il tempo di calcolo cresce esponenzialmente, raggiungendo valori non più accettabili. Il motivo per il quale si ottengono dei tempi di calcolo molto elevati sta nel fatto che gli algoritmi di tipo Branch-and-Bound trovano velocemente la soluzione ottima ma impiegano molto tempo per garantirne l'ottimalità. In conclusione la risoluzione del problema mediante la libreria GLPK garantisce di fornire soluzioni ottime, ma le prestazioni, in termini di tempo di calcolo, sono decisamente scarse. Per problemi di piccole dimensioni l'algoritmo funziona abbastanza bene risolvendo il problema in tempi ragionevoli, incrementando le dimensioni del problema il risultato fornito è sempre l'ottimo ma il tempo di calcolo non è più accettabile. Allo scopo di ridurre i tempi di calcolo ed incrementare le dimensioni del problema da risolvere all'ottimo ho deciso di utilizzare il rilassamento Lagrangeano.



## Capitolo 3

### Un algoritmo Lagrangeano

### 3.1 Rilassamento Lagrangeano

Il rilassamento lagrangeano è una tipologia di rilassamento di problemi, nata nei primi anni '70 dal lavoro di Held e Karp sul problema del TSP [10], consiste nel rilassare alcuni vincoli penalizzandone la violazione tramite termini aggiunti alla funzione obiettivo, in tal modo si semplifica, dal punto di vista computazionale, il problema originario. I vincoli eliminati sono detti vincoli rilassati, le soluzioni che violano tali vincoli tendono ad essere meno convenienti delle soluzioni che li rispettano, a causa delle penalità inserite all'interno della funzione obiettivo. Tali penalità sono dette moltiplicatori lagrangeani. La soluzione ottima del problema rilassato è un bound duale al valore ottimo del problema originale, quindi per avere un bound duale il più vicino possibile al valore ottimo bisogna scegliere in modo ottimale i valori dei moltiplicatori lagrangeani risolvendo il problema duale:

$$\min_{\lambda \geq 0} \left\{ \max_{x \in X} \left\{ cx + \lambda(b - Ax) \right\} \right\}$$

Tuttavia non è garantito che si possa ottenere un valore di bound duale uguale al valore ottimo, avendo così la “Duality Gap”. La soluzione ottima del duale fornisce il miglior bound duale del rilassamento. In generale il rilassamento lagrangeano è più forte del rilassamento continuo, cioè fornisce dei bound duali migliori. Il problema lagrangeano duale consiste nel minimizzare una funzione convessa lineare a tratti, senza minimi locali e non derivabile, corrisponde ad un problema di PL con moltissimi vincoli tanti quanti i punti interi di  $X$ , per risolvere il duale lagrangeano esistono diversi metodi di approssimazione:

- algoritmo del sottogradiente
- multiplier adjustment

- dual ascent

Tra tutti i tipi di approssimazione quella che si è dimostrata sperimentalmente migliore è l'algoritmo del sottogradiente perché offre dei bound migliori, le prove sperimentali sono state fatte da J.E. Beasley descritte nella sua dispensa "Lagrangian Relaxation" [2]. La formulazione del rilassamento lagrangeano è stata ricavata dalla formulazione originale del problema rilassando i vincoli  $x_{ijk} \leq y_{jk} \forall i = 1, \dots, I \forall j = 1, \dots, J \forall k = 1, \dots, K$ . Tali vincoli servono per limitare l'assegnazione del luogo  $i$  alla squadra  $j$  nel turno  $k$  se e solo se la squadra  $j$  è di servizio nel turno  $k$ . Nel nostro caso il numero di vincoli rilassati è  $I \cdot J \cdot K$ , ovvero il prodotto del numero di luoghi, del numero di squadre e del numero di turni. La formulazione del rilassamento che ne deriva è:

$$\begin{aligned}
\min \quad & Z + \sum_{i=1}^I \sum_{j=1}^J \sum_{k=1}^K \lambda_{ijk} (x_{ijk} - y_{jk}) \\
\text{s.t.} \quad & \sum_{k=1}^K y_{jk} = 1 & \forall j = 1, \dots, J \\
& \sum_{j=1}^J x_{ijk} = 1 & \forall i = 1, \dots, I \quad \forall k = 1, \dots, K \\
& \sum_{j=1}^J y_{jk} \geq \lfloor \frac{|J|}{|K|} \rfloor & \forall k = 1, \dots, K \\
& \sum_{j=1}^J y_{jk} \leq \lceil \frac{|J|}{|K|} \rceil & \forall k = 1, \dots, K \\
& d_{ij} x_{ijk} \leq Z & \forall i = 1, \dots, I \quad \forall j = 1, \dots, J \quad \forall k = 1, \dots, K \\
& x_{ijk} \in \{0, 1\} & \forall i = 1, \dots, I \quad \forall j = 1, \dots, J \quad \forall k = 1, \dots, K \\
& y_{jk} \in \{0, 1\} & \forall j = 1, \dots, J \quad \forall k = 1, \dots, K.
\end{aligned}$$

Dalla formulazione del rilassamento si nota che nella funzione obiettivo sono state inserite le penalità relative al vincolo rilassato, costituite dal valore

della violazione del vincolo  $(x_{ijk} - y_{jk})$  e dal coefficiente  $\lambda_{ijk}$ , detto moltiplicatore lagrangeano, questi moltiplicatori hanno lo scopo di accentuare o attenuare la violazione del vincolo a cui sono legati. Analizzando la formulazione del rilassamento si nota che essa è composta da due sottoproblemi indipendenti tra loro, infatti il vincolo rilassato era l'unico che legava sia le variabili  $x$  che  $y$ , così è possibile dividere il problema in due sottoproblemi indipendenti tra loro. Il primo sottoproblema ha come variabili le sole  $x$ , mentre il secondo ha le sole  $y$ , in tal modo essi possono essere risolti separatamente.

$$\begin{aligned}
\min \quad & Z + \sum_{i=1}^I \sum_{j=1}^J \sum_{k=1}^K \lambda_{ijk} x_{ijk} \\
\text{s.t.} \quad & \sum_{j=1}^J x_{ijk} = 1 & \forall i = 1, \dots, I \quad \forall k = 1, \dots, K \\
& d_{ij} x_{ijk} \leq Z & \forall i = 1, \dots, I \quad \forall j = 1, \dots, J \quad \forall k = 1, \dots, K \\
& x_{ijk} \in \{0, 1\} & \forall i = 1, \dots, I \quad \forall j = 1, \dots, J \quad \forall k = 1, \dots, K.
\end{aligned}$$

Il sottoproblema 1 serve per risolvere il problema dell'associazione dei luoghi all'interno dei vari turni alle squadre di pronto intervento.

$$\begin{aligned}
max \quad & \sum_{j=1}^J \sum_{k=1}^K \left( \sum_{i=1}^I \lambda_{ijk} \right) y_{jk} \\
s.t. \quad & \sum_{k=1}^K y_{jk} = 1 & \forall j = 1, \dots, J \\
& \sum_{j=1}^J y_{jk} \geq \lfloor \frac{|J|}{|K|} \rfloor & \forall k = 1, \dots, K \\
& \sum_{j=1}^J y_{jk} \leq \lceil \frac{|J|}{|K|} \rceil & \forall k = 1, \dots, K \\
& y_{jk} \in \{0, 1\} & \forall j = 1, \dots, J \quad \forall k = 1, \dots, K.
\end{aligned}$$

Il sottoproblema 2 serve per ottenere la distribuzione ottima delle squadre nei vari turni, il problema è molto simile, come formulazione, al problema del problema di flusso massimo a costo minimo, differisce per la direzione di ottimizzazione infatti nel problema di flusso si minimizzano i costi, mentre nel sottoproblema 2 si devono massimizzare, ma tale differenza è superabile invertendo la direzione di ottimizzazione e scambiando di segno i costi, così ottenendo la seguente formulazione:

$$\begin{aligned}
min \quad & \sum_{j=1}^J \sum_{k=1}^K \left( - \sum_{i=1}^I \lambda_{ijk} \right) y_{jk} \\
s.t. \quad & \sum_{k=1}^K y_{jk} = 1 & \forall j = 1, \dots, J \\
& \sum_{j=1}^J y_{jk} \geq \lfloor \frac{|J|}{|K|} \rfloor & \forall k = 1, \dots, K \\
& \sum_{j=1}^J y_{jk} \leq \lceil \frac{|J|}{|K|} \rceil & \forall k = 1, \dots, K \\
& y_{jk} \in \{0, 1\} & \forall j = 1, \dots, J \quad \forall k = 1, \dots, K.
\end{aligned}$$

Utilizzando tale formulazione è possibile risolvere il problema di flusso massimo a costo minimo, infatti minimizzare dei costi negativi che equivale a massimizzare costi positivi, utilizzando l'algoritmo di Ford-Fulkerson, realizzato appositamente per i problemi di flusso massimo. La scomposizione applicata al problema del rilassamento lagrangeano permette di risolvere i due sottoproblemi separatamente e successivamente combinare le soluzioni trovate ottenendo così un bound duale.

## 3.2 Problema Lagrangeano duale e algoritmo del sottogradiente

Il problema lagrangeano duale si pone l'obiettivo di minimizzare i valori dei moltiplicatori lagrangeani, scegliendo così in modo ottimale il loro valore per avere un valore del bound duale uguale al valore ottimo. Non è garantito comunque che anche scegliendo in modo ottimale i moltiplicatori lagrangeani si ottenga la soluzione ottima del problema originario, anche se tale soluzione è ammissibile. L'algoritmo del sottogradiente[2] è un algoritmo di ricerca locale che si sposta dalla soluzione corrente  $\lambda^k$  alla soluzione successiva  $\lambda^{k+1}$  compiendo un passo di ampiezza  $\mu^k$  nella direzione opposta a quella del sottogradiente. L'algoritmo del sottogradiente è un algoritmo iterativo che, da un insieme di valori iniziali dei moltiplicatori lagrangeani, evolve generando nuovi valori dei moltiplicatori in base ai valori precedenti, alla violazione del vincolo e dal passo che si vuole compiere per spostarsi. Ad ogni iterazione viene generato un lower bound del problema, tale valore può avere un andamento non omogeneo, infatti si può passare da un lower bound buono ad un valore peggiore.

I punti essenziali dell'algoritmo del sottogradiente consigliati da J.E. Beasley sono:

1. decidere il valore del parametro  $\pi$ , che rappresenta un fattore di ampliamento o riduzione del passo dell'algoritmo, tale valore deve essere  $0 < \pi \leq 2$
2. trovare una soluzione iniziale ammissibile  $Z_{UB}$  da poter utilizzare come upper bound, tale soluzione è ottenuta mediante l'euristica iniziale
3. decidere i valori iniziali dei moltiplicatori lagrangeani, anche se da varie prove sperimentali condotte da J.E. Beasley[2] e testando l'algoritmo per la risoluzione del problema studiato da questa tesi si nota che non sono rilevanti, quindi possono essere inizializzati a valori casuali oppure semplicemente al valore 0
4. risolvere il rilassamento lagrangeano con i correnti valori dei moltiplicatori lagrangeani allo scopo di ottenere una soluzione  $\bar{x}$  ed il valore del lower bound  $Z_{LB}$ , ciò corrisponde a risolvere il sottoproblema 1 e 2 e combinare le soluzioni ottenute
5. calcolo dell'euristica lagrangeana al fine di rendere ammissibile per il problema originario la soluzione trovata risolvendo il rilassamento lagrangeano
6. riduzione del problema che consiste nel fissare il valore delle variabili in modo da ridurre man mano le dimensioni del problema riducendone la complessità computazionale e velocizzandone la risoluzione.
7. calcolo dei sottogradienti  $G$  per i vincoli rilassati valutati con la corrente

soluzione per cui si ha:

$$G_{ijk} = x_{ijk} - y_{jk} \quad \forall i = 1, \dots, I \quad \forall j = 1, \dots, J \quad \forall k = 1, \dots, K$$

8. definizione del passo scalare con :  $T = \frac{\pi(Z_{UB} - Z_{LB})}{\sum_{i=1}^I \sum_{j=1}^J \sum_{k=1}^K (G_{ijk})^2}$  la dimensione del passo dipende dalla differenza tra il lower bound  $Z_{LB}$  corrente e l'upper bound  $Z_{UB}$ , e dal parametro  $\pi$  definito precedentemente e dai sottogradienti calcolati prima
9. aggiornare i moltiplicatori lagrangeani secondo la seguente formula:  

$$\lambda_{ijk} = \max\{0, \lambda_{ijk} + TG_{ijk}\} \quad \forall i = 1, \dots, I \quad \forall j = 1, \dots, J \quad \forall k = 1, \dots, K$$
10. ritornare al passo 4 per calcolare il nuovo lower bound con i nuovi valori dei moltiplicatori lagrangeani.

L'algoritmo così descritto continuerebbe le iterazioni all'infinito per cui è necessario inserire delle condizioni di terminazione dell'algoritmo, occorre un test per verificare se la miglior soluzione trovata è quella ottima, quindi ogni volta che si calcola un lower bound bisogna controllare se il valore coincide con quello della soluzione migliore finora trovata, in caso positivo si ha ottenuto la soluzione ottima e l'algoritmo deve terminare. Un'altra condizione di terminazione è quella per cui tutti i sottogradienti  $G_{ijk}$  hanno valore nullo di conseguenza tutti i vincoli rilassati sono rispettati ed ho ottenuto la soluzione ottima. L'algoritmo può terminare anche senza la garanzia che la soluzione trovata sia ottima, perché è possibile limitare il numero di iterazioni da deve compiere oppure eseguire un test sul valore del parametro  $\pi$ , in caso che il valore diventasse troppo piccolo non è conveniente proseguire con le iterazioni dell'algoritmo perché il passo risulta essere troppo piccolo. In caso che l'algoritmo termini senza garanzia di ottimalità è utile valutare il Gap, ovvero la differenza tra la miglior soluzione trovata ed il massimo lower bound,



questo parametro serve per stimare il possibile errore, in termini di differenza di valore, che la soluzione trovata ha rispetto alla ipotetica soluzione ottima,  $Gap = Z_{UB} - Z_{LBmax}$ . Il lower bound massimo rappresenta il minimo valore che una soluzione può assumere, cioè se esiste una soluzione ottima essa non può essere inferiore in valore al massimo lower bound. L'algoritmo del sottogradient, risolvendo il rilassamento lagrangeano del problema, permette di trovare una soluzione ammissibile per il problema originario, che può essere ottima e con garanzia di ottimalità, tale garanzia è data dalle condizioni di uscita dall'algoritmo. In caso che la soluzione non sia garantita come ottima è possibile avere una stima del possibile errore insito nel valore della soluzione trovata, avendo così un'informazione riguardante la qualità della soluzione.

### 3.3 Algoritmo risolutivo del sottoproblema 1

Il sottoproblema 1 è la parte del rilassamento lagrangeano incentrata sull'assegnazione nei vari turni dei luoghi da tenere sotto controllo da parte delle squadre di pronto intervento, ogni coppia di luogo e squadra è caratterizzata da una distanza  $d_{ij}$  che li separa e ad ogni possibile assegnamento  $x_{ijk}$  è legato un valore di penalità  $\lambda_{ijk}$  riferita al relativo vincolo rilassato. Si vuole ottenere l'assegnamento dei luoghi alle squadre in modo che la distanza massima tra tutte le coppie assegnate sia la minima possibile e che la somma delle penalità derivate dall'assegnamento scelto sia anche essa minima. Il sottoproblema 1 è divisibile in tanti sottoproblemi, uno per ogni turno, indipendenti l'uno dall'altro perché la scelta di assegnare il luogo  $i$  alla squadra  $j$  nel turno  $k$  è indipendente dagli assegnamenti presenti negli altri turni, in sostanza ogni turno è a se stante e forma un sottoproblema autonomo. Inizialmente ho provato a realizzare un algoritmo semplice che scorrendo tutte

le distanze, ordinate precedentemente in modo crescente, sceglie di volta in volta i valori minimi delle penalità  $\lambda_{ijk}$ , ogni penalità scelta corrisponde ad un assegnamento del luogo  $i$  alla sede  $j$  nel turno  $k$ , per cui la variabile  $x_{ijk}$  assume il valore 1. L'algoritmo può essere descritto mediante la seguente formula:

$$\forall Z \quad \min_j \left\{ \lambda_{ijk} \mid d_{ij} \leq Z \right\} \quad \forall i = 1, \dots, I \quad \forall k = 1, \dots, K$$

L'algoritmo che ne deriva procede iterativamente per ogni turno scorrendo tutte le distanze, ordinate in modo crescente, ad ognuna di esse si valuta il valore della penalità legata al turno e alla coppia luogo-squadra a cui la distanza fa riferimento, si accetta la nuova distanza se si ha un miglioramento della penalità, ovvero si migliora la penalità minima associata ad ogni luogo, in tal modo si assegna ogni luogo alla sede che ha la minima penalità associata. Per il corretto funzionamento dell'algoritmo specifico è necessario avere una lista ordinata in modo crescente di tutte le distanze del problema, in cui ogni elemento contiene il valore della distanza ed i riferimenti al luogo e alla sede a cui essa si riferisce, tale lista viene creata al caricamento dei dati del problema e rimane costante durante tutta l'esecuzione dell'algoritmo.

$L$  = lista di distanze  $d_{ij} \forall i, \forall j$  ordinate in modo crescente

$UB := +\infty$

**for**  $i := 1$  **to**  $I$

**for**  $k := 1$  **to**  $K$

$\lambda min_{ik} = +\infty$

$t := 0$

**repeat**

$t := t + 1$

$(i, j, d) := L(t)$

**for**  $k := 1$  **to**  $K$

**if**  $(\lambda_{ijk} < \lambda min_{ik})$  **then**

$\lambda min_{ik} = \lambda_{ijk}$

$jmin_{ik} = j$

$\Lambda = \sum_{i=1}^I \sum_{k=1}^K \lambda min_{ik}$

**if**  $(\Lambda + d < UB)$  **then**

$UB := \Lambda + d$

$Z^* := d$

$\forall i, k \quad x_{ijk}^* := \begin{cases} 1 & \text{se } j = jmin_{ik} \\ 0 & \text{altrimenti} \end{cases}$

**until**  $(t = |L|)$

Figura 3.1: Pseudo-codice dell'algoritmo risolutivo per il sottoproblema 1

Analizzando la formulazione dell'algoritmo si può notare che la sua complessità è dell'ordine di  $O(I \cdot J \cdot K)$ , tale valore risulta essere molto valido. Dato che tale algoritmo deve essere incorporato all'interno dell'algoritmo del sottogradiente è necessario che esso abbia la complessità minima possibile in modo da velocizzarne l'esecuzione. L'algoritmo deve essere ripetuto per ogni turno, così facendo si ottiene l'assegnazione ottima dei luoghi alle sedi nei vari turni, i risultati ottenuti sono più che soddisfacenti perché l'algoritmo fornisce soluzioni ottime che confrontate con quelle ottenute con l'algoritmo che utilizza la libreria GLPK risultano identiche dal punto di vista del valore della funzione obiettivo, e come assegnamento sono equivalenti. Il vantaggio di utilizzare l'algoritmo specifico è la complessità computazionale che è notevolmente inferiore e ciò ha favorito la diminuzione del tempo di calcolo. Infatti da prove sperimentali ho rilevato che il tempo di calcolo è diminuito notevolmente, ottenendo così un notevole risparmio di tempo di calcolo globale dell'algoritmo.

Dimensioni del problema			GLPK		Algoritmo $O(I \cdot J \cdot K)$	
Luoghi	Sedi	Turni	Tempo	Risultato	Tempo	Risultato
1000	10	3			0,531	817,4271
500	10	3			0,141	744,6328
200	10	3			0,015	782,0447
100	10	3	5263	752,48	0,007	752,48
50	10	3	385	688,0548	0,005	688,0548
20	10	3	67	607,635	0,002	607,635
10	10	3	9	543,724	0,0001	543,724

Figura 3.2: Tabella di confronto dei risultati del sottoproblema 1

I dati presenti nella tabella di comparazione sono stati ottenuti sottoponendo ai vari algoritmi lo stesso problema con i medesimi dati, in tal modo la comparazione può essere fatta facilmente e senza imprecisioni. Incrementando le dimensioni del problema non si riescono ad ottenere soluzioni mediante l'algoritmo realizzato con la libreria GLPK per cui è sconsigliabile il suo utilizzo, osservando i valori dei tempi di calcolo ottenuti con l'algoritmo specifico si nota che anche incrementando notevolmente le dimensioni tali valori rimangono molto validi.

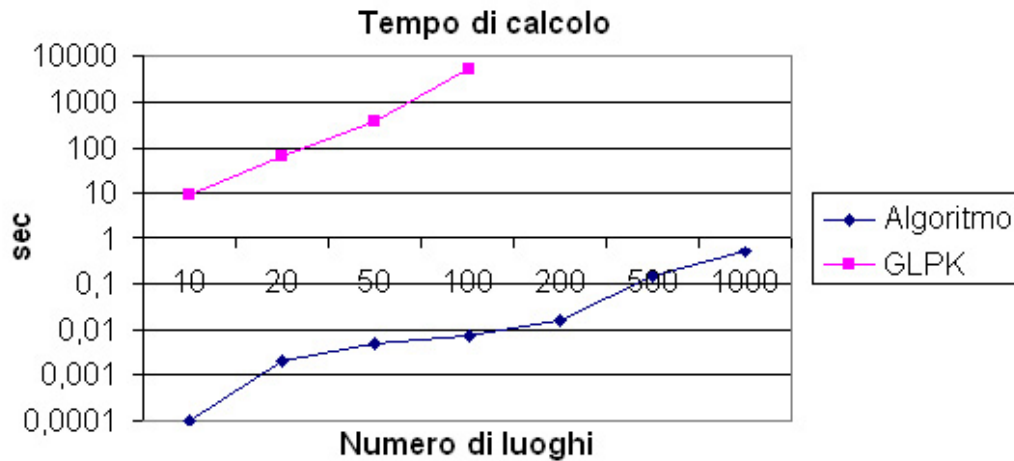


Figura 3.3: Confronto dei tempi di calcolo

I valori riportati nel grafo in figura 3.3 del confronto dei tempi di calcolo sono espressi in scala logaritmica, ciò serve dare maggiore visibilità alla differenza tra le due curve dei tempi. Dal grafico è visibile la differenza di tempo di calcolo fra i due algoritmi, si nota che l'algoritmo GLPK ha una crescita esponenziale dei tempi e che la sua capacità di risolvere problemi di elevate dimensioni è ridotta. Grazie all'algoritmo specifico il tempo di calcolo risulta essere notevolmente inferiore e la curva ha una crescita molto bassa, an-

che incrementando notevolmente le dimensioni del problema. In conclusione ho deciso di inserire all'interno dell'algoritmo del sottogradiente tale algoritmo perché garantisce un notevole risparmio di tempo di calcolo al fronte di soluzioni ottime identiche rispetto a quelle calcolate mediante l'algoritmo GLPK.

### 3.4 Algoritmo risolutivo del sottoproblema 2

Il sottoproblema 2 ha il compito di distribuire in modo ottimo le squadre di pronto intervento nei vari turni, la formulazione con cui si presenta è la seguente:

$$\begin{aligned}
\min \quad & \sum_{j=1}^J \sum_{k=1}^K (-\sum_{i=1}^I \lambda_{ijk}) y_{jk} \\
s.t. \quad & \sum_{k=1}^K y_{jk} = 1 & \forall j = 1, \dots, J \\
& \sum_{j=1}^J y_{jk} \geq \lfloor \frac{|J|}{|K|} \rfloor & \forall k = 1, \dots, K \\
& \sum_{j=1}^J y_{jk} \leq \lceil \frac{|J|}{|K|} \rceil & \forall k = 1, \dots, K \\
& y_{jk} \in \{0, 1\} & \forall j = 1, \dots, J \quad \forall k = 1, \dots, K.
\end{aligned}$$

Tale formulazione richiama la formulazione del problema di flusso massimo a minimo costo, è possibile ricondurre il sottoproblema al problema di flusso massimo a costo minimo mediante l'ausilio di un grafo in cui vi è:

- $s$ , nodo sorgente, da cui parte l'intero flusso
- $t$ , nodo destinazione, a cui l'intero flusso deve arrivare
- $K$ , insieme dei nodi che rappresenta i turni del problema

- $J$ , insieme dei nodi che rappresenta le squadre del problema
- $f$ , nodo fittizio utilizzato per assegnare le squadre restanti ai vari turni

Come si può notare dalla figura 3.4 esiste un arco per ogni coppia di squadra-turno, ed ad ognuno di essi è associato il costo dell'assegnamento della squadra al turno, con capacità 1, ciò permette di assegnare una squadra solamente ad un turno. Ogni arco uscente dal nodo sorgente  $s$  ha capacità  $J \text{ DIV } K$  permettendo così di assegnare ad ogni turno  $J \text{ DIV } K$  squadre, tuttavia restano da assegnare  $J \text{ MOD } K$  squadre, ognuna delle quali deve essere assegnata ad un turno diverso. La funzione del nodo fittizio  $f$  è proprio quella di scegliere i turni ai quali assegnare un'ulteriore squadra.

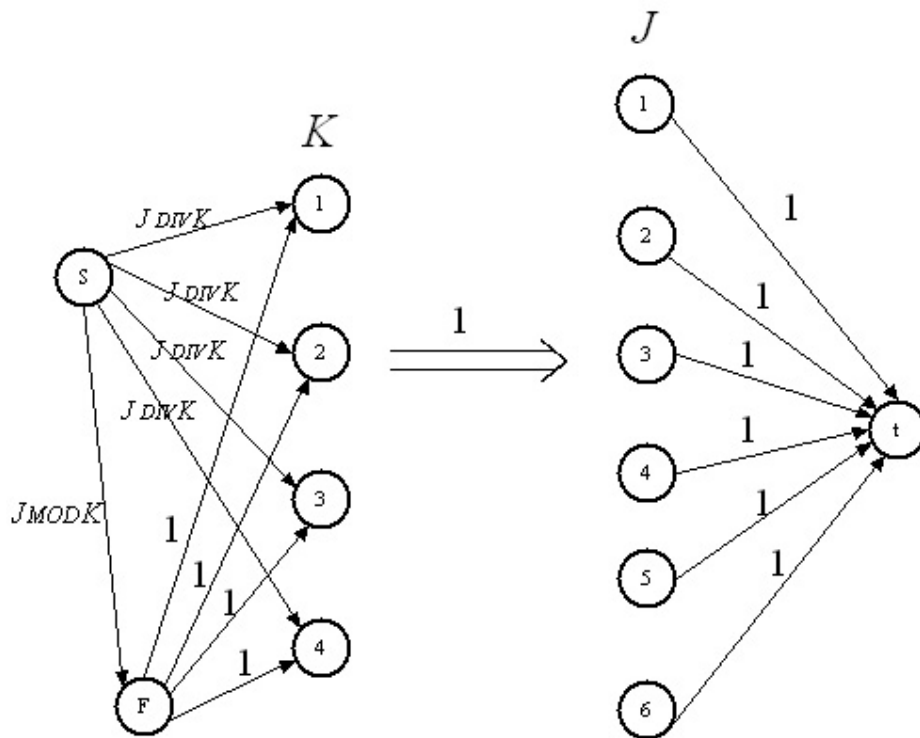


Figura 3.4: Grafico del problema di flusso massimale a costo minimo

Per risolvere il problema di flusso massimo a costo minimo è necessario utilizzare l'algoritmo di Ford-Fulkerson [5], mediante il quale è possibile ottenere il flusso massimo desiderato, e l'algoritmo di Bellman-Ford [7] che permette la ricerca di cammini minimi dal nodo  $s$  al nodo  $t$ . Utilizzando più volte l'algoritmo di Bellman-Ford all'interno dell'algoritmo di Ford-Fulkerson è possibile ottenere il flusso massimo a costo minimo dal nodo  $s$  al nodo  $t$ , ottenendo così la distribuzione ottima delle squadre nei vari turni. Il sottoproblema 2 può essere semplificato nel caso in cui si abbia un numero di squadre multiplo del numero di turni, potendo così raggruppare i due vincoli riguardanti l'equa distribuzione delle squadre in un unico vincolo e la formulazione del sottoproblema risulta essere:

$$\begin{aligned}
\min \quad & \sum_{j=1}^J \sum_{k=1}^K (-\sum_{i=1}^I \lambda_{ijk}) y_{jk} \\
s.t. \quad & \sum_{k=1}^K y_{jk} = 1 & \forall j = 1, \dots, J \\
& \sum_{j=1}^J y_{jk} = \frac{|J|}{|K|} & \forall k = 1, \dots, K \\
& y_{jk} \in \{0, 1\} & \forall j = 1, \dots, J \quad \forall k = 1, \dots, K.
\end{aligned}$$

Grazie a questa semplificazione il sottoproblema può essere ricondotto ad un problema di matching di massima cardinalità a costo minimo, la cui



formulazione è:

$$\begin{aligned}
\min \quad & \sum_{i=1}^N \sum_{j=1}^N c_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{i=1}^N x_{ij} = 1 & \forall j = 1, \dots, N \\
& \sum_{j=1}^N x_{ij} = 1 & \forall i = 1, \dots, N \\
& x_{ij} \in \{0, 1\} & \forall i = 1, \dots, N \quad \forall j = 1, \dots, N.
\end{aligned}$$

Il problema di matching di massima cardinalità insiste su un grafo bipartito  $G(N_1, N_2, E)$  in cui i due insiemi di nodi hanno la stessa cardinalità, mentre la formulazione del sottoproblema 2 insiste su di un grafo bipartito  $G(J, K, E)$  in cui la cardinalità dei due insiemi è diversa, in particolare  $|J| > |K|$ , per portarle alla parità è necessario inserire i nodi mancanti nell'insieme  $K$ , replicando i nodi esistenti più volte, in particolare  $J \text{ DIV } K$  volte. Risolvendo il problema di matching così formulato è possibile assegnare ad ogni squadra un solo turno, rispettando i vincoli del sottoproblema 2. Il problema di matching può essere ridotto nuovamente ad un problema di flusso a costo minimo, per cui trasformare un problema di matching bipartito in un problema di flusso è semplice e basta aggiungere al grafo del problema due nodi fittizi, detti  $s$  e  $t$ , nodo sorgente, da cui parte il flusso e nodo destinazione, nodo in cui arriva il flusso. Il nodo sorgente,  $s$ , ha solo archi in uscita collegati solo ai nodi  $J$  del grafo, mentre i nodi  $K$  sono collegati al nodo di destinazione,  $t$ , tali collegamenti sono formati da archi fittizi. Ogni arco che collega un nodo  $J$  con un nodo  $K$  rappresenta il costo che la squadra deve pagare per essere assegnata al turno. Il grafo di un problema di matching bipartito e del tipo rappresentato in figura 3.5.

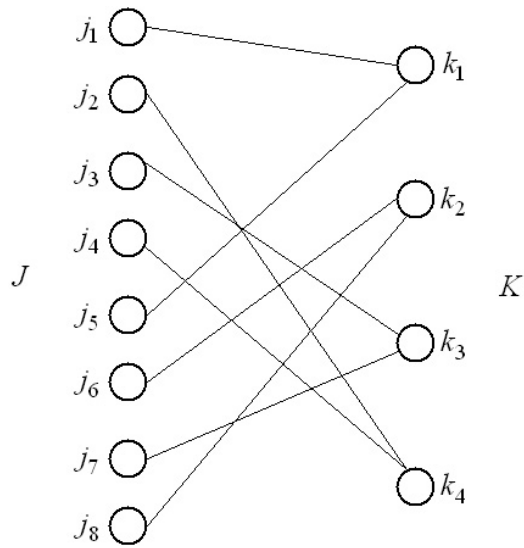


Figura 3.5: Esempio di grafico di matching

Il grafico risultante dalla trasformazione del problema di matching in problema di flusso è:

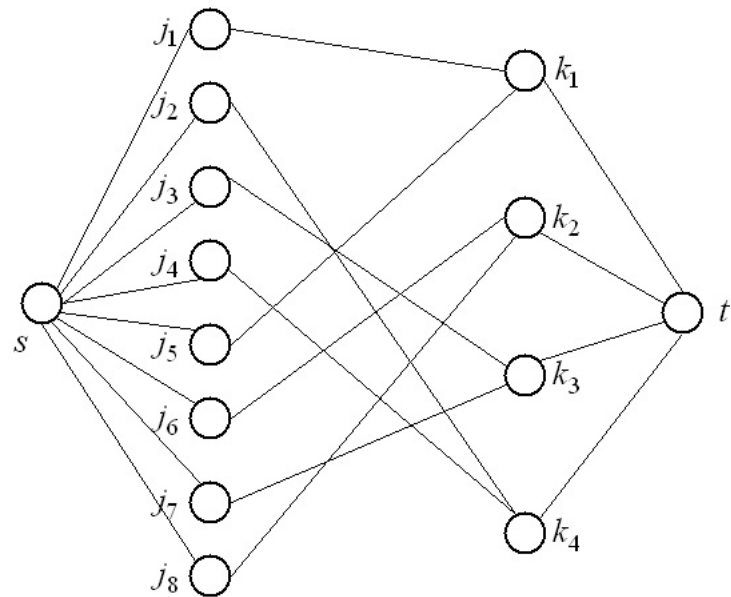


Figura 3.6: Esempio di grafico di flusso

Per risolvere all'ottimo problemi di matching vi sono svariati algoritmi per esempio l'algoritmo di Edmonds, del 1965, è polinomiale, per cui il problema è anch'esso polinomiale, esiste un altro algoritmo in grado di risolvere all'ottimo il problema ed è l'algoritmo Ungherese [6], elaborato da Kuhn nel 1955, ed è un algoritmo di tipo Primale-Duale. Tale algoritmo sfrutta la teoria della dualità ed in particolare le condizioni di scarto complementare, è inizializzato con una soluzione duale ammissibile ed una soluzione primale che rispetta le condizioni di scarto complementare, alterna iterazioni primali e duali allo scopo di ridurre monotonicamente l'inammissibilità primale fino ad azzerarla. Nell'iterazione primale si mantiene fissa la soluzione duale determinando la soluzione primale che minimizza l'inammissibilità soddisfacendo le condizioni di scarto complementare. Nell'iterazione duale invece si mantiene fissa la soluzione primale e si modifica la soluzione duale mantenendo verificate le condizioni di scarto complementare. L'algoritmo Ungherese ha una complessità inferiore rispetto all'algoritmo di Ford-Fulkerson apportando un vantaggio in termini di riduzione del tempo di calcolo, quindi se il numero di squadre del problema affrontato è multiplo del numero di turni è da preferire l'utilizzo dell'algoritmo Ungherese, altrimenti si deve utilizzare l'algoritmo di Ford-Fulkerson. Al fine di valutare le prestazioni dell'algoritmo risolutivo del sottoproblema 2 ho effettuato alcune prove confrontando di volta in volta i risultati con quelli ottenuti risolvendo il problema mediante l'algoritmo del simplesso. Le prove hanno dato tutte esito favorevole all'algoritmo Ungherese, come si può notare dal grafico in figura 3.4, ottenendo un risparmio di tempo di calcolo, i dati sono osservabili nella tabella in figura 3.7. Date le elevate dimensioni dei problemi utilizzati per le prove qui riportate, ho generato casualmente le istanze dei problemi.

Dimensioni del problema					
	Luoghi	Sedi	Turni	Simpleso	Ungherese
1	500	300	50	7 sec	0,047 sec
2	500	500	100	49 sec	0,109 sec
3	600	600	100	71 sec	0,219 sec
4	600	600	200	195 sec	0,203 sec
5	1000	600	200	199 sec	1,14 sec
6	800	800	200	363 sec	7,687 sec

Figura 3.7: Tebella di confronto dei risultati del sottoproblema 2

Analizzando i dati presenti nella tabella della comparazione è possibile vedere come l'algoritmo del simpleso sia nettamente inefficiente e come l'algoritmo Ungherese sia in grado di risolvere in modo ottimo il problema ottenendo tempi di calcolo molto inferiori, quindi è da preferire l'utilizzo dell'algoritmo Ungherese.

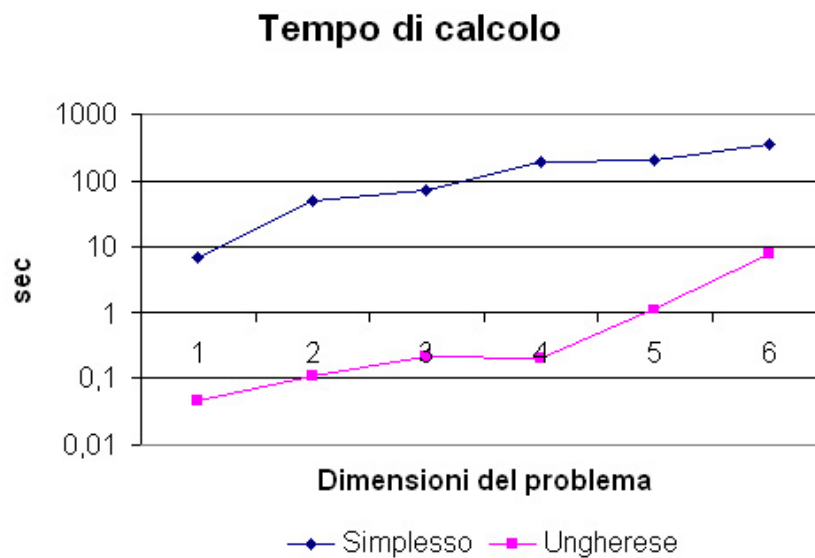


Figura 3.8: Grafico dei tempi di calcolo del sottoproblema 2

L'analisi effettuata trova riscontro nel grafico ricavato dalla tabella di comparazione, infatti si nota che le due linee hanno una pendenza simile ma hanno un offset verticale, ciò dimostra il fatto che i due algoritmi si comportano in modo simile con il crescere delle dimensioni del problema, ma il tempo di calcolo dell'algoritmo del semplice è nettamente superiore rispetto all'algoritmo Ungherese. Per cui l'algoritmo che ho deciso di adottare all'interno dell'algoritmo del sottogradient è l'algoritmo Ungherese, in caso che il numero di squadre sia multiplo del numero di turni, e l'algoritmo di Ford-Fulkerson quando i due valori non siano multipli.

### **3.5 Euristica iniziale**

L'euristica iniziale serve per ottenere una soluzione ammissibile del problema originario da poter utilizzare all'interno dell'algoritmo del sottogradient, tale soluzione deve rispettare ogni vincolo del problema. In questo caso una soluzione iniziale è abbastanza facile da ricavare, infatti è sufficiente distribuire in modo equo le squadre all'interno dei vari turni e successivamente per ogni turno assegnare ogni luogo ad una sola squadra che è di servizio in quel turno, scegliendo ovviamente la squadra avente la distanza minima dal luogo.

$N$  = vettore del numero massimo di squadre attive in ogni turno

$d_{max} := 0$

$j := 0$

**for**  $k := 1$  **to**  $K$

**for**  $n := 1$  **to**  $N_k$

$y_{jk} := 1$

$j++$

**for**  $i := 1$  **to**  $I$

$d_{min} := +\infty$

$j_{min} := 0$

**for**  $j_1 := 1$  **to**  $J$

**if**  $(y_{j_1 k} = 1) \& (d_{min} > d_{ij_1})$  **then**

$d_{min} := d_{ij_1}$

$j_{min} := j_1$

$x_{ij_{min} k} := 1$

**if**  $(d_{max} < d_{ij_{min}})$   $d_{max} := d_{ij_{min}}$

Figura 3.9: Pseudo-codice dell'euristica iniziale

L'euristica iniziale produce come risultato il valore della distanza massima  $d_{max}$  ed il relativo assegnamento scelto, in tal modo si ha un Upper Bound iniziale da utilizzare come prima pietra di paragone per le soluzioni ottenute nell'algoritmo del sottogradiente. Dalle prove effettuate i valori

delle euristiche sono risultati abbastanza soddisfacenti perché permettono di ricavare più velocemente la soluzione ottima, risparmiando iterazioni sia nell'algoritmo del sottogradiente che nel Branch-and-Bound.

### 3.6 Euristica Lagrangeana

L'euristica lagrangeana ha lo scopo di rendere ammissibile la soluzione del rilassamento lagrangeano per il problema originario, per far ciò è necessario che la soluzione rispetti tutti i vincoli rilassati, ovvero il vincolo  $x_{ijk} \leq y_{jk} \quad \forall i = 1, \dots, I \quad \forall j = 1, \dots, J \quad \forall k = 1, \dots, K$  che serve per garantire che l'assegnamento di un luogo ad una squadra in un determinato turno avvenga solo se la squadra è di servizio in quello specifico turno. Risolvendo all'ottimo sia il sottoproblema 1 che il sottoproblema 2 è di norma che la soluzione trovata non rispetti il vincolo rilassato, cioè è possibile che un luogo sia assegnato ad una squadra in un turno mentre quella squadra non sia di servizio in quel turno. Per rendere tale soluzione ammissibile per il problema originario è necessario assegnare nuovamente per ogni turno i luoghi alle squadre che sono di servizio in quel turno, la scelta di quale squadra è dedicata a sorvegliare un luogo si basa sulla distanza tra il luogo e la squadra, scegliendo ovviamente quella minima. In tal modo l'euristica lagrangeana mantiene inalterato l'assegnamento scelto per le variabili  $y_{jk}$  mentre elabora un nuovo assegnamento per le variabili  $x_{ijk}$ .

$d_{max} := 0$

**for**  $i := 1$  **to**  $I$

**for**  $k := 1$  **to**  $K$

$j^* := \underset{j|y_{jk} = 1}{argmin} \left\{ d_{ij} \right\}$

$x_{ij^*k} := 1$

**if**  $(d_{max} < d_{ij^*})$   $d_{max} := d_{ij^*}$

Figura 3.10: Pseudo-codice dell'euristica lagrangeana

L'euristica lagrangeana fornisce come risultato il valore della  $d_{max}$ , ovvero la massima distanza tra tutte le coppie luogo-squadra scelte, tale valore è lo strumento di confronto fra soluzioni in modo da stabilire quale sia la migliore, ovvero quella avente il valore inferiore. L'euristica lagrangeana è un algoritmo piuttosto semplice come complessità che nel caso peggiore è  $O(I \cdot J \cdot K)$ . Tale complessità risulta essere molto valida, permettendone così di ottenere soluzioni ammissibile per il problema originario in un tempo ridotto. Il vantaggio dell'utilizzo dell'euristica lagrangeana si riscontra nel tempo di calcolo, infatti ne permette un notevole risparmio.



### 3.7 Fissaggio di variabili

Il fissaggio di variabili è la procedura dedicata alla riduzione delle dimensioni del problema, ciò serve per velocizzare l'esecuzione dell'algoritmo stesso avendo meno variabili da utilizzare. Per valutare la possibilità di fissare il valore di una variabile occorre valutare il suo costo ridotto, infatti analizzando il costo ridotto della variabile, il suo valore attuale, il lower bound corrente e la miglior soluzione trovata è possibile decidere se fissare il valore di una determinata variabile. Il costo ridotto rappresenta il costo che deve essere pagato per far sì che quella variabile entri in base, ovvero entri a far parte della soluzione. Analizzando le formulazioni di entrambe i sottoproblemi è possibile effettuare il fissaggio di variabili esclusivamente nei seguenti casi:

- se  $d_{ij} > Z_{UB} \forall i = 1, \dots, I \quad \forall j = 1, \dots, J$  allora  $x_{ijk} = 0 \forall k = 1, \dots, K$ , poiché non è conveniente scegliere un assegnamento che abbia una distanza superiore a quella migliore finora trovata
- se  $y_{jk} = 0$  e  $Z_{LB} - \bar{c}_{y_{jk}} > Z_{UB}$  allora  $y_{jk}$  può essere posta a 0.

Il costo ridotto delle variabili  $y_{jk}$  è noto ed è fornito gratuitamente dall'algoritmo dell'Ungherese. Fissare i valori delle le variabili  $x_{ijk}$  non ha effetti sulle altre variabili perché esse sono già state scartate dalla soluzione a causa della distanza associata che risulta essere superiore alla miglior soluzione trovata. Imporre il valore delle variabili  $y_{jk}$  comporta la necessità di effettuare i controlli riguardanti la distribuzione delle squadre nei turni, in particolare controllando se in quel turno non si abbia proibito il numero massimo di squadra possibili, in tal modo è necessario rendere attive quelle rimanenti. Lo pseudo-codice che ne deriva è riportato in figura 3.11.

```

for  $i := 1$  to  $I$ 
  for  $j := 1$  to  $J$ 
    for  $k := 1$  to  $K$ 
      if ( $x_{ijk}$  non è fissata )
        if ( $x_{ijk} = 0$ )&( $d_{ij} > Z_{UB}$ ) then
          fisso  $x_{ijk} := 0$ 
for  $j := 1$  to  $J$ 
  for  $k := 1$  to  $K$ 
    if ( $y_{jk}$  non è fissata )
       $cr := 0$ 
      for  $i := 1$  to  $I$ 
         $cr := cr - \lambda_{ijk}$ 
       $cr := cr - u_j$ 
       $cr := cr - vk_k$ 
      if ( $y_{jk} = 0$ )&( $(Z_{LB} + cr) > Z_{UB}$ ) then
        fisso  $y_{jk} = 0$ 
      if ( $j$  ha un solo turno disponibile ) then
        assegno  $j$  al turno  $k$  libero
      if ( proibito il numero massimo di squadre nel turno  $k$ ) then
        impongo attive le restanti squadre
      for  $i := 1$  to  $I$ 
        fisso  $x_{ijk} = 0$ 

```

Figura 3.11: Pseudo-codice della procedura di fissaggio di variabili

Dalle prove sperimentali effettuate ho notato che fissare le variabili  $x_{ijk}$  è abbastanza utile e permette di ridurre il tempo di calcolo complessivo dell'algoritmo, mentre per quanto riguarda il fissare variabili  $y_{jk}$  ho ottenuto dei risultati poco convincenti.

### 3.8 Risultati sperimentali al nodo radice

Terminata la creazione dell'algoritmo del sottogradiente, ho eseguito una serie di prove al fine di valutare le prestazioni che l'algoritmo è in grado di raggiungere, a questo scopo mi sono tornati utili i risultati ottenuti con l'algoritmo creato mediante la libreria GLPK. La valutazione delle prestazioni si basa sull'analisi del tempo di calcolo e delle dimensioni del problema; per eseguire questi test ho utilizzato delle istanze di problemi presenti nella OR Library, adattandole opportunamente per l'algoritmo. Ho sottoposto all'algoritmo inizialmente problemi di modeste dimensioni, incrementandole di volta in volta di piccoli passi, in modo da conoscere il limite massimo di dimensioni e di poter vedere l'evoluzione del tempo di calcolo.

File	Luoghi	Sedi	Turni	Euristica	Soluzione trovata	Durata	Lower Bound massimo	Gap %	Numero Iterazioni
gap15-5-2.txt	15	5	2	25	20	0,125	19	5,000	674
mdmkp20-6-2.txt	20	6	2	768	612	0,094	482	21,242	649
csp20-6-3.txt	20	6	3	567	567	0,172	558	1,587	626
csp20-9-6.txt	20	9	6	1543	1362	1,609	1327	2,570	988
assign30-9-5.txt	30	9	5	96	93	1,953	65	30,108	855
gap35-7-4.txt	35	7	4	36	27	1,172	23	14,815	806
pmed37-10-3.txt	37	10	3	93	93	0,718	78	16,129	517
mdmkp40-6-2.txt	40	6	2	768	670	0,250	522	22,090	630
pmed40-8-4.txt	40	8	4	600	105	0,875	87	17,143	620
csp41-9-5.txt	41	9	5	1518	1447	1,813	1392	3,801	624
pmed50-8-4.txt	50	8	4	600	131	1,375	111	15,267	664
assign50-12-7.txt	50	12	7	100	97	10,469	57	41,237	994
pmed60-8-4.txt	60	8	4	600	158	1,735	124	21,519	605
mdmkp100-10-3.txt	100	10	3	866	712	3,718	508	28,652	704
assign150-9-3.txt	150	9	3	92	77	8,672	56	27,273	643
mdmkp200-10-3.txt	200	10	3	869	744	13,140	582	21,774	680
mdmkp200-20-3.txt	200	20	3	657	475	15,875	342	28,000	630
mdmkp300-30-3.txt	300	30	3	583	393	41,391	289	26,463	738
mdmkp300-60-3.txt	300	60	3	58284	225	35,844	146	35,111	429
assign400-20-5.txt	400	20	5	92	76	379,219	41	46,053	1001
p1000-10-3.txt	1000	10	3	970	844	380,563	650	22,986	711

Figura 3.12: Tabella delle prestazioni del sottogradiente

Dai dati presenti nella tabella è possibile vedere come il tempo di calcolo dell'algoritmo del sottogradiente evolve incrementando le dimensioni del problema. Fino a dimensioni dell'ordine di 150 luoghi, 9 sedi e 3 turni il tempo di calcolo raggiunge al massimo i 10 sec, superato questo limite il tempo di calcolo inizia una crescita più considerevole ottenendo tempi di calcolo dell'ordine delle decine di secondi. Il vero limite dell'algoritmo si mostra in corrispondenza dell'ultimo problema, quello avente le dimensioni maggiori, infatti il tempo di calcolo è di ben 380 sec che è un tempo abbastanza elevato, incrementando ulteriormente le dimensioni del problema occorrerebbe

un tempo notevolmente superiore per la sua risoluzione.

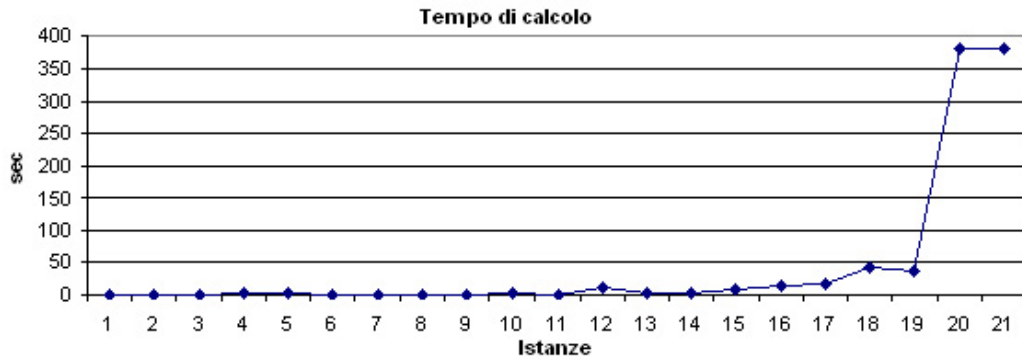


Figura 3.13: Grafico dei tempi di calcolo del sottogradiente

Un elemento di valutazione della soluzione fornita dall'algoritmo del sottogradiente è il Gap, ovvero il possibile errore che può esistere nella soluzione, questo valore è calcolato come differenza tra il massimo lower bound trovato e la miglior soluzione, più piccolo è il valore e migliore è la soluzione fornita. Come si può notare dai valori della tabella, il Gap rimane abbastanza alto, infatti il valor medio è circa 30%, tuttavia vi sono problemi in cui si ha ottenuto un Gap molto piccolo, e ciò significa che la soluzione fornita è molto probabile che sia quella ottima.

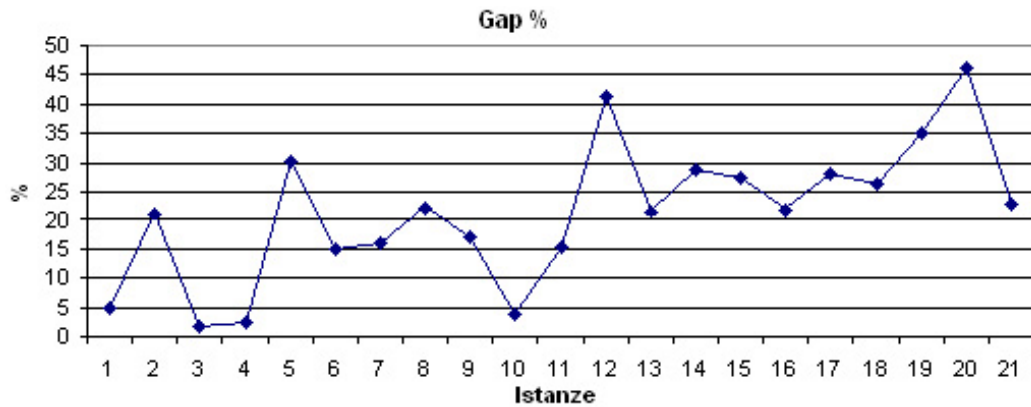


Figura 3.14: Grafico del gap del sottogradiente

### 3.9 Branch-and-Bound

L'algoritmo Branch-and-Bound[9] ha lo scopo di trovare la soluzione ottima e di garantirne l'ottimalità, l'algoritmo è composto da due parti fondamentali, la parte di Branching e la parte di Bounding. Nella fase di branching[1] viene fissato il valore delle variabili appartenenti al vincolo scelto per tale operazione, tale scelta ricade sul vincolo rilassato violato avente la distanza minima. Il vincolo rilassato violato si presenta con la forma  $x_{ijk} = 1$  e  $y_{jk} = 0$ , per tanto è possibile fare branching sulle variabili  $y_{jk}$ , modificando opportunamente le variabili  $x_{ijk}$ , infatti è possibile imporre che in un sottoproblema la variabile  $y_{jk}$  assuma il valore 1 mentre nell'altro sottoproblema può assumere il valore 0, ottenendo così due sottoinsiemi disgiunti delle soluzioni del problema originario ciascuno assegnato ad un sottoproblema. Le operazioni da compiere nella fase di branching sono regolate dalla strategia di branching. La fase successiva è la fase di bounding[1], ovvero la valutazione dei sottoproblemi generati, tale valutazione viene effettuata sottoponendo all'algoritmo del sottogradiente il sottoproblema ottenendo così il relativo lower bound. Tale valore è confrontato con il valore della miglior soluzione trovata al fine

di decidere se è necessario continuare l'esplorazione di quel sottoproblema oppure è possibile scartarlo. I sottoproblemi che necessitano di continuare nell'esplorazione devono essere memorizzati e quindi è necessario utilizzare una struttura dati che permetta di inserire ed estrarre i vari sottoproblemi in base ad un criterio, il criterio definisce la strategia di esplorazione. Infatti se si decide di utilizzare come criterio il lower bound si mantengono i sottoproblemi ordinati per lower bound crescente e si estrae sempre il sottoproblema con lower bound minore, realizzando così una strategia di tipo "Best-First-Search". L'algoritmo Branch-and-Bound può non garantire l'ottimalità della soluzione terminando a causa della scadenza del time out. Il motivo dell'utilizzo di un time out è che gli algoritmo di tipo Branch-and-Bound trovano rapidamente la soluzione ottima ma impiegano tanto tempo per verificarne l'ottimalità, per ciò per problemi di grandi dimensioni è utile imporre un time out che blocchi l'algoritmo. Nel caso cui la soluzione non abbia la garanzia di ottimalità l'algoritmo fornisce alcuni strumenti che possono indicare la qualità della soluzione, questi strumenti sono:

- **Gap** è la differenza tra la soluzione ed il minimo lower bound al termine dell'algoritmo e rappresenta il possibile errore che la soluzione può avere, tanto più è piccolo tanto più la soluzione si avvicina a quella ottima
- **Lower bound massimo** è il massimo lower bound incontrato durante l'esecuzione.
- **Numero di nodi non esplorati** è il numero di problemi di cui non sono stati generati i sottoproblemi.

Grazie a questi tre parametri è possibile avere un'informazione più completa sulla soluzione trovata. In conclusione l'algoritmo Branch-and-Bound

incorpora al suo intero l'algoritmo del sottogradiente per la valutazione dei sottoproblemi, il quale incorpora a sua volta l'algoritmo per la risoluzione del sottoproblema 1 e del sottoproblema 2. Lo pseudo-codice dell'algoritmo Branch-and-Bound è rappresentato in figura 3.15.

*inizializza(radice)* inizializzazione del nodo radice

*lower\_bound(radice)* Valutazione del nodo radice

*inserisci(radice)* inserimento del nodo radice nell'albero

**while** ( true )

**if** ( albero vuoto ) ottimo = true; break;

**if** ( lower bound massimo == miglior soluzione ) ottimo = true; break;

**if** ( time out scaduto ) ottimo = false; break;

*estrai(nodo)* estrazione del nodo dall'albero

*genera\_figli(nodo)* generazione dei figli del nodo estratto

*elimina(nodo)* eliminazione del nodo estratto

Figura 3.15: Pseudo-codice dell'algoritmo Branch-and-bound

### 3.9.1 Strategia di branching

La strategia di Branching detta le regole per la generazione dei sottoproblemi partendo dal problema padre, per partizionare l'insieme delle soluzioni del problema padre in sottoinsiemi disgiunti è necessario agire sui valori delle variabili del problema, imponendo che assumano determinati valori. Come già descritto precedentemente l'operazione di branching verrà effettuato esclusivamente sulle variabili  $y_{jk}$ , tale operazione necessita che venga scelto un vincolo del problema per ottenere la variabile di cui si intende fissare il va-



lore. Successivamente si esegue il branching sulla variabile del vincolo scelto,  $y_{jk}$ , in base alla seguenti regole:

- nel primo sottoproblema la variabile assume il valore 1
- nel secondo sottoproblema assume il valore 0

Prima di fissare il valore della variabile vengono eseguiti i controllo che ne determinano la sua possibilità, tali controlli dipendono dal sottoproblema infatti:

- per il primo sottoproblema è possibile fissare il valore di  $y_{jk}$  a 1 se nel turno  $k$  non sono ancora state attivate il numero massimo di squadre
- per il secondo sottoproblema è possibile fissare il valore a 0 se esistono altri turni in cui la sede  $j$  può essere attivata e se esistono altre squadre che possono essere attivate nel turno  $k$  al posto della squadra  $j$

Nel caso cui non sia possibile fissare il valore della variabile in un sottoproblema, tale sottoproblema risulta essere inammissibile per cui verrà eliminato. Fissare il valore di una variabile  $y_{jk}$  ha delle conseguenze sulle altre variabili del problema, infatti quando:

- si fissa  $y_{jk} = 0$ , è possibile imporre che nessun luogo sia assegnato alla squadra  $j$  nel turno  $k$  dato che essa non è di servizio, per cui  $\forall i = 1, \dots, I \quad x_{ijk} = 0$ , è possibile inoltre controllare se la squadra  $j$  ha un unico turno in cui può essere attivata controllare se si abbia raggiunto il numero massimo di squadre proibite nel turno  $k$ , imponendo di servizio le rimanenti squadre velocizzando così l'algoritmo
- si impone  $y_{jk} = 1$  è possibile controllare se si ha raggiunto il numero massimo di squadre attive nel turno  $k$  proibendo le rimanenti, inoltre

è possibile proibire la squadra  $j$  in tutti i turni eccetto il turno  $k$ ; dato che il vincolo scelto per l'operazione di branching ha l'assegnamento  $x_{ijk} = 1$  con distanza  $d_{ij}$  minima è possibile fissare  $x_{ijk} = 1$

La procedura di branching è divisibile in due parti, nella prima si ricerca il vincolo da utilizzare e la seconda parte serve per generare i sottoproblemi, lo pseudocodice che ne deriva è il seguente:

$i_v$

$j_v$  variabili per mantenere il riferimento al vincolo scelto

$k_v$

Generazione del sottoproblema 1 con  $y_{j_vk_v} = 1$

**if** ( fissare la squadra  $j_v$  nel turno  $k_v$  )

$y_{j_vk_v} = 1$

**if** ( proibite tutte le sedi rimanenti )

**for**  $j := 1$  **to**  $J$

**if** ( $y_{jk}$  non 'e fissata)  $y_{jk} = 0$

**for**  $k := 1$  **to**  $K$

**if** ( $k \notin k_v$ )  $y_{jvk} = 0$  azzeramento dell'assegnamento della squadra negli altri turni

$x_{ij_vj_vk_v} = 1$  assegnamento del luogo alla sede fissata

**for**  $kj := 1$  **to**  $J$

**if** ( $j \notin j_v$ )  $x_{i_vj_vk_v} = 0$  proibizione dell'assegnamento del luogo ad un'altra sede

Generazione del sottoproblema 2 con  $y_{j_vk_v} = 0$

**if** ( proibire la squadra  $j_v$  nel turno  $k_v$  )

$y_{j_vk_v} = 0$  proibizione dell'assegnamento della sede

**if** ( esiste un solo turno per la squadra  $j_v$  )

**for**  $k := 1$  **to**  $K$

**if** ( $y_{jvk}$  non è fissata )

```

 $y_{jvk} = 1$  assegnamento della squadra al turno
if ( massimo numero squadre proibite nel turno  $k_v$  )
    for  $j := 1$  to  $J$ 
        if ( $y_{jk_v}$  non è fissata )
             $y_{jk_v} = 1$ 
    for  $i := 1$  to  $I$ 
         $x_{xjvk_v} = 0$  proibizione di assegnare i luogo alla squadra non attiva

```

### 3.9.2 Strategia di esplorazione dell'albero

Dato che non è possibile esaminare contemporaneamente i diversi “figli” di ogni problema “padre” per cui è necessario stabilire un ordinamento dei sottoproblemi “aperti”, definendo così una politica di esplorazione dell'albero di decisione. Esistono vari tipi di strategie di esplorazione, la più comune è la politica di tipo “Best-First-Search”, nella quale si esplora per primo il nodo aperto più promettente, ovvero quello con il lower bound inferiore, infatti l'ordinamento viene effettuato in base al valore del lower bound. Nella “Depth-First-Search”, ricerca in profondità, si esplora un ramo ancora prima di generare quelli successivi, questa strategia si presta molto bene ad una realizzazione ricorsiva, non richiede la memorizzazione dell'albero e fornisce rapidamente un bound primale. Per realizzare tale strategia è necessario ordinare i nodi “aperti” per il numero progressivo di generazione del nodo, estraendo sempre il nodo con il numero maggiore, ovvero l'ultimo nodo generato. Possono essere implementate strategie ibride che modificano a run-time la loro politica per esempio procedendo inizialmente con una strategia di tipo DFS e successivamente passando alla BFS. Allo scopo di ricercare una

strategia migliore ho implementato le seguenti tre strategie:

- “Best-First-Search” ottenuta ordinando per lower bound crescente i nodi aperti ed estraendo sempre il nodo avente lower bound minore
- “Depth-First-Search” ottenuta ordinando per numero di generazione decrescente ed estraendo sempre il nodo con tale numero maggiore
- “Breadth-First-Search” è una strategia di ricerca in cui i nodi aperti sono ordinati in base al livello dell’albero di decisione a cui appartengono

Per verificare quale tra le strategie implementate sia la migliore ho compiuto diverse prove sperimentali sottoponendo all’algoritmo, implementato con le diverse strategie, il medesimo problema ed ho ottenuto i dati presenti nella tabella 3.16.

File	Luoghi	Sedi	Tumi	Best-First-Search				Breadth-First-Search				Depth-First-Search			
				Soluzione	Durata	Memoria	Gap %	Soluzione	Durata	Memoria	Gap %	Soluzione	Durata	Memoria	Gap %
mdmkp20-6-2.txt	20	6	2	612	0,250	5,258		612	0,375	5,227		612	0,406	5,801	
csp20-6-3.txt	20	6	3	567	0,250	6,598		567	0,390	6,582		567	0,390	6,582	
assign30-10-5.txt	30	10	5	88	28,656	94,418		88	836,000	50,395		88	885,969	75,395	
mdmkp40-6-2.txt	40	6	2	670	0,369	8,895		670	0,344	8,871		670	0,359	8,871	
pmed40-8-4.txt	40	8	4	105	0,390	21,895		105	196,375	35,133		105	179,625	48,395	
pmed50-8-4.txt	50	8	4	131	0,625	27,207		131	223,672	46,848		131	224,891	66,512	
pmed60-8-4.txt	60	8	4	158	0,657	32,520		158	398,625	55,910		158	403,438	79,324	
assign150-9-3.txt	150	9	3	77	135,375	303,107		77	488,125	101,363		77	481,734	133,590	
mdmkp300-30-3.txt	300	30	3	336	3619,906	14316,934	15,774	346	3607,000	1486,996	18,208	358	3625,985	2389,457	20,950
assign400-20-5.txt	400	20	5	74	3734,594	3090,016	40,541	75	3613,047	1755,918	42,867	74	3651,672	3325,137	41,892
mdmkp300-60-3.txt	300	60	3	191	3676,297	7496,004	25,131	180	3638,672	4418,070	20,000	197	3616,828	10573,109	26,904

Figura 3.16: Tabella di comparazione delle strategie di esplorazione

Analizzando i dati presenti nella tabella di confronto delle strategie è possibile notare come la strategia di tipo “Best-First-Search”, BFS, riesca a risolvere all’ottimo i problemi in un tempo notevolmente inferiore rispetto alle altre strategie, ciò lo si può notare anche dal grafico di comparazione dei tempi di calcolo. Tale differenza è possibile grazie al fatto che nella “Best-First-Search” è possibile interrompere l’algoritmo Branch-and-Bound quando si trova un valore di lower bound uguale al valore della miglior soluzione trovata, in tal modo è possibile scartare i nodi restanti perché si ha la garanzia dell’ottimalità della soluzione trovata. Se si adotta una strategia di tipo “Breadth-First-Search”, BHFS, o “Depth-First-Search”, DFS, non è possibile attuare una terminazione come quella descritta precedentemente perché si rischierebbe di scartare nodi contenenti la soluzione ottima, per cui è possibile terminare l’algoritmo Branch-and-Bound all’ottimo solo se non vi sono più nodi da esplorare, in tal modo il tempo di calcolo dell’algoritmo può incrementarsi notevolmente. Le considerazioni espresse trovano riscontro nel grafico dei tempi di calcolo in figura 3.17.

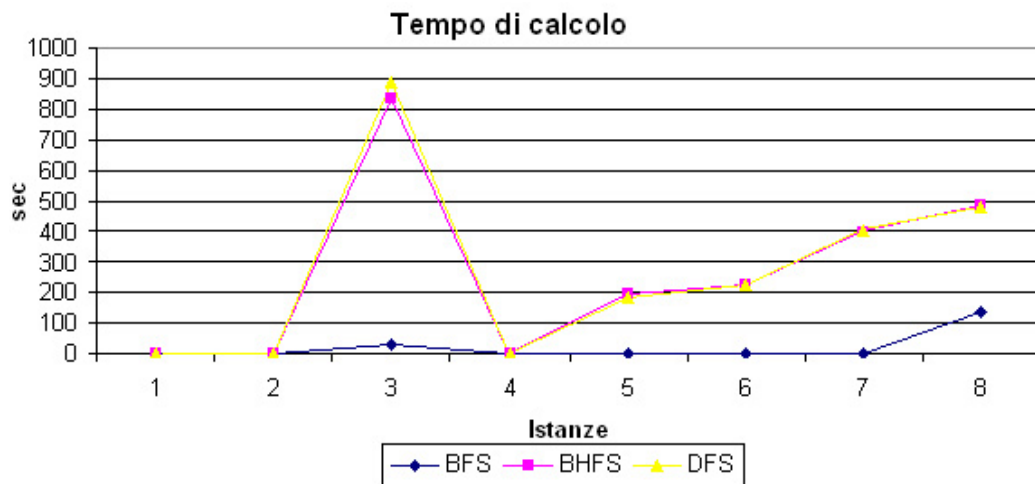


Figura 3.17: Grafico di comparazione dei tempi di calcolo delle strategie

Come si può notare dai dati presenti nella tabella e dal grafico dei tempi di calcolo le strategie di esplorazione BHFS e DFS sono pressoché simili, mentre la strategia BFS è notevolmente migliore, infatti fornisce tempi di calcolo nettamente inferiori al fronte di identiche soluzioni ottime, quindi sarebbe preferibile utilizzarla nell'algoritmo Branch-and-Bound. Altre differenze tra le varie strategie sono state rilevate quando ho sottoposto all'algoritmo problemi di elevate dimensioni che non sono stati risolti all'ottimo a causa della scadenza del time out, grazie a queste prove si è potuto analizzare la qualità delle soluzioni ottenute mediante le diverse strategie, i parametri utilizzati sono il valore della soluzione trovata e la percentuale di Gap.

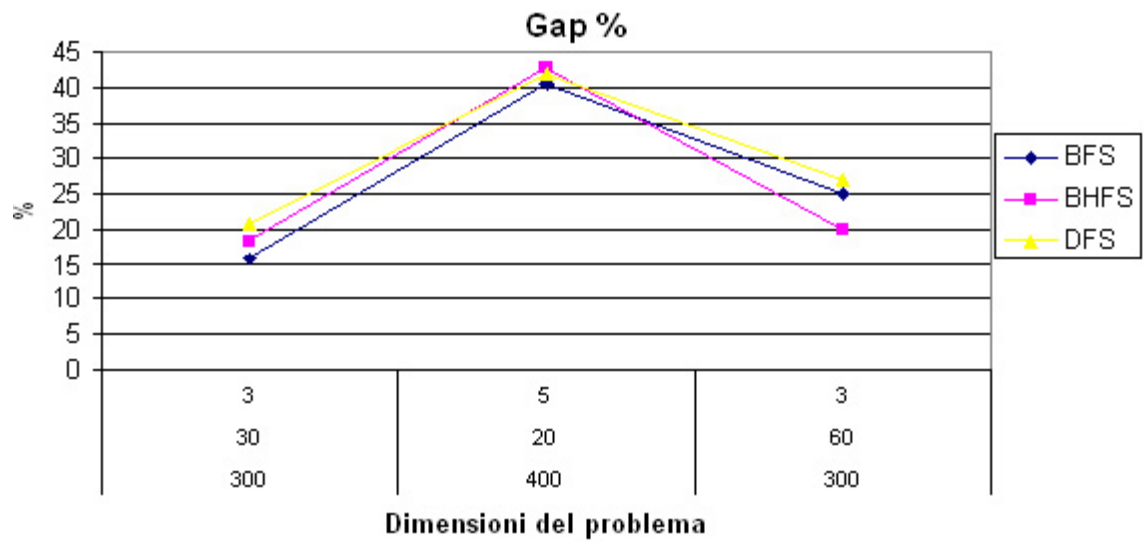


Figura 3.18: Grafico di comparazione delle percentuali di Gap

Il grafico delle percentuali di Gap mostra come la strategia di esplorazione BFS fornisce una percentuale di Gap inferiore rispetto alle altre strategie, ciò significa che le soluzioni ricavate mediante BFS contengono un possibile errore inferiore rispetto alle altre, non si garantisce che la soluzione sia migliore ma si garantisce che il possibile errore compiuto sia minore.



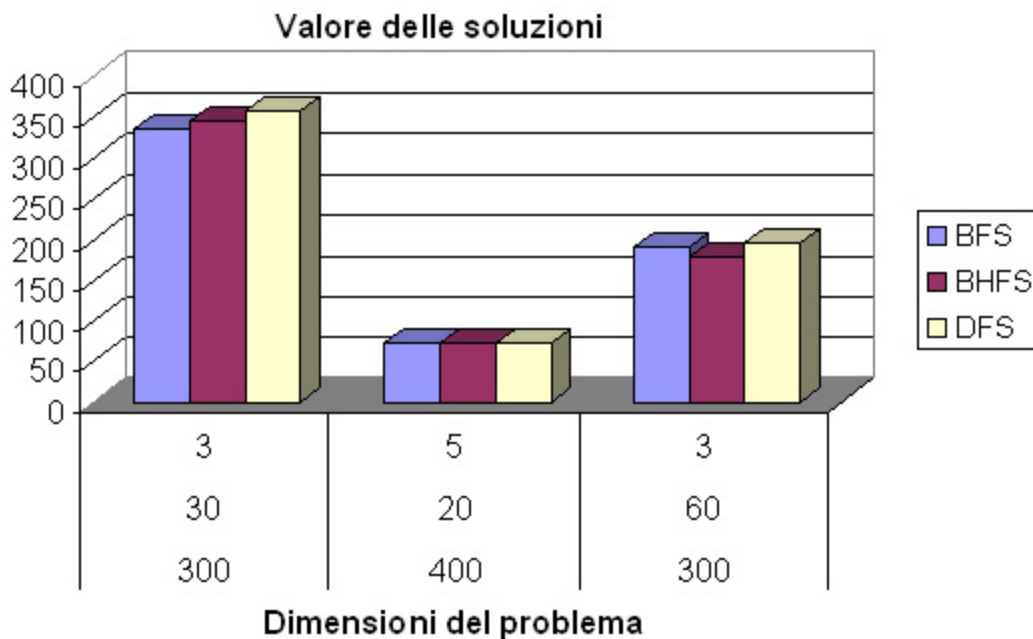


Figura 3.19: Grafico di comparazione delle soluzioni

Le soluzioni ottenute mediante le diverse strategie di esplorazione sono equivalenti, quindi non esiste una strategia che fornisce effettivamente sempre soluzioni migliori rispetto alle altre, per cui la scelta della strategia da adottare è stata basata sui criteri del tempo di calcolo e del Gap fornito. In conclusione ho scelto la strategia BFS perché fornisce dei tempi di calcolo notevolmente inferiori e dei Gap migliori, per implementare questo tipo di strategia è necessario ordinare i nodi aperti per valori crescenti di lower bound, tali nodi sono memorizzati in una struttura dati. Per memorizzare i nodi aperti era sufficiente una lista, ma la complessità delle operazioni di inserimento ordinato e di estrazione sono rispettivamente  $O(n)$  e  $O(1)$ , l'operazione di inserimento può richiedere un tempo elevato se il numero di nodi aperti è elevato, per velocizzare tale operazione ho adottato una struttura dati diversa, lo heap [4]. Grazie allo heap è possibile velocizzare l'operazione

di inserimento a discapito dell'operazione di estrazione, ma l'incremento della complessità di tale operazione è più che accettabile, infatti la sua complessità sale a  $O(\log n)$  ma la complessità dell'operazione di inserimento scende a  $O(\log n)$ , in tal modo si ha un notevole risparmio di tempo di calcolo.

### 3.10 Risultati Sperimentali

Allo scopo di valutare le prestazioni dell'algoritmo Branch-and-Bound ho effettuato numerose prove sperimentali basate su istanze di problemi della OR-Library opportunamente adattate per il problema in questione. Le istanze del problema hanno dimensioni diverse, differendo per numero di luoghi, sedi e turni oltre che per le distanze luoghi-sedi. Lo scopo di queste diversità è quello di valutare il comportamento dell'algoritmo in diverse situazioni e trovarne il limite prestazionale. Inizialmente ho sottoposto all'algoritmo istanze con dimensioni modeste e incrementando man mano tali dimensioni allo scopo di raggiungere il limite, i dati che ho ricavato sono riportati in figura 3.20.

File	Luoghi	Sedi	Turni	Soluzione	Nodo	Livello	Durata	Memoria	Gap %	Lower Bound massimo	Lower Bound minimo finale	Numero nodi generati	Massimo numero nodi aperti
gap15-5-2.txt	15	5	2	20	0	0	0,141	3,070		20		3	1
mdmkp20-6-2.txt	20	6	2	612	0	0	0,250	5,258		612		5	2
csp20-6-3.txt	20	6	3	567	0	0	0,250	6,598		567		3	1
csp20-9-6.txt	20	9	6	1362	0	0	2,672	18,418		1362		1	1
assign30-9-5.txt	30	9	5	93	0	0	530,578	421,947		93		285	142
assign30-10-5.txt	30	10	5	88	4	2	28,656	94,418		88		47	23
gap35-7-4.txt	35	7	4	27	0	0	0,563	16,875		27		1	1
pmed37-10-3.txt	37	10	3	93	0	0	0,594	19,426		93		1	1
mdmkp40-6-2.txt	40	6	2	670	0	0	0,359	8,895		670		3	1
pmed40-8-4.txt	40	8	4	105	0	0	0,390	21,895		105		1	1
csp41-9-5.txt	41	9	5	1447	0	0	885,703	912,172		1447		467	233
pmed50-8-4.txt	50	8	4	131	0	0	0,625	27,207		131		1	1
pmed60-8-4.txt	60	8	4	158	0	0	0,657	32,520		158		1	1
mdmkp100-10-3.txt	100	10	3	698	0	0	110,016	159,918		698		39	19
assign150-9-3.txt	150	9	3	77	0	0	135,375	303,107		77		61	30
mdmkp200-10-3.txt	200	10	3	744	3	2	223,266	363,699		744		47	23
mdmkp200-20-3.txt	200	20	3	446	19	3	3662,578	2219,055	20,404	420	355	171	86
mdmkp300-30-3.txt	300	30	3	336	77	10	3619,906	14316,934	15,774	304	283	523	262
assign400-20-5.txt	400	20	5	74	41	6	3734,594	3090,016	40,541	52	44	63	32
p1000-10-3.txt	1000	10	3	844	13	4	3609,938	1918,715	16,588	762	704	49	25
mdmkp300-60-3.txt	300	60	3	191	105	17	3676,297	7496,004	25,131	180	143	125	63

Figura 3.20: Tabella dei risultati dell'algoritmo Branch-and-bound

I risultati ottenuti sono molto soddisfacenti, infatti l'algoritmo riesce a risolvere in modo ottimo problemi di dimensioni dell'ordine di 200 luoghi, 10 sedi e 3 turni in tempi più che ragionevoli, incrementando le dimensioni l'algoritmo tende a fornire delle soluzioni senza la garanzia di ottimalità a causa della scadenza del time-out. Il time-out serve per arrestare l'algoritmo Branch-and-Bound quando la sua esecuzione si protrae per un tempo elevato, così facendo la garanzia di ottimalità della soluzione sfuma, ma l'algoritmo fornisce alcuni parametri come il lower bound massimo ed il Gap utili per valutare la qualità della soluzione fornita. Infatti è noto che gli algoritmi di tipo Branch-and-Bound trovano velocemente la soluzione "ottima" ma impiegano un tempo notevole per garantirne l'ottimalità, per cui è possibile terminare l'algoritmo dopo un periodo di tempo trascurando così l'esplorazione dei nodi ancora aperti. Analizzando i valori di Gap ottenuti con problemi di elevate dimensioni si nota che il possibile errore insito nella soluzione è abbastanza

piccolo garantendo così la validità della soluzione trovata e che il suo valore non differisca di molto dal valore della possibile soluzione ottima.

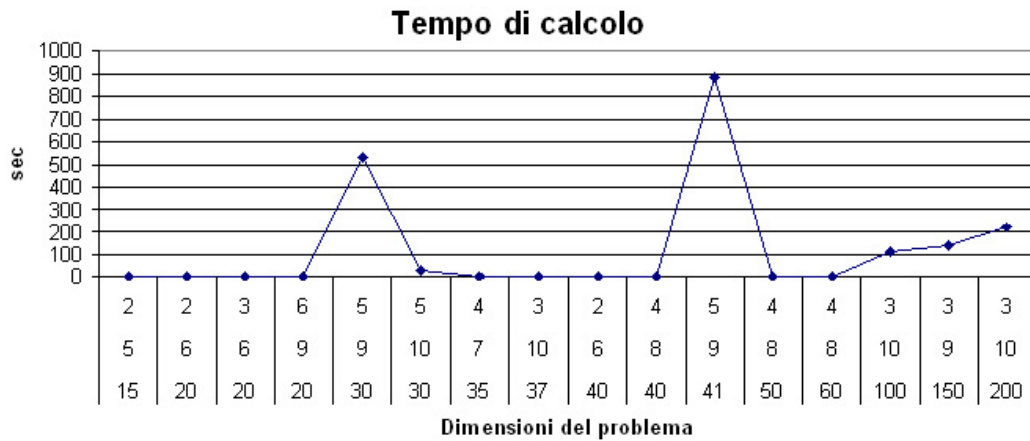


Figura 3.21: Grafico dei tempi di calcolo

Dal grafico dei tempi di calcolo delle istanze di problemi risolte in modo ottimo si nota come la crescita del tempo di calcolo sia lineare, ciò dimostra che incrementando notevolmente le dimensioni del problema da risolvere il tempo di calcolo subisce una crescita inferiore rispetto alla crescita subita dalle dimensioni del problema. Il tempo massimo riscontrato nella risoluzione ottima dei problemi è inferiore a 900 sec ed è un valore accettabile, tuttavia tale valore rappresenta un picco nel grafico dei tempi di calcolo, mentre il valor medio è di 120 sec che risulta essere molto valido. L'algoritmo permette di risolvere in modo ottimo un problema avente dimensioni dell'ordine di 200 luoghi, 10 sedi e 3 turni. Incrementando ulteriormente le dimensioni l'algoritmo non riesce a fornire la garanzia di ottimalità della soluzione fornita per cui il tempo di calcolo risulta essere uguale alla durata del time-out.

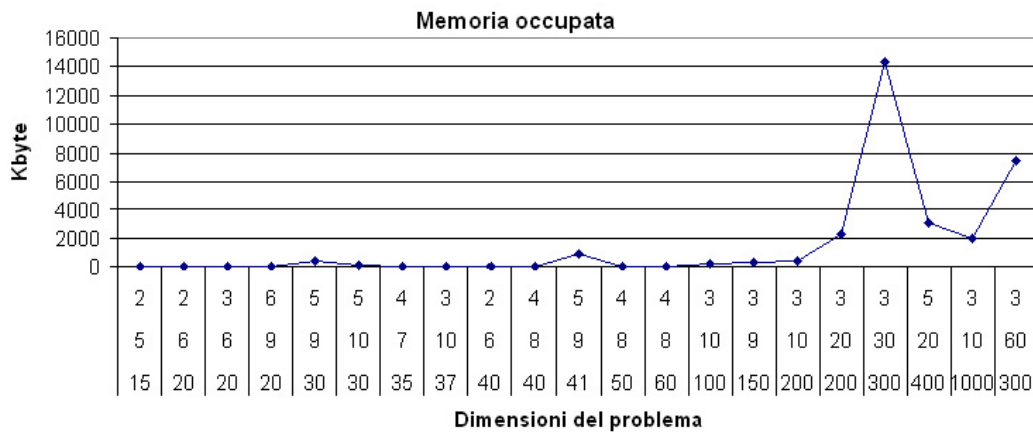


Figura 3.22: Grafico dell'occupazione di memoria

Un parametro di valutazione delle prestazioni di un algoritmo è la memoria che esso occupa durante la sua esecuzione. Dal grafico dell'occupazione di memoria è visibile come per le istanze di problema risolte all'ottimo la memoria richiesta per l'esecuzione dell'algoritmo rimanga notevolmente inferiore al valore di 1000Kbyte. Per istanze di elevate dimensioni di cui non si ha ottenuto una soluzione ottima la memoria occupata subisce un incremento notevole a causa del maggior numero di nodi mantenuti aperti, oltre che all'incremento del numero di variabili necessarie al problema. Il valore massimo raggiunto rimane comunque inferiore al valore di 15000Kbyte che risulta essere abbastanza basso e non costituisce un limite per l'esecuzione dell'algoritmo.

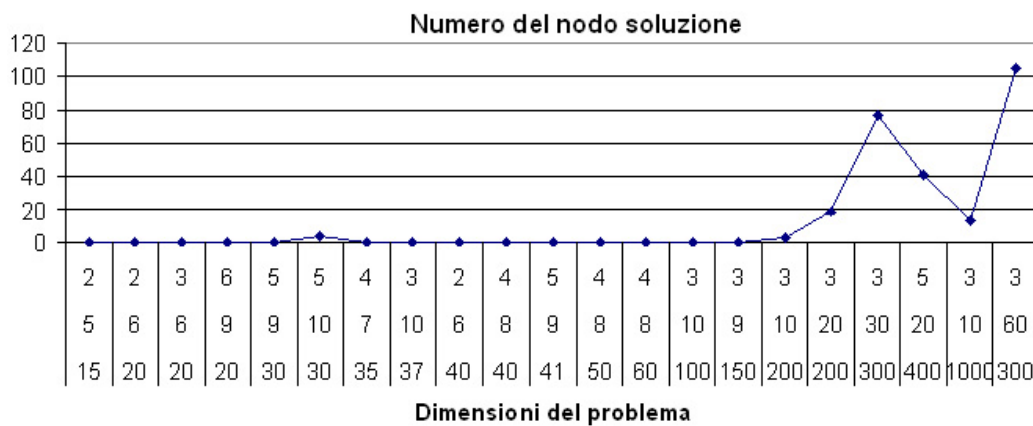


Figura 3.23: Grafico del numero del nodo della soluzione

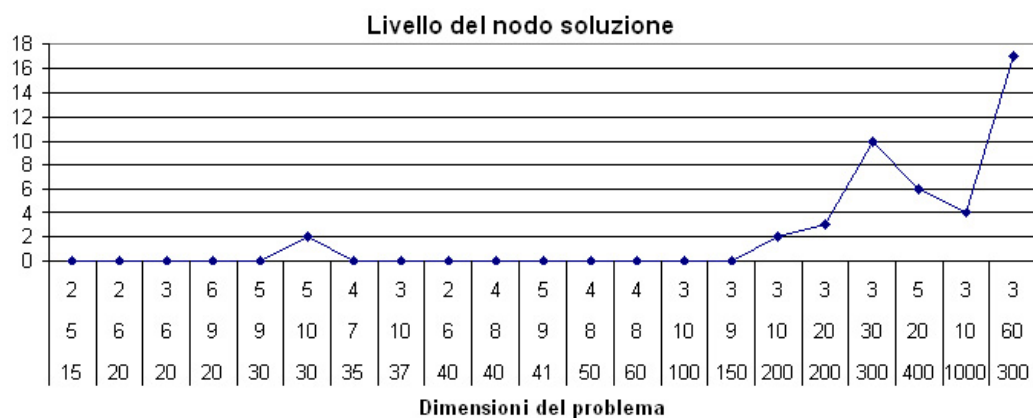


Figura 3.24: Grafico del livello del nodo della soluzione

Dai grafici illustranti il numero ed il livello del nodo in cui l'algoritmo ha trovato la soluzione vi è la conferma della teoria per la quale gli algoritmi di tipo Branch-and-Bound trovano rapidamente la soluzione, nei primi nodi esplorati, ed utilizzano il restante tempo per garantirne l'ottimalità. Infatti come si può notare nella maggior parte dei problemi affrontati la soluzione è stata ricavata addirittura dalla radice dell'albero di decisione, mentre nelle restanti istanze, quelle di cui l'algoritmo non è riuscito a calcolare la soluzione ottima, la soluzione migliore è stata trovata nei primi nodi esplorati

eccetto in un caso in cui la soluzione migliore è stata trovata quasi al termine dell'algoritmo. Il livello del nodo in cui è stata trovata la soluzione ci dà l'informazione di quanto si è dovuto scendere in profondità nell'albero di decisione per trovare tale soluzione, rappresentando il numero di iterazioni di branching a partire dalla radice dell'albero si ha dovuto effettuare per raggiungere la tale soluzione. Come si può notare anche il grafico riguardante il livello conferma la teoria sulla ricerca della soluzione ottima per algoritmi di tipo Branch-and-Bound.

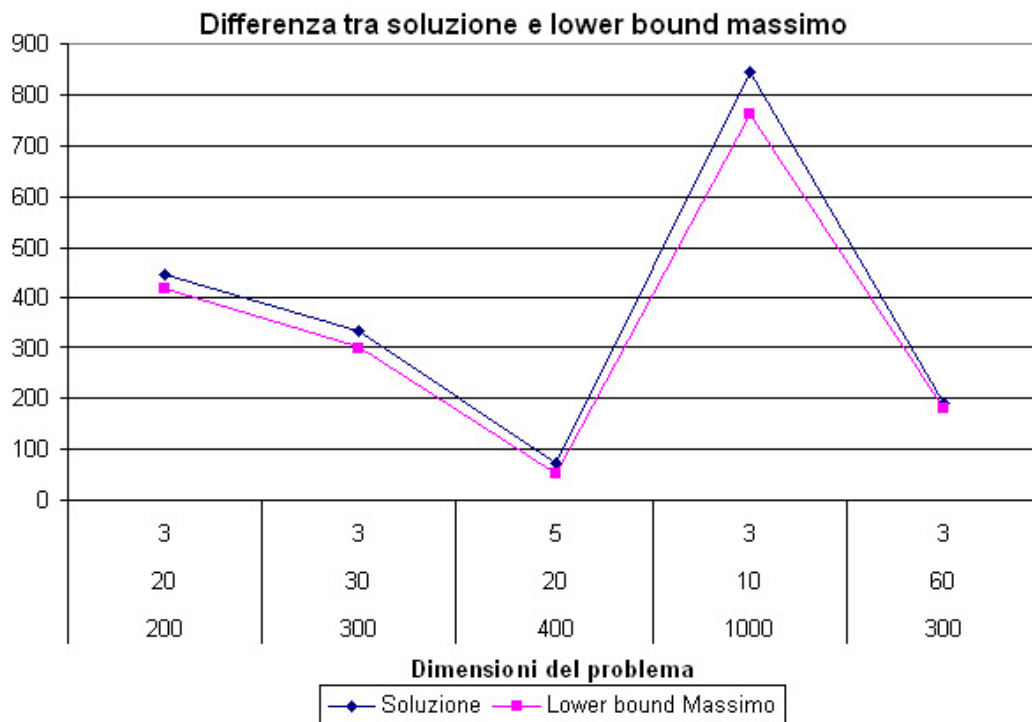


Figura 3.25: Grafico della soluzione e massimo lower bound

Nel caso in cui la soluzione fornita dall'algoritmo non abbia la garanzia di ottimalità è possibile verificare quanto l'algoritmo sia stato in grado di avvicinare la soluzione a quella ottima, infatti il massimo lower bound rappresenta il valore minimo che la possibile soluzione ottima può avere. Quindi tanto più

il valore del lower bound massimo sia vicino al valore della soluzione trovata tanto più la soluzione trovata ha probabilità di essere effettivamente quella ottima. Dal grafico della differenza tra soluzione e lower bound massimo è possibile vedere come l'algoritmo fornisce dei valori di lower bound massimi molto validi e vicini alla soluzione trovata, quindi la qualità della soluzione è elevata e la differenza dalla soluzione ottima è molto ridotta. Allo scopo di valutare la bontà della soluzione è possibile analizzare la percentuale di Gap, che rappresenta la differenza in percentuale tra la soluzione trovata ed il minimo lower bound appartenente ai nodi rimasti inesplorati al termine dell'algoritmo. Tali valori di Gap, riportati nella tabella dei risultati sperimentali, risultano essere anch'essi molto validi rafforzando quindi la qualità della soluzione fornita dall'algoritmo.

In conclusione l'algoritmo riesce ad ottenere soluzioni ottime con tempi di calcolo ridotti per istanze del problema con dimensioni dell'ordine di 200 luoghi, 10 sedi e 3 turni, incrementando notevolmente le dimensioni è possibile ottenere delle soluzioni molto valide ma che purtroppo non hanno la garanzia dell'ottimalità a causa del time-out imposto. Al fine di ottenere tale garanzia è possibile innalzare il valore del time-out concedendo quindi all'algoritmo più tempo per la sua esecuzione. Il limite dimensionale delle istanze da sottoporre all'algoritmo non è legato a limiti strutturali dell'implementazione dell'algoritmo ma è legato più al fattore del tempo di calcolo necessario per la risoluzione delle istanze, l'istanza di maggiori dimensioni che ho sottoposto all'algoritmo riguardava 300 luoghi, 60 sedi e 3 turni, tali valori possono essere ancora incrementati innalzando così il limite dimensionale dell'algoritmo.



## Capitolo 4

### Interfaccia

## 4.1 Aspetto generale

L'interfaccia di un algoritmo dedicato alla risoluzione del problema della dislocazione e della turnazione di squadre di pronto intervento deve offrire la possibilità di interagire con una mappa sulla quale devono essere disposti i possibili luoghi e le sedi relative alle squadre di pronto intervento. L'interfaccia progettata per questo algoritmo è simile all'interfaccia di un comune applicativo sviluppato per ambienti Microsoft Windows, strutturata a multi-finestra in cui ogni finestra ha uno specifico compito. La componente principale dell'interfaccia è formata dal menù di gestione dell'applicazione, con cui è possibile creare e gestire una mappa, salvare e gestire un progetto, calcolare la soluzione di un problema e salvarne il risultato.

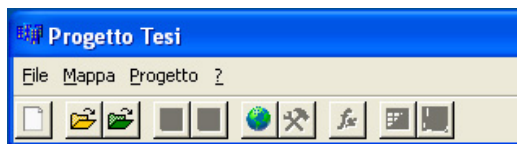


Figura 4.1: Finestra principale

L'interfaccia si basa su due concetti principali:

**Mappa** rappresentata da un'immagine di uno stradario relativo al territorio interessato su cui è possibile disporre i luoghi e le sedi d'interesse al problema

**Progetto** rappresenta la soluzione del problema con il relativo assegnamento rappresentato graficamente sulla mappa correlato dalle statistiche relative al calcolo della soluzione

## 4.2 Mappa

La finestra “Mappa” ha il compito di visualizzare lo stradario e di gestire il posizionamento dei luoghi e delle squadre di pronto intervento, la finestra è composta da una zona bianca destinata al caricamento dello stradario e alla visualizzazione del risultato finale. Per creare una nuova mappa si deve avere a disposizione l’immagine dello stradario relativo al luogo su cui si intende dislocare i luoghi, una volta caricata tale immagine è necessario definire le dimensioni del problema, ovvero il numero di luoghi, squadre e turni, inoltre è necessario impostare sia il time-out dell’algoritmo che la scala dello stradario ed infine il file in cui salvare le statistiche di esecuzione dell’algoritmo. La scala della mappa è un valore che rappresenta il numero di centimetri reali corrispondenti ad un centimetro della mappa, tale valore è utilizzato per rappresentare le distanze tra i vari luoghi e le sedi delle squadre. Terminata la fase di inserimento dei parametri è possibile posizionare sulla mappa i luoghi e le sedi del problema, la disposizione è molto semplice infatti è sufficiente cliccare sulla mappa nella posizione desiderata che il luogo oppure la sede verrà posizionata. Ultimata la fase di posizionamento è possibile salvare la mappa in modo che essa possa essere riutilizzata ed eventualmente modificata in un secondo momento.

Per la rappresentazione dei luoghi o delle sedi ho scelto dei quadrati con bordo colorato, blu per i luoghi e rosso per le sedi, per distinguere i vari luoghi e le varie sedi ho inserito il numero progressivo di creazione del luogo o della sede in modo da avere un riferimento univoco.

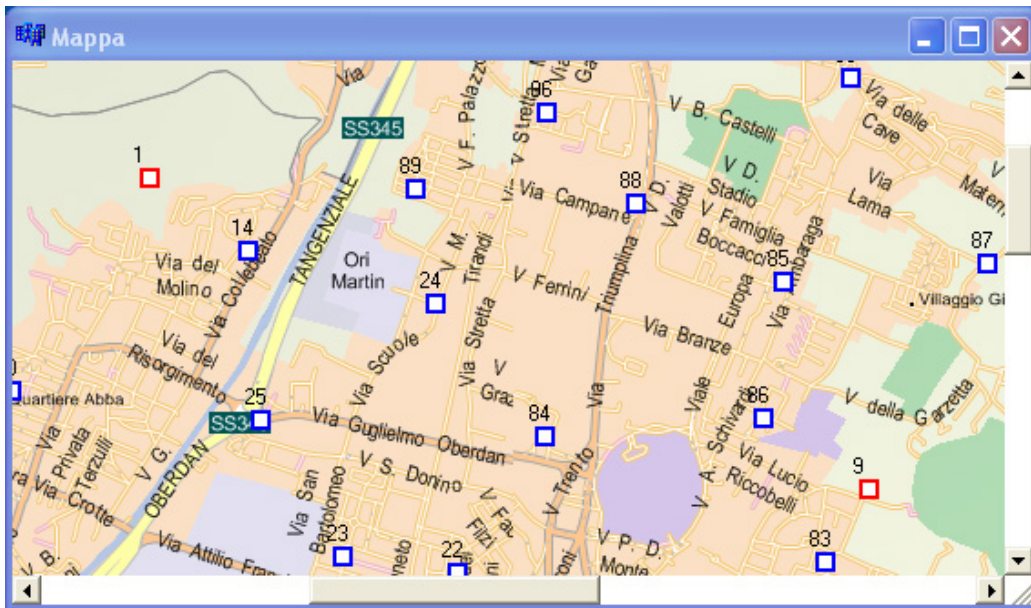


Figura 4.2: Esempio di mappa con luoghi e sedi

### 4.3 Progetto

Nel progetto trova posto la soluzione ottenuta dall'algoritmo, per calcolare tale soluzione è sufficiente utilizzare il pulsante "Calcola" dalla finestra "Strumenti" oppure accedere all'omonimo pulsante dalla finestra principale. I dati relativi alle distanza luogo-sede sono calcolati come distanza euclidea tra tutti i luoghi e le sedi, tali distanze sono successivamente arrotondate all'intero superiore, in modo da poter lavorare su dati di tipo intero. Una volta avviato l'algoritmo l'applicativo rimane in attesa della sua terminazione, durante l'esecuzione è possibile vedere il progresso dell'algoritmo nella finestra "Progresso" un cui vi è una barra di progressione ed un campo in cui visualizza la percentuale di avanzamento dell'algoritmo. Al termine dell'algoritmo l'interfaccia fornisce un messaggio sull'ottimalità della soluzione trovata, se essa è garantita oppure no, inoltre nella finestra "Mappa" si visualizza l'as-

segnamento relativo alla soluzione trovata. Ogni assegnamento luogo-sede è rappresentato mediante un linea che congiunge il luogo e la sede, ad ogni linea è associato un colore che ne distingue il turno a cui fa riferimento.

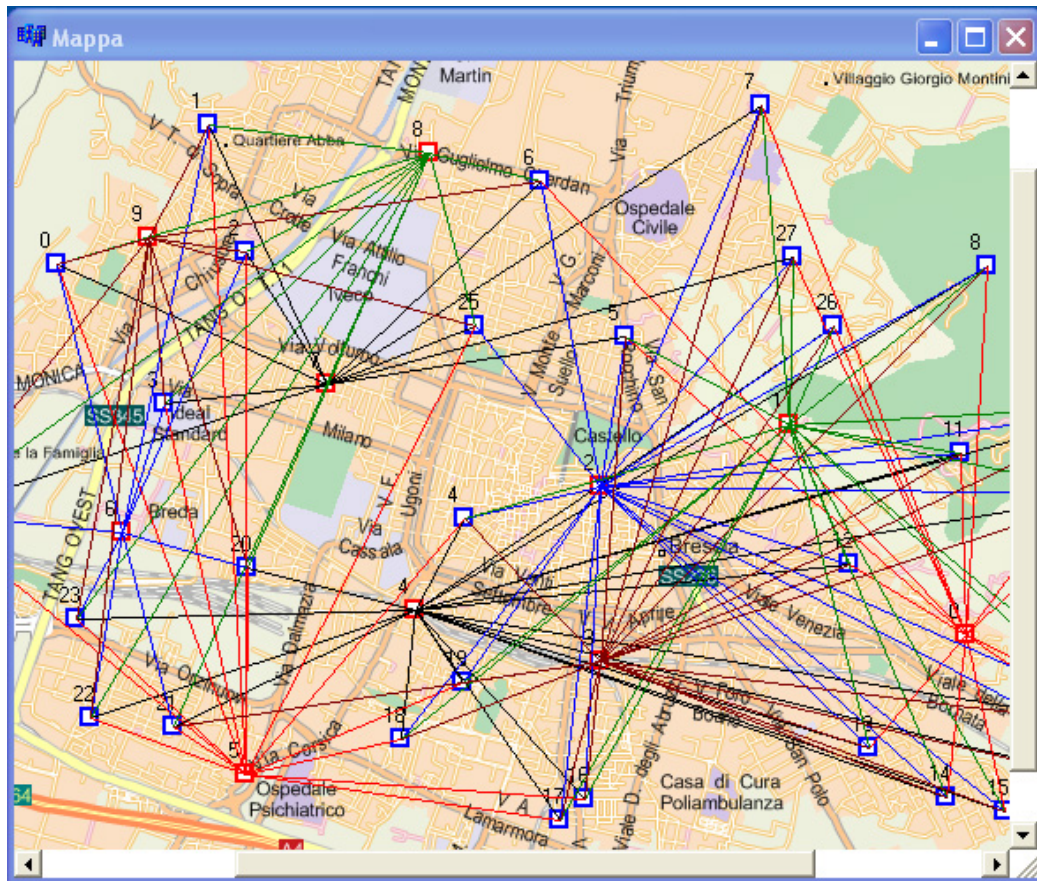
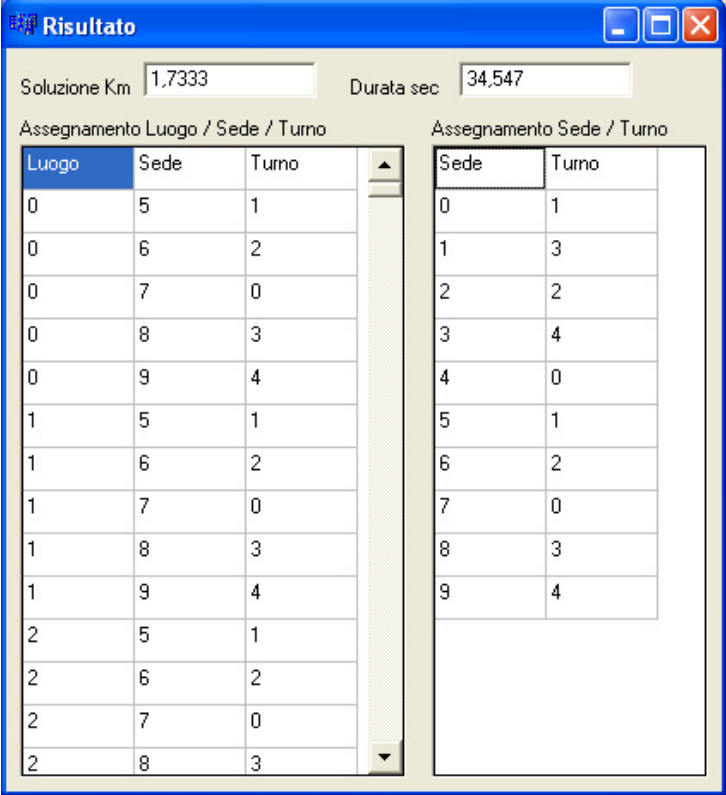


Figura 4.3: Esempio di assegnamento

## 4.4 Risultato

Al termine dell'algoritmo l'applicativo oltre a fornire un'informazione iniziale riguardante l'ottimalità della soluzione trovata, apre la finestra "Risultato" in cui è possibile vedere il valore della soluzione, la durata dell'algoritmo e gli assegnamenti dei luoghi alle sedi e la turnazione delle squadre nei vari turni. Il valore della soluzione è espresso in chilometri con un'approssimazione alla quarta cifra decimale, ciò garantisce che la soluzione abbia un'approssimazione di 5 centimetri.



The screenshot shows a window titled "Risultato" with a blue title bar. Inside, there are two input fields: "Soluzione Km" with the value "1,7333" and "Durata sec" with the value "34,547". Below these are two tables. The left table is titled "Assegnamento Luogo / Sede / Turno" and has columns "Luogo", "Sede", and "Turno". The right table is titled "Assegnamento Sede / Turno" and has columns "Sede" and "Turno".

Luogo	Sede	Turno
0	5	1
0	6	2
0	7	0
0	8	3
0	9	4
1	5	1
1	6	2
1	7	0
1	8	3
1	9	4
2	5	1
2	6	2
2	7	0
2	8	3

Sede	Turno
0	1
1	3
2	2
3	4
4	0
5	1
6	2
7	0
8	3
9	4

Figura 4.4: Esempio di risultato

Oltre alla finestra “Risultato” è possibile attivare la finestra “Statistiche” in cui sono riportate le statistiche riguardanti l’esecuzione dell’algoritmo, come:

- nodo e livello della soluzione
- durata dell’algoritmo
- lower bound massimo
- gap
- numero di nodi generati
- somma di tutte le distanze relative all’assegnamento
- assegnamento con la distanza massima

Questi dati forniscono informazioni sul comportamento dell’algoritmo e sulla qualità delle soluzioni fornite in caso che essa non abbia la garanzia di ottimalità.

The screenshot shows a window titled "Statistiche" with a table of statistics and a table of results. The statistics table includes fields like "Soluzione Km", "Nodo della soluzione", "Livello della soluzione", "Durata sec", "Time out sec", "Lower bound massimo", "Numero nodi generati", "Numero massimo di nodi aperti", "Numero nodi rimanenti nello heap", "Numero iterazioni", "Somma delle distanze", and "Distanze massime". The results table has columns for "Turno", "Luogo", "Sede", and "Distanza".

Statistiche			
Soluzione Km	1,7333	Ottima	
Nodo della soluzione	0		
Livello della soluzione	0		
Durata sec	34,547		
Time out sec	1200		
Lower bound massimo	1,7333		
Numero nodi generati	13		
Numero massimo di nodi aperti	6		
Numero nodi rimanenti nello heap	5		
Numero iterazioni	6		
Somma delle distanze	110,1407		
Distanze massime			
Turno	Luogo	Sede	Distanza
0	28	4	1,7333
1	1	5	1,3629
2	28	2	1,4592
3	22	8	1,3814

Figura 4.5: Esempio di statistiche

Grazie alla finestra “Strumenti” è possibile modificare la visualizzazione del risultato ottenuto, applicando le viste per:

- turno, in cui si visualizza l’assegnamento relativo al turno specificato
- sede, in cui si visualizza l’assegnamento dei luoghi alla sede specificata
- luogo, si visualizza l’assegnamento del determinato luogo alle varie sedi attive nei turni

In fase di visualizzazione dell’assegnamento scelto è possibile applicare uno zoom alla mappa in modo da avere una visione più globale o più specifica della soluzione del problema, è possibile salvare l’immagine dell’assegnamento visualizzato in modo da poterla stampare in un secondo momento.



The image shows a software window titled "Strumenti" (Tools) with a blue header bar. The window contains several filter sections and a list of action buttons. The "Vista per Turno" (View by Shift) section has a dropdown menu set to "Tutti" (All). The "Zoom" section has a dropdown menu set to "100". The "Vista per Sedi" (View by Seats) section has a dropdown menu set to "Tutte" (All). The "Vista per Luoghi" (View by Places) section has a dropdown menu set to "Tutti" (All). Below these filters are four buttons: "Salva" (Save), "Risultato" (Result), "Statistiche" (Statistics), and "Mappa" (Map).

Strumenti	
Vista per Turno	Zoom
Tutti	100
Vista per Sedi	
Tutte	
Vista per Luoghi	
Tutti	
Salva	
Risultato	
Statistiche	
Mappa	

Figura 4.6: Esempio di strumenti

# Bibliografia

- [1] G. Righini *Guida alla realizzazione di algoritmi "Branch-and-Bound"*,  
Note del polo - Rapporti interni 5, Marzo 2000
- [2] J.E. Beasley *Lagrangian Relaxation*, The Management School, Londra,  
Maggio 1992
- [3] GNU *Gnu Linear Programming Kit Reference Manual*, Draft Edition,  
Gennaio 2005
- [4] A. Bertossi *Algoritmi e strutture dati*, UTET Libreria, 2000
- [5] L.R. Ford, D.R. Fulkerson *Flows in Networks*, Princeton Univ.Pres, 1962
- [6] H.W. Kuhn *The Hungarian method for the assignement problem*,  
Nav.Res.Logistics Quarterly Vol 2, 1955
- [7] R. Bellman *Dynamic Programming*, Princeton Univ.Pres, 1957
- [8] L. Wolsey *Integer Programming*, Wiley, 1999
- [9] T. Ibaraki *Enumerative Approaches to Combinatorial Optimization*,  
Annals of Operation Research, Vol 10,11, J.C. Baltzer AG, Basel, 1987
- [10] Held M. Karp R.M. *The travelling-salesman problem and minimum  
spanning trees*, 1970

# Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Motivazione . . . . .	2
1.2	Descrizione informale . . . . .	2
1.3	Formulazione matematica . . . . .	4
1.4	Lower-Bound del problema . . . . .	7
<b>2</b>	<b>Solutori general-purpose</b>	<b>9</b>
2.1	Programmazione lineare intera . . . . .	10
2.2	Solutori general-purpose . . . . .	10
2.3	Risultati sperimentali . . . . .	11
<b>3</b>	<b>Un algoritmo Lagrangeano</b>	<b>15</b>
3.1	Rilassamento Lagrangeano . . . . .	16
3.2	Problema Lagrangeano duale e algoritmo del sottogradiente . . . . .	20
3.3	Algoritmo risolutivo del sottoproblema 1 . . . . .	23
3.4	Algoritmo risolutivo del sottoproblema 2 . . . . .	28
3.5	Euristica iniziale . . . . .	35
3.6	Euristica Lagrangeana . . . . .	37
3.7	Fissaggio di variabili . . . . .	39
3.8	Risultati sperimentali al nodo radice . . . . .	41

3.9	Branch-and-Bound . . . . .	44
3.9.1	Strategia di branching . . . . .	46
3.9.2	Strategia di esplorazione dell'albero . . . . .	49
3.10	Risultati Sperimentali . . . . .	56
<b>4</b>	<b>Interfaccia</b>	<b>63</b>
4.1	Aspetto generale . . . . .	64
4.2	Mappa . . . . .	65
4.3	Progetto . . . . .	66
4.4	Risultato . . . . .	68

# Elenco delle figure

1.1	Pseudo-codice dell'algoritmo per il calcolo del lower-bound . . .	8
2.1	Tabella dei risultati sperimentali . . . . .	13
2.2	Grafico dei tempi di calcolo . . . . .	14
3.1	Pseudo-codice dell'algoritmo risolutivo per il sottoproblema 1 .	25
3.2	Tabella di confronto dei risultati del sottoproblema 1 . . . . .	26
3.3	Confronto dei tempi di calcolo . . . . .	27
3.4	Grafico del problema di flusso massimo a costo minimo . . . . .	29
3.5	Esempio di grafico di matching . . . . .	32
3.6	Esempio di grafico di flusso . . . . .	32
3.7	Tebella di confronto dei risultati del sottoproblema 2 . . . . .	34
3.8	Grafico dei tempi di calcolo del sottoproblema 2 . . . . .	34
3.9	Pseudo-codice dell'euristica iniziale . . . . .	36
3.10	Pseudo-codice dell'euristica lagrangeana . . . . .	38
3.11	Pseudo-codice della procedura di fissaggio di variabili . . . . .	40
3.12	Tabella delle prestazioni del sottogradiente . . . . .	42
3.13	Grafico dei tempi di calcolo del sottogradiente . . . . .	43
3.14	Grafico del gap del sottogradiente . . . . .	44
3.15	Pseudo-codice dell'algoritmo Branch-and-bound . . . . .	46
3.16	Tabella di comparazione delle strategie di esplorazione . . . . .	51

3.17	Grafico di comparazione dei tempi di calcolo delle strategie . .	53
3.18	Grafico di comparazione delle percentuali di Gap . . . . .	54
3.19	Grafico di comparazione delle soluzioni . . . . .	55
3.20	Tabella dei risultati dell'algoritmo Branch-and-bound . . . . .	57
3.21	Grafico dei tempi di calcolo . . . . .	58
3.22	Grafico dell'occupazione di memoria . . . . .	59
3.23	Grafico del numero del nodo della soluzione . . . . .	60
3.24	Grafico del livello del nodo della soluzione . . . . .	60
3.25	Grafico della soluzione e massimo lower bound . . . . .	61
4.1	Finestra principale . . . . .	64
4.2	Esempio di mappa con luoghi e sedi . . . . .	66
4.3	Esempio di assegnamento . . . . .	67
4.4	Esempio di risultato . . . . .	68
4.5	Esempio di statistiche . . . . .	70
4.6	Esempio di strumenti . . . . .	71