UNIVERSITÀ DEGLI STUDI DI MILANO
DIPARTIMENTO DI TECNOLOGIE DELL'INFORMAZIONE
Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di PROGETTAZIONE E ANALISI DI ALGORITMI

Accoppiamento con vincoli di risorsa

Studenti:
Andrea Gardoni xxxxxx
Luca Uberti Foppa xxxxxx
Massimo Manara xxxxxx

Docente del corso:
Roberto Cordone

Anno Accademico 2006/2007

Indice

Introduzione

Il problema affrontato è quello dell'accoppiamento a costo minimo con vincoli di risorsa.
Si devono assegnare un certo numero di persone (o macchine o entità) a un certo numero di progetti (o lavori), in modo che ciascuna sia assegnata a un progetto diverso. Assegnare una persona a un progetto comporta un costo, ma anche l'uso di una risorsa. Si vuole minimizzare il costo totale di tutti gli assegnamenti, garantendo che l'uso totale della risorsa non superi una data disponibilità (Budget).
Un'applicazione pratica del problema può essere appunto l'assegnamento di lavoratori a progetti.
La tecnica utilizzata per raggiungere la soluzione del problema è di tipo branch and bound. [Vol05].

Capitolo 1

Il problema

1.1 Definizione

Il problema affrontato è l'accoppiamento a costo minimo con vincoli di risorsa.

Si devono assegnare n persone (o macchine o entità) a n progetti (o lavori), in modo che ciascuna persona sia assegnata a un progetto diverso. Assegnare la persona i al progetto j comporta un costo c_{ij}, ma anche l'uso di una risorsa in quantità limitata r_{ij}.

Il problema può essere definito tramite un grafo bipartito $G = (V_1 \cup V_2, E)$ dove il sottoinsieme di vertici V_1 rappresenta l'insieme delle persone, il sottoinsieme di vertici V_2 rappresenta l'insieme dei progetti, mentre l'insieme dei lati E rappresenta gli accoppiamenti possibili tra persone e progetti.
Si vuole minimizzare il costo totale di tutti gli assegnamenti, garantendo che l'uso totale della risorsa non superi una data disponibilità B (*Budget*).

1.2 Formulazione

DATI:

- $|V_1| = n$ persone e $|V_2| = n$ progetti
- c_{ij} costo di attribuzione progetto $j \in V_2$ alla persona $i \in V_1$
- r_{ij} risorsa consumata dalla persona $i \in V_1$ per il fare il progetto $j \in V_2$
- L'uso di risorse totale:

$$R = \sum_{i \in V_1} \sum_{j \in V_2} r_{ij} \tag{1.1}$$

- Il budget B, viene calcolato come (ρ è un parametro):

$$B = \frac{R}{n} \cdot \rho \qquad \rho \in \{0.6, 0.8, 0.1\} \tag{1.2}$$

VARIABILI:

- Una soluzione è individuata dalle variabili:

$$x_{i,j} = \begin{cases} 1 & \text{progetto } j \in V_2 \text{ alla persona } i \in V_1, \\ 0 & \text{altrimenti.} \end{cases}$$

VINCOLI:

- Una persona deve essere associata a un solo progetto:

$$\sum_{j \in V_2} x_{ij} = 1 \qquad i \in V_1 \tag{1.3}$$

- Un progetto deve essere associato a una sola persona:

$$\sum_{i \in V_1} x_{ij} = 1 \qquad j \in V_2 \tag{1.4}$$

- Vincolo di budget

$$\sum_{i \in V_1} \sum_{j \in V_2} r_{ij} x_{ij} \leq B \tag{1.5}$$

OBIETTIVO:

- Si vuole minimizzare il costo totale:

$$\min \sum_{i \in V_1} \sum_{j \in V_2} c_{ij} x_{ij} \tag{1.6}$$

Capitolo 2

L'algoritmo di minimizzazione

Per la risoluzione del problema utilizziamo la tecnica del Branch and Bound, la quale si basa sulla scomposizione del problema originale in sottoproblemi (nodi), più semplici da risolvere.
Per limitare il numero di nodi da esaminare si associa ad ogni nodo un "lower bound", ossia un limite inferiore, che indica una stima per difetto del valore di tutte le soluzioni complete ottenibili come discendenti di quel nodo. Anche un "upper bound" deve essere noto e viene usato come termine di confronto per i lower bounds dei nodi. L'upper bound è il valore della migliore soluzione trovata sinora. Se un nodo dell'albero ha un lower bound maggiore o uguale all'upper bound corrente, esso viene chiuso, cioè viene eliminato dalla lista dei nodi aperti. Infatti tutte le soluzioni complete che si otterrebbero da esso avrebbero un valore non migliore del valore della soluzione ottima corrente. La stessa cosa avviene se si dimostra che il consumo di risorsa sul nodo non può essere inferiore al budget perchè le soluzioni complete che sottrarrebbero non possono essere ammissibili .

2.1 Calcolo del lower bound

Per il calcolo del lower bound (`LB`), una prima idea è quella di ignorare il vincolo del *Budget* affinché il problema diventi risolvibile in tempo polinomiale $O(N^3)$, dato che si riduce a un linear assignement problem. Questo può essere risolto usando l'algoritmo descritto in [JV87], disponibile in rete come libreria `lap.c`.
Il linear assignement problem, è definito come segue:
DATI:

- $|V_1| = n$ persone e $|V_2| = n$ compiti

- c_{ij} costo di attribuzione del compito $j \in V_2$ alla persona $i \in V_1$

VARIABILI:

$$x_{i,j} = \begin{cases} 1 & \text{compito } j \in V_2 \text{ a persona } i \in V_1, \\ 0 & \text{altrimenti.} \end{cases}$$

VINCOLI:

- Una persona deve essere associata a un solo progetto:

$$\sum_{j \in V_2} x_{ij} = 1 \qquad i \in V_1 \tag{2.1}$$

- Un progetto deve essere associato a una sola persona:

$$\sum_{i \in V_1} x_{ij} = 1 \qquad j \in V_2 \tag{2.2}$$

- Vincolo di budget

$$\sum_{i \in V_1} \sum_{j \in V_2} r_{ij} x_{ij} \leq B \tag{2.3}$$

OBIETTIVO:

- si vuole minimizzare il costo totale:

$$\min \sum_{i \in V_1} \sum_{j \in V_2} c_{ij} x_{ij} \tag{2.4}$$

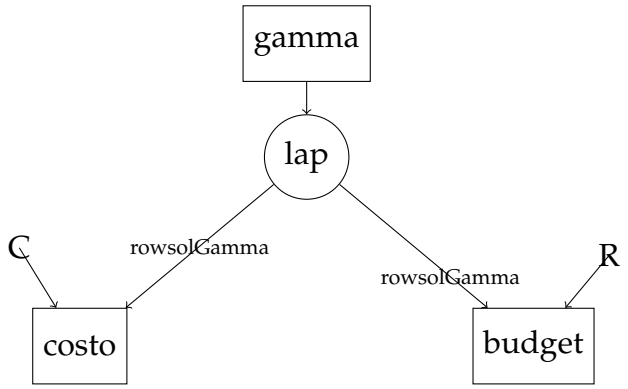
Il calcolo del lower bound, avviene nella funzione `ProcessBnode` da parte della funzione `CalcoloLB`, che riceve: il nodo corrente `N`, i dati del problema `PD`, e la soluzione migliore (Best Solution) `BS`. Nella struttura dati del nodo, `N.LB`, viene salvato il valore del lower bound calcolato tramite la funzione `lap`, la quale riceve come input la dimensione del problema, `dim` e la matrice dei costi `assigncost`, e fornisce come output la soluzione stessa, descritta attraverso i due vettori `LBcolsol` e `LBrowsol` che riportano, rispettivamente, la persona a cui è assegnato ogni progetto e il progetto assegnato a ogni persona.

2.2 Penalità Lagrangiane

Per evitare di generare tramite branching nodi che non possono portare un miglioramento, vengono inserite condizioni note come penalità lagrangiane. Se il costo ridotto di una cella della matrice dei costi, è maggiore dell'intervallo esistente tra l'upper bound e il lower bound, allora è possibile scartare tale cella fissando la corrispondente cella nella matrice dei vincoli a zero. Questo perché tale costo stima per difetto l'incremento che la funzione obiettivo subirebbe forzando la corrispondente variabile a 1.

2.3 Calcolo dell'upper bound

Per il calcolo dell'upper bound (`UB`), viene creata una nuova matrice γ (`gamma`) che contiene una combinazione convessa dei costi e del consumo di risorsa. Essa è quindi definita come $\gamma_{ij} = \alpha \cdot c_{ij} + (1 - \alpha) \cdot r_{ij}$; dove c_{ij} è la matrice dei costi, mentre r_{ij} quella delle risorse. Sulla matrice γ, viene eseguita la funzione `lap`, la quale restituisce il vettore delle soluzioni `rowsolGamma`; grazie alle coordinate fornite dal vettore, viene calcolato il costo sulla matrice dei costi, e il consumo sulla matrice delle risorse.



A ogni iterazione, il suo costo (`UB`), viene confrontato con l'upper bound globale nella struttura dati `BestSolution`, se è minore, viene aggiornata la migliore soluzione trovata fino a quel momento.

Infine α è un parametro che varia tra 0 e 1. Valutata (sempre con `lap.c`) la soluzione ottima rispetto a gamma, si può calcolarne il consumo totale di risorsa e il costo. Se il consumo è minore o uguale al budget, questo significa che si è dato troppo peso al consumo di risorsa. Quindi il

parametro α cresce come segue:

$$\begin{aligned} \alpha_m &= \alpha \\ \alpha_M &= \alpha_M \\ \alpha &= \frac{1}{2}\alpha + \frac{1}{2}\alpha_M. \end{aligned}$$

Se il consumo è maggiore del budget, questo significa che si è dato troppo poco peso al consumo di risorsa. Quindi il parametro α si modifica come segue:

$$\begin{aligned} \alpha_M &= \alpha \\ \alpha_m &= \alpha_m \\ \alpha &= \frac{1}{2}\alpha + \frac{1}{2}\alpha_m. \end{aligned}$$

L'aggiornamento di α viene eseguito per 10 iterazioni. Il valore iniziale di α è 1 (cioè si minimizza il solo consumo di risorsa). Se la soluzione corrispondente ha un consumo superiore al budget B, possiamo concludere che nessuna soluzione del nodo corrente è ammissibile, quindi il nodo si può chiudere senza altre elaborazioni.
In Pseudocodice 2.1 viene mostrata la logica del calcolo.

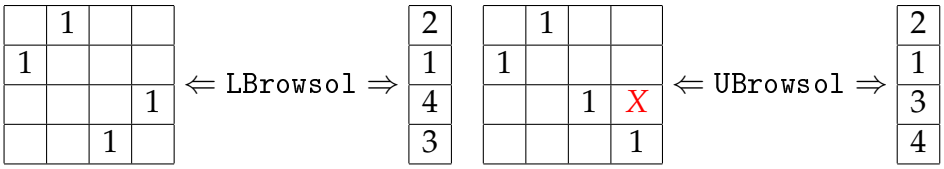
Pseudocodice 2.1: Calcolo dell'upper bound

```
1  CalcoloUB(Nodo, PD, BS)
   {
3      alpha = 0.5;
       alpham = 0.0;
5      alphaM = 1.0;

7      for (k=0; k<ITERAZIONI; k++) {
           for (i=0; i<dim; i++)
9              for (j=0; j<dim; j++)
                   gamma[i][j] = alpha*assigncost[i][j]+(1-alpha)*risorse[i][j];
11
           minimoConsumoRisorse = lap (dim, gamma, rowsolGamma, colsolGamma);
13
           // Calcolo costi
15         costo = 0.0;
           for ( i=0; i<dim; i++ )
17                 costo += assigncost[i][pta->BNinfo.UBrowsol[i]];

19                 // Calcolo del consumo
           consumo = 0.0;
21         for ( i=0; i<dim; i++ )
                   consumo += risorse[i][pta->BNinfo.UBrowsol[i]];
23
           if ( consumo <= budget ) {
25             alphaTemp = alpha;
               alpha = alpha/2 + alphaM/2;
27             alpham = alphaTemp;
```

```
            if ( costo<pBS->UB )
                pta->BNinfo.UB = costo;
        }
        else {
            alphaTemp = alpha;
            alpha = alpha/2 + alpham/2;
            alphaM = alphaTemp;
        }
    }
}
```

2.4 Strategia di branching

La strategia di branching consiste nel fissare una variabile ancora libera a 0 oppure a 1. La variabile viene scelta confrontando la soluzione associata al lower bound e quella associata all'upper bound, cioè i due vettori `LBrowsol` e `UBrowsol`. Se i due bound differiscono, le due soluzioni sono diverse. Si osservano le diversità e si considera la prima variabile presente nella soluzione dell'upper bound e assente in quella del lower bound. La figura seguente illustra un esempio: le matrici rappresentano le soluzioni associate al lower bound (a sinistra) e all'upper bound (a destra). A destra di ogni matrice compaiono i vettori `LBrowsol` e `UBrowsol` che contengono, per ogni riga, l'indice della colonna in cui compare un 1.



Una volta definito il punto di branching, la funzione `DeriveBnode` crea due figli, copiando nel figlio le informazioni del problema padre e forzando la variabile di branching a 1 nel primo figlio, a 0 nel secondo.

Pseudocodice 2.2: DeriveBnode

```
DeriveBnode (father, f, son)
{
    // Copio la matrice dei vincoli da padre in figlio
    for (i=0; i<father->BNinfo.dim; i++)
        for (j=0; j<father->BNinfo.dim; j++)
            son->BNinfo.vincoli[i][j] = father->BNinfo.vincoli[i][j];

    // Inserisco le informazioni di branching
    if ( f==1 ) {
        // Fisso a 0 tutti i valori contenuti nella colonna della cella
        // di branching tranne quello della cella stessa.
        son->BNinfo.vincoli[father->branch.i][father->branch.j] = 1;
        for (i=0; i<father->BNinfo.dim; i++)
            if (i != father->branch.i)
```

```
15                  son->BNinfo.vincoli[i][father->branch.j] = 0;

17          // Fisso a 0 tutti i valori contenuti nella riga della cella
            // di branching tranne quello della cella stessa.
19          for (j=0; j<father->BNinfo.dim; j++)
                if (j != father->branch.j)
21                  son->BNinfo.vincoli[father->branch.i][j] = 0;
        }
23      else
            son->BNinfo.vincoli[father->branch.i][father->branch.j] = 0;
25  }
```

2.5 Strategia di visita dell'albero

Le strategie di visita implementate sono tre:

- depth first search, ovvero in profondità
- breadth first search, ovvero in ampiezza
- best first search

2.5.1 Depth First Search

Questa strategia, inserisce il nodo `N` in cima alla lista `BT` dei nodi aperti. Sviluppa, quindi, l'albero in profondità, partendo dalla radice ed arrivando fino alle foglie. Ad ogni iterazione l'algoritmo sceglie di esplorare uno tra i nodi figli del nodo appena esplorato. Se questo non ha figli, l'algoritmo passa al nodo di più basso livello non ancora esplorato.
Quindi si esplora un ramo ancora prima di generare quelli successivi. Siccome i nodi vengono inseriti sempre dalla cima, per poter ottenere la DFS occorre inserire ogni nuovo nodo in cima alla lista.

Pseudocodice 2.3: Depth First Search

```
1 if (VisitStrategy == DEPTH_FIRST)
  {
3     inslista(N,primolista(BT),BT);
  }
```

2.5.2 Breadth First Search

Questa strategia, inserisce il nodo `N` in coda alla lista `BT` dei nodi aperti. Sviluppa, quindi, l'albero in larghezza, considerando prima tutti i nodi dello stesso livello per poi passare a quelli del livello sottostante. Si inizia calcolando l'upper bound del nodo radice, generando tutti i suoi figli e

calcolando i corrispondenti upper bound. Vengono quindi generati tutti i figli di ogni figlio della radice, e così via, esplorando sempre tutti i nodi di un livello prima di passare al livello successivo.

Pseudocodice 2.4: Depth First Search

```
if (VisitStrategy == BREADTH_FIRST)
{
    inslista(N,succlista(ultimolista(BT),BT),BT);
}
```

2.5.3 Best First Search

Questa strategia, mantiene la lista dei problemi aperti ordinata per bound crescente. Quindi inserisce il nuovo nodo prima del primo problema peggiore di esso.
Ad ogni iterazione l'algoritmo sceglie di esplorare uno tra i nodi a cui è associato il migliore bound della soluzione ottima.

Pseudocodice 2.5: Depth First Search

```
if (VisitStrategy == BEST_FIRST)
{
  pos = primolista(BT);
  while (! finelista(pos, BT) )
  {
      if ( pos->BNinfo.LB > pta->BNinfo.LB )
          break;
      pos = succlista(pos,BT);
  }
  inslista(pta,pos,BT);
}
```

Capitolo 3

I risultati sperimentali

La macchina utilizzata per le sperimentazioni ha le seguenti caratteristiche:

- CPU: Intel Core Duo T2400 1.87 GHz
- HD: 160 GB 7200 rpm
- RAM: 1 GB
- SO: Microsoft Windows XP [Versione 5.1.2600]
- marca/modello: Sony VAIO

3.1 I dati

I dati che vengono passati al problema sono: la dimensione `dim`, la matrice dei costi `assigncost`, la matrice delle risorse `risorse`.
La dimensione del problema è data dalla dimensione della matrice dei costi la quale viene letta da file (vedi Sezione 3.1.1), mentre la matrice delle risorse viene generata a partire da quella dei costi (vedi Sezione 3.1.2). Affinché l'istanza del problema possa essere codificata nel linguaggio AMPL, è stato necessario formattare il file in modo conforme. Questo ha consentito di risolvere problemi col risolutore gratuito GLPK e conservare quindi il valore ottimo. Le successive sezioni descrivono i passaggi per ottenere tale formattazione.

3.1.1 Lettura matrice dei costi

Dato un file (vedi Istanza 3.1) contenente la sola matrice dei costi, l'obiettivo della procedura di codifica è quello di formattare la stessa istanza in linguaggio AMPL (vedi Istanza3.2 in Sezione 3.1.3).

```
1 100
  52 89 40 77 89 14 9 77 92 77 52 53 96
3 96 92 76 33 81 92 84 36 81 47 55 87 35
```

Istanza 3.1: File istanza originale

Nella riga 1, in Istanza 3.1, troviamo la dimensione dell'istanza; le successive righe contengono i costi c_{ij}.

3.1.2 Generazione della matrice delle risorse

Per la generazione casuale dei consumi di risorsa r_{ij}, si è utilizzata l'equazione (3.1):

$$r_{ij} = min\left(c_{min} + c_{max} - c_{ij} + 10 \cdot \lambda\left(0, 10\right), c_{max}\right) \tag{3.1}$$

dove c_{min} e c_{max}, sono stati calcolati rispettivamente come il minimo ed il massimo elemento della matrice dei costi c_{ij}; infine, $\lambda\left(0, 10\right)$ rappresenta un numero casuale intero tra 0 e 10 generato utilizzando la libreria *rand.h*; il fattore 10 rappresenta un disturbo alla funzione che ritorna un numero intero casuale. Il minimo ottenuto tra i valori sopra e c_{rand}, calcolato come `assigncost[n*ran1(seed)][n*ran1(seed)]` rappresenta il consumo di risorsa. L'obietivo è quello di generare istanze difficili, in cui il consumo di risorsa è negativamente correlato al costo c_{ij}; in quanto quelle poco costose tendono a essere inammissibili; e viceversa le ammissibili a essere costose.

3.1.3 Formato di output delle istanze

La nuova formattazione è rappresentata in Istanza 3.2 ordinata per righe crescenti e per colonne crescenti:

```
1 param N := 100;
  param C :=
3 [0,0] 52 [0,1] 89 [0,2] ...
  param R :=
5 [0,0] 89 [0,1] 94 ...
  param B := 75;
7 end;
```

Istanza 3.2: File istanza nuovo

La prima riga riporta la dimensione dell'istanza. Segue la matrice dei costi nel formato $[i, j]$ c_{ij}, quindi la matrice delle risorse definita in modo analogo a quella dei costi, infine il budget B e infine il terminatore end.

3.2 Risultati

3.2.1 Consumi casuali

I risultati mostrati nella Tabella 3.1 si riferiscono a istanze del problema in cui sia la matrice dei costi che la matrice delle risorse, sono generate in modo casuale; la matrice delle risorse è comune alle cinque istanze di ciascuna dimensione.
La prima colonna contiene il nome dell'istanza; la seconda la dimensione, la terza il budget, la quarta colonna rappresenta il valore migliore ottenuto dal nostro algoritmo, la quinta riporta in secondi il numero di nodi generati, l'ultima colonna il tempo di computazione. Gli ottimi sono stati tutti verificati con GLPK.

Tabella 3.1: Risultati per le istanze costi risorse casuali

Nome ist.	Dim.	Budget	Sol.	n° nodi	Tempo
test11x11-1.dat	11	301	116	29	0.04
test11x11-2.dat	11	301	139	29	0.04
test11x11-3.dat	11	301	190	70	0.03
test11x11-4.dat	11	301	164	28	0.04
test11x11-5.dat	11	301	188	56	0.03
test12x12-1.dat	12	325	138	60	0.06
test12x12-2.dat	12	325	171	335	0.10
test12x12-3.dat	12	325	121	161	0.06
test12x12-4.dat	12	325	187	153	0.04
test12x12-5.dat	12	325	164	388	0.15
test13x13-1.dat	13	370	148	17	0.01
test13x13-2.dat	13	370	213	208	0.07
test13x13-3.dat	13	370	180	111	0.03
test13x13-4.dat	13	370	232	4380	0.76
test13x13-5.dat	13	370	118	212	0.04
test14x14-1.dat	14	425	141	80	0.09
test14x14-2.dat	14	425	149	369	0.15
test14x14-3.dat	14	425	175	450	0.28
test14x14-4.dat	14	425	158	122	0.06
test14x14-5.dat	14	425	72	107	0.04
test15x15-1.dat	15	437	205	5181	1.40
test15x15-2.dat	15	437	155	115	0.09

Segue...

Nome ist.	Dim.	Budget	Sol.	n° nodi	Tempo
test15x15-3.dat	15	437	201	699	0.23
test15x15-4.dat	15	437	246	1280	0.35
test15x15-5.dat	15	437	186	649	0.26

3.2.2 Consumi negativamente correlati ai costi

I risultati mostrati in Tabella 3.2 si riferiscono a istanze del problema in cui la matrice delle risorse è stata ricavata utilizzando il procedimento descritto in Sezione 3.1.2 in modo da essere negativamente correlate.
La prima colonna contiene il nome dell'istanza; la seconda la dimensione, la terza il budget, la quarta colonna rappresenta il valore migliore ottenuto dal nostro algoritmo, la quinta è l'ottimo (ottenuto con il codice GLPK), la sesta riporta il numero di nodi generati, l'ultima colonna il tempo di computazione in secondi.
Le prove sono state eseguite fissando un numero massimo di nodi pari a 500.000 e la strategia di visita best first.

Tabella 3.2: Risultati matrice costi/risorse - parzialmente casuali

Nome ist.	Dim.	Budget	Sol.	Ottimo	n° nodi	Tempo
test11x11.dat	11	301	217	217	1564	0.26
test12x12.dat	12	325	194	194	4454	0.62
test13x13.dat	13	370	319	319	151.092	188.31
test14x14.dat	14	425	242	242	25877	10.39
test15x15.dat	15	437	290	290	343968	1179.14
test16x16.dat	16	450	159	159	64	0.04
test17x17.dat	17	481	244	244	451296	2876.03
test18x18.dat	18	522	174	174	4213	1.2

La figura 3.1 rappresenta il numero di nodi utilizzati per trovare la soluzione, al variare della dimensione dell'istanza presa in esame. Come si evince dall'istogramma, questo valore è fortemente variabile. La figura 3.2 rappresenta il tempo di computazione, espresso in secondi, per trovare la soluzione. È chiaro che il tempo è fortemente correlato al numero di nodi analizzati.

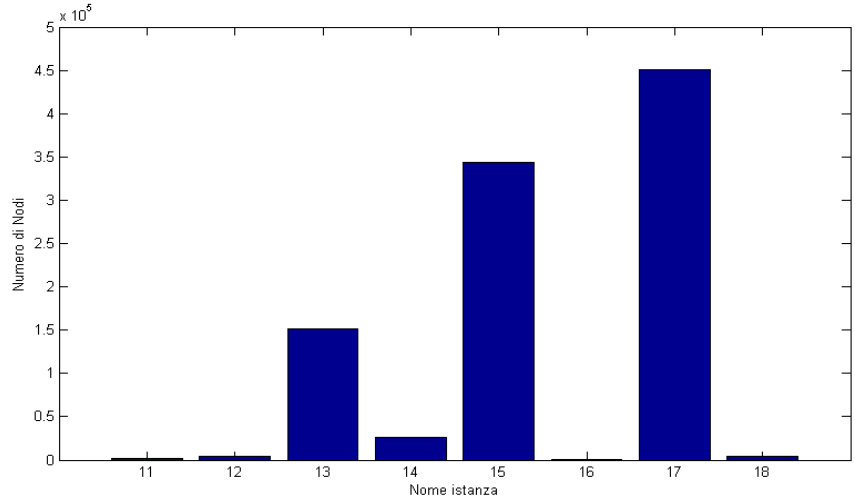


Figura 3.1: Numero di nodi necessari per risolvere ogni istanza

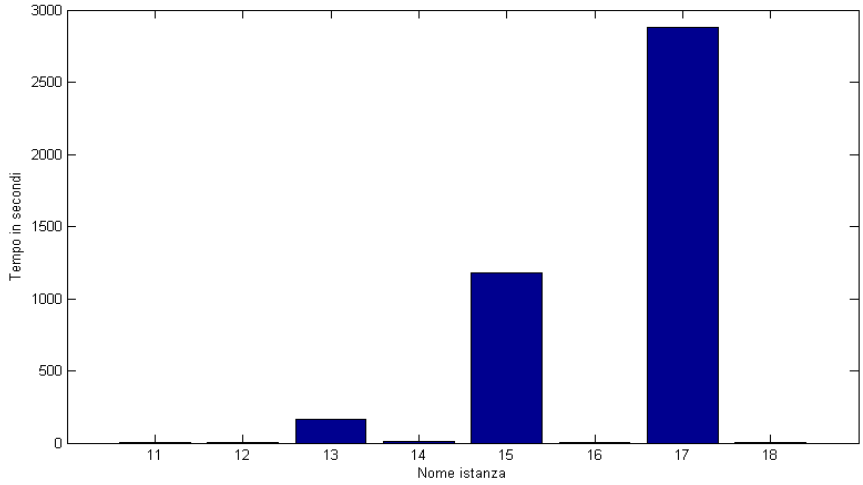


Figura 3.2: Tempo di calcolo necessario per risolvere ogni istanza

Bibliografia

[JV87] R. Jonker and A. Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38(4):325–340, 1987.

[Vol05] P.M.D. Lieshout A. Volgenant. A branch-and-bound algorithm for the singly constraind assignement problem. *European Journal of Operational Research*, 176:151–164, 2005.