UNIVERSITA' DEGLI STUDI DI PADOVA

FACOLTA' DI INGEGNERIA

Corso di Laurea triennale in Ingegneria dell'Automazione

TESINA DI LAUREA

ALGORITMI EURISTICI PER PROBLEMI DI SCHEDULING SU MACCHINE PARALLELE

Relatore: Prof. Michele Monaci

Laureando: Luca Casarin

Matricola: 578948-Iam

ANNO ACCADEMICO 2010-2011

INDICE

1. Introduzione 3

2. Calcolo del Lower Bound 4

3. Introduzione del vincolo sul numero massimo di attività parallele 8

   3.1 Primo algoritmo proposto 9

   3.2 Secondo algoritmo proposto 13

   3.3 Costruzione delle istanze di prova 15

   3.4 Makespan calcolato sulle istanze di prova 15

   3.5 Test dell'algoritmo su un'istanza al variare del numero di macchine . . 21

   3.6 Prestazioni del secondo algoritmo al variare del numero di *run* 28

   3.7 Confronto tra i due algoritmi 32

4. Conclusione 36

5. Bibliografia e Sitografia 37

6. Allegati 38

   6.1 Implementazione del primo algoritmo in java 39

   6.2 Implementazione del secondo algoritmo in java 50

   6.3 Risultati dei test effettuati sui due algoritmi 69

1.

INTRODUZIONE

Al giorno d'oggi l'inasprimento della concorrenza a livello globale rende sempre maggiore la necessità delle imprese di ottimizzare l'uso delle risorse aziendali al fine di massimizzare i profitti ed essere competitive nel mercato. Grazie ad un buon piano di lavoro è possibile, infatti, ridurre il tempo di lavorazione e, di conseguenza, i costi di produzione. Viene affrontato questo problema nell'elaborato: la schedulazione di più lavori su macchine parallele. In altre parole si vuole individuare una sequenza di lavoro sulle singole risorse aziendali al fine di ridurre il tempo di completamento (makespan) del progetto considerato. Con progetto viene inteso l'insieme delle attività A_i (i = 1, … , m), ciascuna di durata d_i nota, da intraprendere per la realizzazione di prodotti o servizi; fra alcune di questa attività esistono delle relazioni di precedenza. Nell'elaborato ho sviluppato io stesso degli algoritmi in grado di calcolare il makespan e fornire un dettagliato piano di lavoro. Questi permettono di calcolare l'istante in cui ogni attività deve iniziare ed individuano la macchina, tra quelle disponibili, che dovrà eseguirne la lavorazione. Rispettando tali indicazioni sarà possibile minimizzare il tempo per concludere il progetto. Successivamente questi algoritmi sono stati implementati nel linguaggio di programmazione java e testati su delle istanze di prova. Da tali test si sono potute valutare sia le prestazioni che gli algoritmi sviluppati garantiscono, sia alcune peculiarità dei problemi di scheduling.

2.

CALCOLO DEL LOWER BOUND

Il LOWER BOUND (limite inferiore) è la migliore soluzione ottenibile che, nei problemi di scheduling, consiste nel calcolare il makespan (tempo minimo per concludere il progetto) avendo a disposizione un numero infinito di risorse. Questo equivale nella realtà ad avere a disposizione un numero illimitato di macchine, di attrezzature e di manodopera; cosa che nella maggior parte dei casi non sarà possibile. Tuttavia la conoscenza del limite inferiore permette di valutare la qualità di una soluzione ammissibile, ovvero la sua distanza dall'ottimo. Il calcolo del LOWER BOUND è un problema facile risolvibile in un tempo polinomiale. Esistono due algoritmi molto efficienti: tecnica CPM e tecnica PERT.

CPM è l'acronimo di Critical Path Metod (metodo del percorso critico). Tale algoritmo è stato sviluppato nel 1957 dalla Catalytic Costruction Company: permette di individuare in un diagramma a rete la sequenza di attività critica ai fini della realizzazione del progetto.

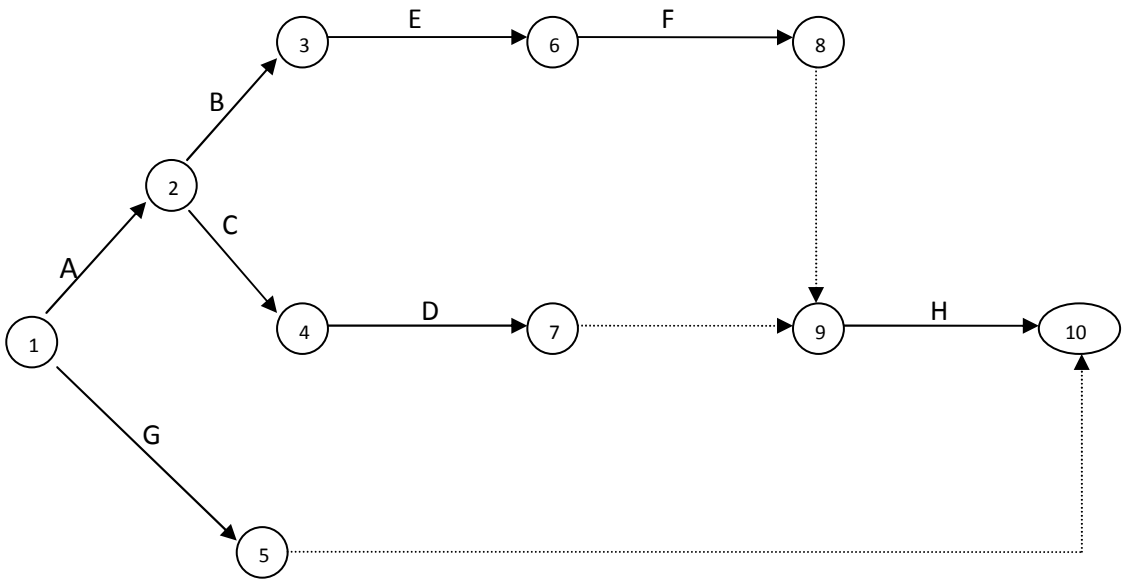
PERT sta invece per Program Evaluation and Review Technique, è una tecnica sviluppata nel 1958 dalla Booz, Allen &Hamilton, Inc per l'ufficio Progetti Speciali della marina degli Stati Uniti. All'epoca, in piena guerra fredda, tale programma venne sfruttato per diminuire i tempi e costi necessari per la produzione di sottomarini nucleari.

Questi algoritmi sono concettualmente molto simili e tutt'oggi sono di riferimento per la gestione di progetti. Un progetto, come è stato definito nell'introduzione, è un insieme di attività A_i ($i = 1, \dots, m$), ciascuna di durata d_i nota, tra le quali esistono delle relazioni di precedenza del tipo A_i « A_j (l'attività A_j può iniziare solo dopo il completamento dell'attività A_i). Nell'elaborato con i termini attività o lavoro si vuole intendere un'azione volta ad eseguire un particolare compito. Entrambe le tecniche citate utilizzano una rappresentazione del progetto per mezzo di un grafo $G(V, A)$ orientato, pesato e aciclico. Nel grafo ogni arco $A_h = (v_i, v_j)$ rappresenta un'attività, mentre ogni vertice indica il verificarsi di un evento: l'inizio o la fine di un lavoro. Inoltre i pesi $d(v_i, v_j)$, associati ad ogni arco, sono la durata dell'attività (v_i, v_j).

Può essere considerato il seguente esempio:

NOME ATTIVITA'	DURATA	PRECEDENZE
A	6	-
B	2	A
C	6	A
D	8	C, A
E	4	B
F	5	E
G	1	-
H	6	D, F

Tale progetto può quindi essere rappresentato dal seguente grafico:



Le attività fittizie (tratteggiate nel grafo) hanno durata nulla, nella realtà non esistono ma vengono introdotte per mantenere i rapporti di precedenza. Inoltre le tecniche PERT e CPM richiedono che il grafo G, che rappresenta il progetto, abbia un solo vertice iniziale e un solo vertice finale, come nell'esempio. Per concludere i vertici sono numerati in ordine topologico, cioè deve sempre valere $i<j$ per ogni arco (i, j). L'ordinamento è estremamente importante perché l'algoritmo seguendolo rispetta i vincoli di precedenza.

Il CPM, a differenza del PERT, utilizza stime deterministiche delle durate delle attività e non si preoccupa di introdurre gli effetti dell'incertezza nella realizzazione di queste. Tale tecnica è quindi molto utilizzata nel caso in cui la durata delle attività sia prevedibile con sufficiente precisione; questo comporta il vantaggio di calcolare a sua volta un risultato deterministico. L'algoritmo CPM permette quindi, nel caso delle condizioni sopra citate, di calcolare con maggiore precisione il makespan.
Se le attività presentano, invece, una ripetitività di esecuzione in cui la durata non si è rivelata costante, si ricorre alla tecnica PERT: questa rappresenta le attività come variabili aleatorie di cui occorre stimare le distribuzioni di probabilità. Il risultato finale fornito sarà a sua volta probabilistico.
Di seguito viene approfondito il funzionamento del CPM poiché, a differenza del PERT, ha come obbiettivo quello di ridurre la durata del progetto sopportando il minimo costo: esattamente l'obbiettivo finale dell'elaborato.
Tale tecnica associa a ciascun vertice *h* appartenente al grafo un istante minimo $Tmin_h$ prima del quale l'evento corrispondente non può accadere. In questo modo il tempo $Tmin_n$ associato all'ultimo vertice del grafo G rappresenta la durata minima del progetto. Per ogni vertice h viene inoltre calcolato un istante massimo $Tmax_h$ in cui l'evento corrispondente può accadere senza compromettere la durata minima del progetto. Dai valori $Tmin$ e $Tmax$ calcolati si possono ricavare le seguenti informazioni:

- L'istante minimo in cui un' attività può iniziare;
- L'istante massimo in cui un'attività può iniziare, ovvero di conseguenza lo slittamento massimo che può subire un lavoro senza comportare un ritardo nella realizzazione del progetto;
- Il makespan: il tempo minimo per realizzare il progetto sarà pari all'indice associato all'ultimo vertice, il quale rappresenta l'evento che segnala la conclusione dei lavori;
- Il cammino critico: è composto dalle attività che hanno slittamento nullo ($Tmin$= $Tmax$), ovvero dalle attività che non possono subire ritardi senza comportare un ritardo della data di conclusione. Ogni progetto ha almeno un cammino critico.

Applicando la tecnica PERT/CPM all'esempio si ottiene:

ATTIVITA'	DURATA	PRED	$Tmin$	$Tmax$	SLITTAMENTO	CRITICA
A	6	-	0	6	0	Si
B	2	A	6	11	3	No
C	6	A	6	12	0	Si
D	8	C, A	12	20	0	Si
E	4	B	8	15	3	No
F	5	E	12	20	3	No
G	1	-	0	26	25	No
H	6	D, F	20	26	0	Si

L'algoritmo CPM si può quindi riassumere:

algoritmo CPM

Begin

1) $Tmin[1] = 0$;
2) Ordina topoligicamente i vertici;
3) **for** h:=2 **to** n **do**
   $Tmin[h] := \max\{ Tmin[i] + d(i,h) : (i,h)$ appartiene $\delta^{-}(h) \}$;
4) $Tmax[n] := Tmin[n]$; // durata minima del progetto
5) **for** h := n-1 **downto** 1 **do**
   $Tmax[h] := \min\{ Tmax[j] - d(h,j) : (h,j)$ appartiene $\delta^{+}(h) \}$;

end.

Attenendomi a tale algoritmo ho realizzato nel linguaggio di programmazione java un programma che calcoli il LOWER BOUND di un progetto. I risultati ottenuti con tale programma vengono analizzati e discussi più avanti nell'elaborato. Il punto 5 non è stato implementato in quanto non utile ai fini della ricerca condotta.

3.

INTRODUZIONE DEL VINCOLO SUL NUMERO MASSIMO DI ATTIVITA' PARALLELE

Nel paragrafo 2 si è studiato il problema di scheduling su macchine parallele ipotizzando di avere a disposizione un numero infinito di risorse. Questa condizione ha reso il problema facile ed ha permesso di calcolare la soluzione ottima usando algoritmi noti. Tuttavia la tecnica CPM non è applicabile nei problemi di gestione che abitualmente le aziende devono affrontare, questo perché si deve considerare il numero limitato di risorse a disposizione. Diventa così importante per le imprese gestire in modo ottimale le macchine, le attrezzature e la manodopera che hanno a disposizione, organizzando i lavori in modo da concludere il progetto nel minore tempo possibile. Per far fronte a questo problema ho proposto io stesso degli algoritmi, successivamente implementati nel linguaggio di programmazione java, in grado di calcolare il makespan. Tali algoritmi sono quindi stati sviluppati imponendo, a differenza di quanto fatto nel paragrafo 2, un limite alle attività che possono essere svolte contemporaneamente. Va detto fin da subito che l'introduzione di questo vincolo rende il problema NP-difficile, ovvero diventa impossibile calcolare la soluzione ottima in un tempo polinomiale. È quindi necessario sviluppare algoritmi euristici, ossia algoritmi che non garantiscono di ottenere la soluzione ottima, ma in genere sono in grado di fornire una "buona" soluzione ammissibile per il problema. Questi solitamente sono strutturati in modo semplice e si basano su operazioni elementari. Per realizzare i programmi che successivamente vengono proposti mi sono rifatto a una delle principali tecniche algoritmiche conosciute: gli algoritmi greedy. Tali algoritmi greedy determinano la soluzione attraverso una sequenza di decisioni "localmente ottime", senza mai tornare, modificandole, sulle decisioni prese. Questi algoritmi hanno il vantaggio di essere facili da implementare e di avere una notevole efficienza computazionale. Questo a discapito di non garantire l'ottimalità, ed a volte neppure l'ammissibilità, della soluzione trovata. Gli algoritmi greedy, categoria a cui fanno parte quelli proposti, possono essere semplificati con il seguente schema:

Algoritmi greedy (E, S, Q)
Begin
S := ∅;
Q := E;
repeat
e := *best*(Q);
Q := Q \ {e};
S = S U e;
Until Q = ∅;
End.

Nel caso del problema di scheduling E è l'insieme di tutte le attività necessarie per la realizzazione del progetto, S è l'insieme degli elementi che sono stati inseriti nella soluzione parziale ed, infine, Q è l'insieme degli elementi ancora da esaminare.

3.1 PRIMO ALGORITMO PROPOSTO

Algoritmo 1
Begin
Q:= *array che contiene tutte le attività del progetto;*
n:= *numero totale attività;*
m:= *numero totale delle macchine disponibili;*
// *ordino le attività in modo che abbino durata crescente*
1) i := 0;
repeat
S[i] := a | $durata_a < durata_j$ ∀ j ∈ Q; // $a \in Q$
Q := Q\{a};
i++;
until Q = ∅;

```
2)      // ordino le attività in modo che siano rispettati i vincoli di precedenza;
        i := 0;
        repeat
            for k=0 to n-1 do {
                if(S[k] non ha vincoli di precedenza)   Q[i] := S[k];
                S := S\{S[k]};
                Rimuovere_vincolo(S[k]);
                i++;
                Continue; }
        until S = ∅;
3)      for h:=0 to n-1 do {
            M[1].fare_lavoro(Q[h]);
4)          // salvo l'istante in cui inizia il lavoro h-esimo
            Q[h].set_Tmin(M[1].get_fine_lavori());
5)          // salvo l'istante in cui la macchina conclude il lavoro h-esimo
             d = M[1].get_fine_lavori() + Q[h].get_durata();
             M[1].set_fine_lavori(d);
6)          // salvo nella posizione 1 dell'array la macchina "più scarica"
             temp = M[1];
             M[1] = M[j]  |  M[j].get_fine_lavori() < M[i].get_fine_lavori() ∀
i=2,…,m ;
             M[j] = temp;   }
        end.
```

L'algoritmo da la precedenza, ogni volta che è possibile scegliere, alle attività con tempo di esecuzione minore. All'inizio tutte le macchine sono scariche, ossia M(j)=∅ per j=1,…,m. Ad ogni iterazione viene selezionato un lavoro *i* ancora da assegnare e lo si assegna alla macchina più "scarica", ossia a quella con tempo di completamento, relativo alla soluzione parziale, più basso. In caso di parità viene usata una qualunque delle macchine col tempo di completamento corrente minimo. Il criterio con cui viene scelto il lavoro *i* , come già detto, consiste nel far iniziare l'attività con durata minore tra quelle ammissibili. Per implementare ciò vengono ordinate prima le attività (*punti 1 e 2*) in modo che, facendo iniziare i lavori seguendo tale ordine, ad ogni iterazione

sono sempre rispettati sia il criterio di scelta che i vincoli di precedenza. Per realizzare tale ordinamento topologico vengono prima disposte le attività in ordine crescente di durata senza considerare le precedenze. Solo a questo punto si riordina il tutto tenendo conto dei vincoli di precedenza. Riprendendo l'esempio visto nel paragrafo 2 si ottiene applicando l'algoritmo il seguente ordinamento topologico:

(INIZIO)

ATTIVITA'	DURATA
A	6
B	2
C	6
D	8
E	4
F	5
G	1
H	6

(PUNTO 1)

ATTIVITA'	DURATA
G	1
B	2
E	4
F	5
A	6
C	6
H	6
D	8

(PUNTO 2)

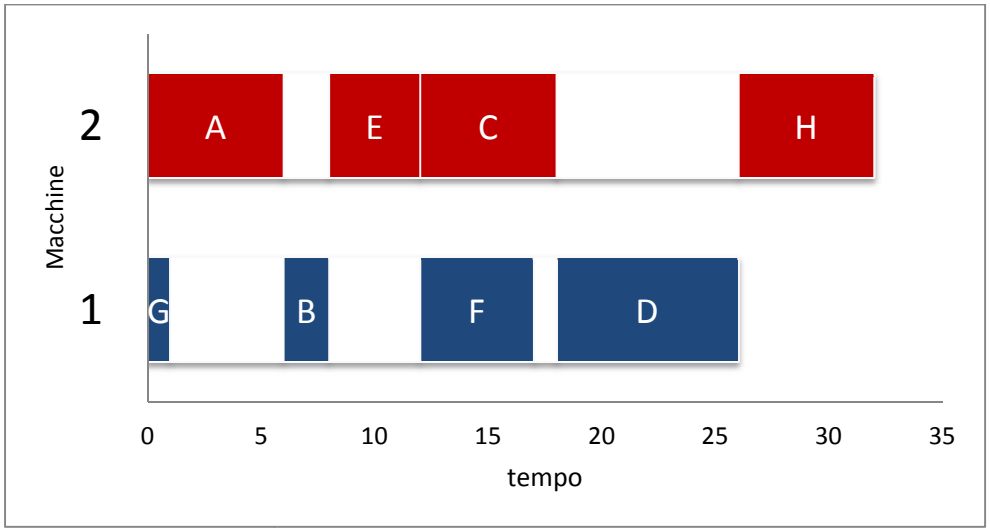
ATTIVITA'	DURATA	PRECEDENZE
G	1	-
A	6	-
B	2	A
E	4	B
F	5	E
C	6	A
D	8	C, A
H	6	D, F

Ogni volta che un'attività viene assegnata ad una macchina (punto 3) vengono anche aggiornati l'istante di inizio del lavoro i scelto ($Tmin_i$) e l'istante di fine lavori parziale della macchina che lo esegue. Il $Tmin$ viene posto dall'algoritmo, nel *punto 4*, pari al valore massimo tra l'istante in cui la macchina in questione si è "liberata" e gli istanti in cui sono terminate le attività di cui, causa vincoli di precedenza, è necessario attendere la conclusione. Questo permette di evitare errori nel caso in cui il lavoro j, necessario per iniziare il lavoro i selezionato secondo l'ordinamento fatto, debba ancora essere completato da una seconda macchina. Nel punto 5 la soluzione parziale viene aggiornata dall'algoritmo e posta pari all'istante in cui la macchina si scarica nuovamente dopo aver eseguito il lavoro: questo equivale quindi a porla pari alla somma tra $Tmin_i$, aggiornato nel punto precedente, e la durata dell'attività i considerata. L'algoritmo, infine, termina quando sono state eseguite tutte le attività.

Applicando tale procedimento all'esempio studiato e imponendo il vincolo di 2 attività parallele massime si ottiene soluzione euristica pari a 32. Il makespan è quindi aumentato, come era logico immaginare.
Di seguito sono riportati in tabella gli istanti, calcolati con il programma java realizzato (vedi allegati), in cui iniziano i singoli lavori:

ATTIVITA'	DURATA	PRED	
A	6	-	0
B	2	A	6
C	6	A	12
D	8	C, A	18
E	4	B	8
F	5	E	12
G	1	-	0
H	6	D, F	26

Tale risultato può essere rappresentato con un diagramma di Grantt, il quale fornisce una buona rappresentazione temporale del progetto: può essere visto come un calendario dei lavori, utile al fine di pianificare, coordinare e illustrazione lo stato di avanzamento del progetto. Da questo grafico si capisce quali sono le attività critiche e quella che è stata la divisione dei lavori tra le macchine.
Le attività critiche, ovvero quelle che non possono subire ritardi senza rimandare la data di scadenza del progetto sono: A, B, E, C, D e H.
Infine l'algoritmo assegna alla macchina numero 1 le attività G, B, F e D; mentre alla macchina numero 2 le attività A, E, C e H.



3.2 SECONDO ALGORITMO PROPOSTO

Algoritmo 2

```
 Begin
      S:= array che contiene tutte le attività del progetto;
       n:= numero totale attività;
       m:= numero totale delle macchine disponibili;
       r:= numero di run;
1)     //  ordino le attività in modo che siano rispettati i vincoli di precedenza;
       i := 0;
       repeat
       for k=0 to n-1 do    {
             if(Q[k] non ha vincoli di precedenza)   S[i] := Q[k];
             Q := Q\{Q[k]};
              Rimuovere_vincolo(Q[k]);
              i++;
              Continue;    }
        until Q = ∅;
       Q:=S;
2)      //   ripeto il procedimento r volte
        for j=1 to r do  {
3)           // assegno casualmente i lavori alle m macchine
             for y=0 to n-1do   {
                    h:= numero random compreso tra 1 e m;
                    M[h].Fare_lavoro(Q[y]);
4)                  //  salvo l'istante in cui inizia il lavoro y-esimo
                    Q[y].set_Tmin(M[h].get_fine_lavori());
5)                  //  salvo l'istante in cui la macchina conclude il lavoro h-esimo
                    d = M[h].get_fine_lavori() + Q[y].get_durata();
                    M[h].set_fine_lavori(d);    }
6)          //  salvo in un array il makespan ottenuto
            for w:=1 to m do   {
            if(M[w].get_fine_lavori() < V[j])    V[j]:=M[].get_fine_lavori();   }
       }   //   viene ripetuto il ciclo "for" r volte
```

```
7)      //  la soluzione finale dell'algoritmo è il migliore makespan ottenuto
        for  i:= 1 to r do  {
              if(V[i]<S)   S:=V[i];   }
 end.
```

Anche questo secondo algoritmo appartiene alla categoria greedy e fa della velocità di calcolo il suo punto di forza. Inizialmente, nel punto 1, le attività che compongono l'istanza vengono ordinate in modo da rispettare i vincoli di precedenza. Per realizzare questo, come fatto per il precedente algoritmo, si ordina l'array Q in modo che l'attività i sia preceduta nell'ordine da tutte le attività che necessariamente devono essere ultimate per permetterne l'inizio. A questo punto ogni lavoro viene assegnato in modo casuale alla macchina che lo deve svolgere. Ad ogni iterazione vengono anche aggiornati l'istante di inizio del lavoro i scelto ($Tmin_i$) e l'istante di fine lavori parziale della macchina che lo esegue. Il $Tmin$ viene posto dall'algoritmo, come nel precedente algoritmo, pari al valore massimo tra l'istante in cui la macchina in questione si è "liberata" e gli istanti in cui sono terminate le attività di cui, causa vincoli di precedenza, è necessario attendere la conclusione. Si ricorda che questo permette di evitare errori nel caso in cui il lavoro j, necessario per iniziare il lavoro i selezionato, debba ancora essere completato da una seconda macchina. Anche la soluzione parziale viene aggiornata dall'algoritmo e posta pari all'istante in cui la macchina si scarica nuovamente dopo aver eseguito il lavoro: questo equivale quindi a porla pari alla somma tra $Tmin_i$, aggiornato nel punto precedente, e la durata dell'attività i considerata.
Il makespan ottenuto in questo modo non è sicuramente affidabile a causa dell'assegnamento casuale che è stato fatto. Ad esempio potrebbe anche verificarsi che tutti i lavori siano assegnati alla stessa macchina, il che rende l'organizzazione fatta molto scadente. Tuttavia, grazie ai tempi di calcolo molto veloci dell'algoritmo, ho realizzato un programma che ripeta r volte questo procedimento. Tale tecnica si basa sulle leggi probabilistiche: se ripeto più volte una data esperienza aumentano le probabilità di successo (Legge dei Grandi Numeri).

3.3 COSTRUZIONE DELLE ISTANZE DI PROVA

Di questi algoritmi sono state studiate le soluzioni ottenute al variare del numero di attività totali, al variare del numero di macchine disponibili e, infine della probabilità di precedenza. Sono quindi stati scelti progetti di $n = 10, 50, 100, 200$ attività e per ognuna di queste situazioni si è supposto un numero di macchine disponibili pari a $n/4$, $n/3$ e $n/2$. Tale valore è stato approssimato per eccesso, risultando:

- 3, 4 e 5 macchine nel caso $n=10$;
- 13, 17 e 25 macchine nel caso $n=50$;
- 25, 34 e 50 macchine nel caso $n=100$;
- 50, 67 e 100 macchine nel caso $n=200$.

Nelle istanze di prova la durata delle attività è stata scelta random, compresa tra 1 e 10. Invece per determinare le precedenze si è generato, per ogni coppia di job (i,j) con $i < j$, un numero casuale *P*, compreso tra 0 e 1, imponendo la precedenza tra i job i e j se e solo se $P < Pmin$, dove $Pmin$ e' un qualche valore di probabilità prefissato. Il valore $Pmin$ è stato posto pari: $Pmin = 0.8$ (molte precedenze), $Pmin = 0.5$ (numero medio di precedenze) e $Pmin = 0.2$ (poche precedenze). Per ognuna di queste combinazioni si è testato l'algoritmo su 5 istanze per un totale di 180 problemi.

3.4 MAKESPAN CALCOLATO SULLE ISTANZE DI PROVA

Su queste istanze generate casualmente sono stati testati entrambi gli algoritmi realizzati. Per il momento viene considerato, per il secondo algoritmo, un numero di run r pari a 100, valore sufficiente per garantire buone prestazioni.
Nelle tabelle che successivamente sono riportate sono inserite le medie dei risultati ottenuti dai vari test effettuati, i risultati sulle singole istanze sono presenti in allegato.
Le istanze sono suddivise in base al valore $Pmin$ che le caratterizza. Per ognuna di queste categorie viene riportata la soluzione euristica calcolata dagli algoritmi, il lower boud, il rapporto Z/LB tra questi 2 valori e il tempo di calcolo.
I risultati inoltre si distinguono nel seguente modo:

= soluzione euristica calcolata dal primo algoritmo;

= soluzione euristica calcolata dal secondo algoritmo.

- Ponendo la probabilità di precedenza $\underline{Pmin=0,2}$ i risultati ottenuti sono:

1. SOLUZIONE EURISTICA (Z):

ATTIVITA' (n)	$n/4$ MACCHINE	$n/3$ MACCHINE	$n/2$ MACCHINE
10	33.6	26.6	25.6
	29.8	24.4	24.6
50	101.6	86.2	87.4
	103.4	87.6	87.4
100	178	174.8	175.6
	179.6	176.2	175.6
200	357.8	356	341.4
	361.2	357	341.4

2. LOWER BOUND (LB):

ATTIVITA' (n)	$n/4$ MACCHINE	$n/3$ MACCHINE	$n/2$ MACCHINE
10	28	24.4	24.6
50	99.4	86.2	87.4
100	178	174.8	175.6
200	357.8	356	341.4

3. Z/LB:

ATTIVITA' (n)	$n/4$ MACCHINE	$n/3$ MACCHINE	$n/2$ MACCHINE
10	1.2	1.09	1.04
	1.06	1	1
50	1.02	1	1
	1.04	1.02	1
100	1	1	1
	1.01	1.01	1
200	1	1	1
	1.01	1.003	1

4. TEMPO DI CALCOLO (in nanosecondi):

ATTIVITA' (n)	$n/4$ MACCHINE	$n/3$ MACCHINE	$n/2$ MACCHINE
10	0	6.4	3.2
	28,2	18,6	22
50	18.6	18.6	15.8
	100,2	103,4	109,2
100	31.4	31.2	28
	143,4	140,6	153,2
200	109.2	106.2	106.4
	259,6	265,6	299,8

- Ponendo la probabilità di precedenza $\underline{Pmin=0,5}$ i risultati ottenuti sono:

1. SOLUZIONE EURISTICA (Z):

ATTIVITA' (n)	$n/4$ MACCHINE	$n/3$ MACCHINE	$n/2$ MACCHINE
10	34,8	41,2	45,8
	33,6	40,6	45,8
50	167,4	172,4	170,8
	167,4	172,4	170,8
100	338,8	348	348,2
	338.8	348	348.2
200	708,4	684,4	676,4
	708.4	684.4	676.4

2. LOWER BOUND (LB):

ATTIVITA' (n)	$n/4$ MACCHINE	$n/3$ MACCHINE	$n/2$ MACCHINE
10	33,2	40,6	45,8
50	167,4	172,4	170,8
100	338,8	348	348,2
200	708,4	684,4	676,4

3. Z/LB:

ATTIVITA' (n)	$n/4$ MACCHINE	$n/3$ MACCHINE	$n/2$ MACCHINE
10	1.048	1.015	1
	1.012	1	1
50	1	1	1
	1	1	1
100	1	1	1
	1	1	1
200	1	1	1
	1	1	1

4. TEMPO DI CALCOLO (in nanosecondi):

ATTIVITA' (n)	$n/4$ MACCHINE	$n/3$ MACCHINE	$n/2$ MACCHINE
10	3,2	3	0
	21,6	25,2	34,2
50	15,6	15,6	15,6
	109,4	112,4	115,6
100	37,4	37,4	37,8
	172	162,8	168,4
200	147	156	147
	465,4	462,6	468,6

- Ponendo la probabilità di precedenza $Pmin=0,8$ i risultati ottenuti sono:

1. SOLUZIONE EURISTICA (Z):

ATTIVITA' (n)	$n/4$ MACCHINE	$n/3$ MACCHINE	$n/2$ MACCHINE
10	52	43,4	46,2
	52	43,4	46.2
50	240,2	230,6	227,4
	240.2	230.6	227.4
100	476,6	451,8	476,8
	476.6	451.8	476.8
200	936,4	961,2	957,6
	936.4	961.2	957.6

2. LOWER BOUND (LB):

ATTIVITA' (n)	$n/4$ MACCHINE	$n/3$ MACCHINE	$n/2$ MACCHINE
10	52	43,4	46,2
50	240,2	230,6	227,4
100	476,6	451,8	476,8
200	936,4	961,2	957,6

3. Z/LB:

ATTIVITA' (n)	$n/4$ MACCHINE	$n/3$ MACCHINE	$n/2$ MACCHINE
10	1	1	1
	1	1	1
50	1	1	1
	1	1	1
100	1	1	1
	1	1	1
200	1	1	1
	1	1	1

4. TEMPO DI CALCOLO (in nanosecondi):

ATTIVITA' (n)	$n/4$ MACCHINE	$n/3$ MACCHINE	$n/2$ MACCHINE
10	0	3,2	3,2
	28,2	25	28
50	15,6	19	18,8
	115,8	112,4	112,4
100	53	50	47
	181,4	178	181,2
200	187,4	165,8	171,8
	662,6	640,8	612,6

Dai risultati ottenuti si possono trarre importanti informazioni sia sulle prestazioni dei due algoritmi sviluppati, sia sui problemi di organizzazione delle attività :

I. Entrambi gli algoritmi permettono di calcolare una "buona" soluzione euristica. Questo lo si può vedere dal rapporto tra il makespan e il rispettivo lower bound ottenuti su un'istanza. Inoltre già da questi primi test si nota che il primo algoritmo sviluppato funziona meglio sulle istanze grandi, mentre il secondo sulle istanze piccole. Si sono ottenuti, infatti, prestazioni migliori del secondo sulle istanze costituite da 10 job, mentre prestazioni migliori del primo sulle istanze di 50, 100 e 200 attività.

II. Entrambi gli algoritmi hanno buone prestazioni anche per quando riguarda i tempi di calcolo. Questa è una prerogativa degli algoritmi greedy, i quali svolgendo operazioni per lo più elementari e veloci permettono di calcolare la soluzione prontamente. In questo caso si sono riscontrati tempi di calcolo dell'ordine del nanosecondo per le istanze più piccole e dell'ordine del microsecondo per quelle più grandi. Sebbene siano entrambi soddisfacenti è sicuramente molto più veloce il primo algoritmo. Per quanto riguarda la seconda tecnica studiata si potrebbe diminuire il numero di run al fine di migliorare il tempo di calcolo, questo a discapito della qualità della soluzione trovata.

III. Si può notare che all'aumentare del numero dei vincoli di precedenza tra le attività, ovvero all'aumentare del valore $Pmin,$ aumenta anche il valore della soluzione euristica e del lower bound trovati. Si nota inoltre che, sempre aumentando il valore $Pmin$, sono necessarie meno macchine per raggiungere la migliore soluzione ammissibile. Infatti, un maggior numero di vincoli limita il numero di attività parallele.
Il tempo di calcolo invece rimane pressoché invariato.

IV. Aumentando il numero di attività dell'istanza aumentano di conseguenza sia il tempo necessario per terminare il progetto, sia il tempo di calcolo. Si osserva inoltre che per raggiungere la migliore soluzione ammissibile, ovvero quando la soluzione euristica uguaglia il lower bound, è necessario un minor numero di macchine. Infatti nelle tabelle il rapporto $Z/_{LB}$ migliora all'aumentare del numero n di lavori, questo finché non raggiunge il valore 1.

V. Infine si è osservato che all'aumentare del numero di macchine a disposizione il valore della soluzione euristica trovata dall'algoritmo (makespan) diminuisce. Dai valori di media riportati nelle precedenti tabelle questo non è visibile; si è quindi

testato l'algoritmo su alcune istanze d'esempio al variare del numero di macchine. I risultati ottenuti sono riportati nel paragrafo seguente.

3.5 TEST DELL'ALGORITMO SU UN ISTANZA AL VARIARE DEL NUMERO DI MACCHINE

Si sono testati gli algoritmi su una specifica istanza al variare del numero di macchine *m = 1,…, n* (dove n è il numero di attività totali). Questo test è utile, ad esempio, per valutare il numero di risorse necessarie per terminare un progetto entro i tempi richiesti dal committente. Un'altra applicazione può essere quella di sfruttarlo per valutare un buon compromesso tra i risultati che si vogliono ottenere e il numero di risorse in cui si deve investire. Infatti, causa i costi spesso elevati dei macchinari, è bene valutare se le migliorie che un acquisto comporta valgono la spesa necessaria. Un buon indice per valutare la qualità della soluzione euristica ottenuta con m macchine, è dato dal rapporto tra la soluzione stessa e il lower bound (*paragrafo 2*).
Di seguito sono riportati i risultati ottenuti considerando istanze con probabilità di precedenza $Pmin=0,2,$ ovvero poche precedenze. Questi sono stati calcolati applicando il primo algoritmo. Vengono riportati unicamente questi nell'elaborato perché le conclusioni che verranno ricavate sono analoghe per entrambe le tecniche sviluppate.

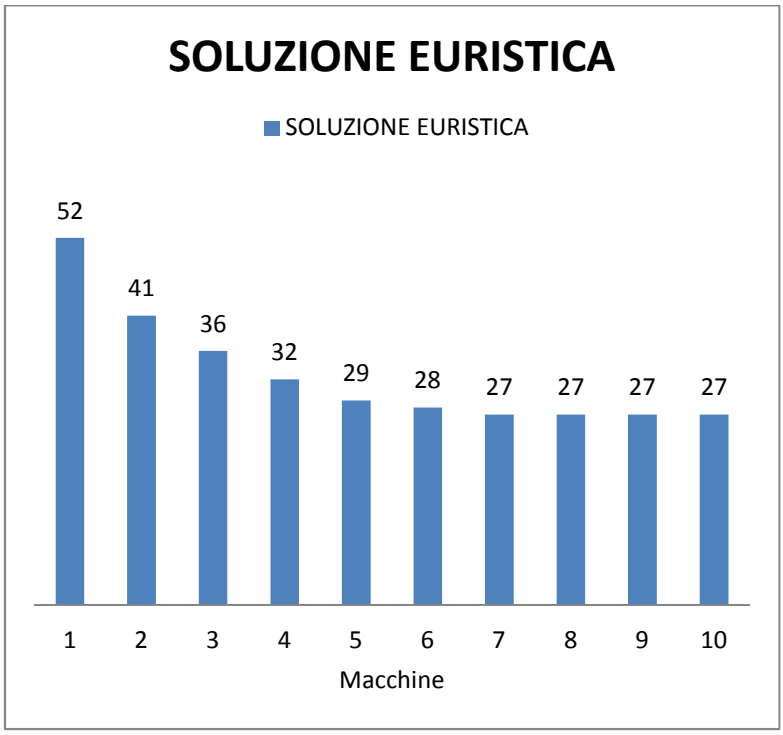
1) La prima istanza considerata è:

NOME ATTIVITA'	DURATA	PRECEDENZE
A	2	-
B	8	-
C	8	-
D	8	-
E	1	C
F	5	-
G	1	-
H	6	A, C, D
I	6	B, E, H
L	7	B, D, F, I

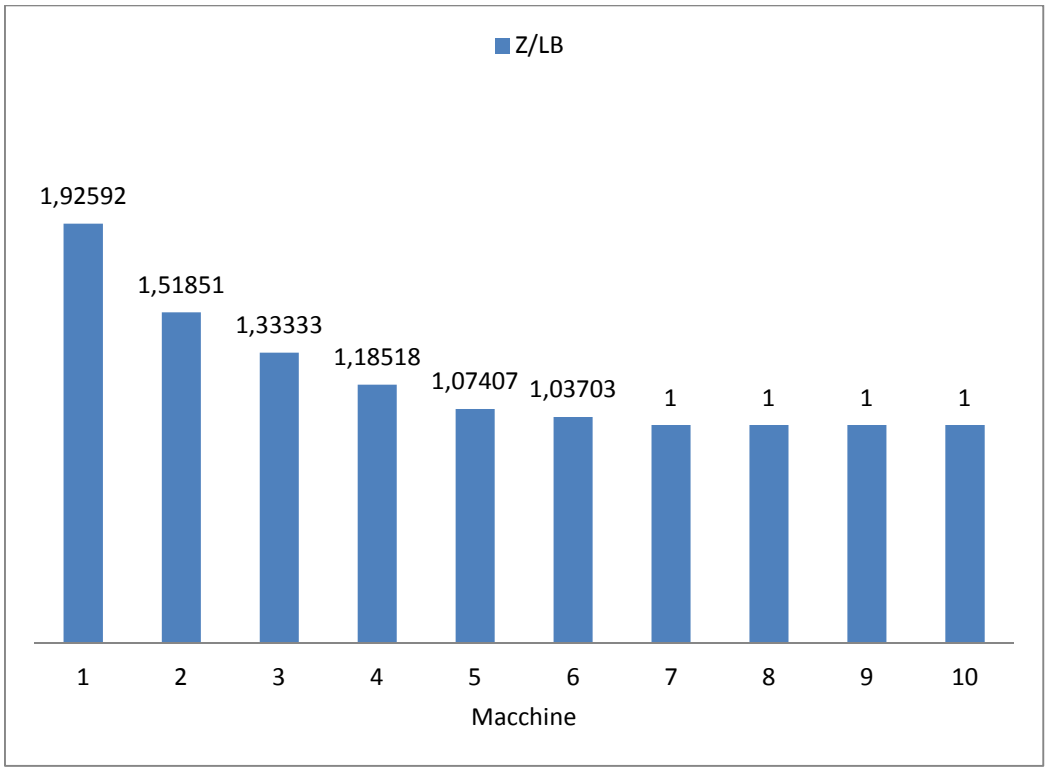
I risultati ottenuti in questo test sono:

NUMERO MACCHINE	SOLUZIONE EURISTICA	LOWER BOUND	Z/LB
1	52	27	1,925925926
2	41	27	1,518518519
3	36	27	1,333333333
4	32	27	1,185185185
5	29	27	1,074074074
6	28	27	1,037037037
7	27	27	1
8	27	27	1
9	27	27	1
10	27	27	1

Questi sono stati ottenuti applicando l'algoritmo descritto in questo paragrafo alla stessa istanza e variando il vincolo del numero di attività parallele ammesse.
Dai risultati ottenuti, come logico, si osserva che la soluzione euristica (makespan) migliora all'aumentare del numero di macchine a disposizione; questo fino a raggiungere l'ottimo, ovvero la migliore soluzione ottenibile, sfruttando 7 macchine.
Già da questo esempio si capisce che non ha senso eccedere con il numero di risorse: sarebbe un investimento inutile. Questo sarà ancora più chiaro vedendo i risultati ottenuti su istanze più grandi. Le soluzioni euristiche ottenute in questo esempio possono essere quindi riassunte nel seguente grafico:

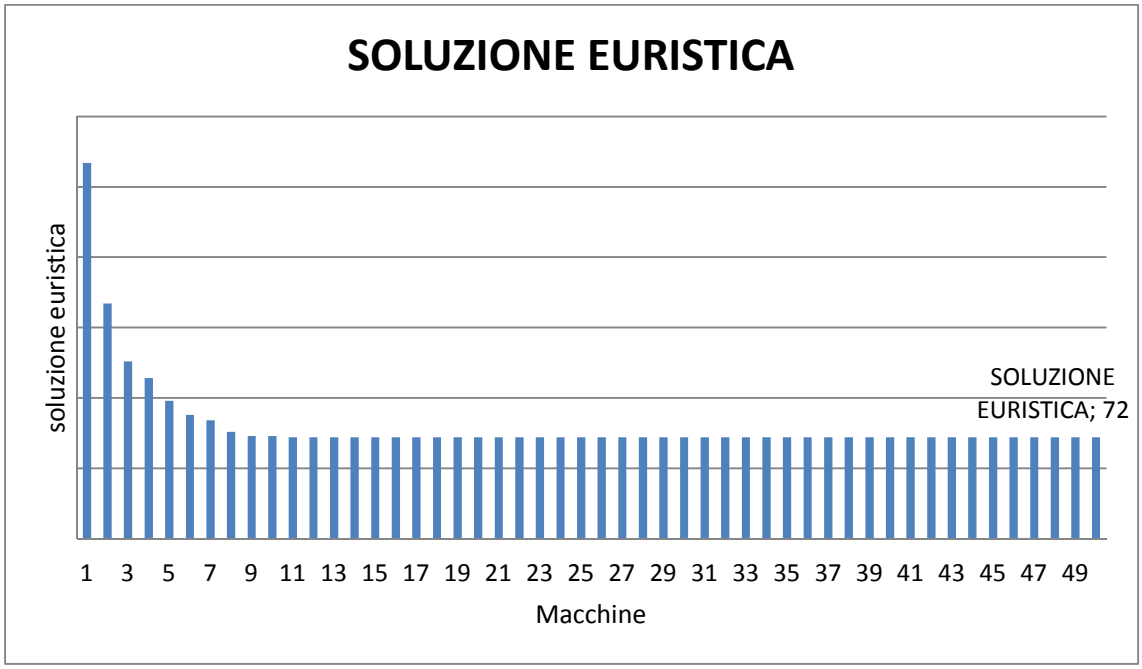


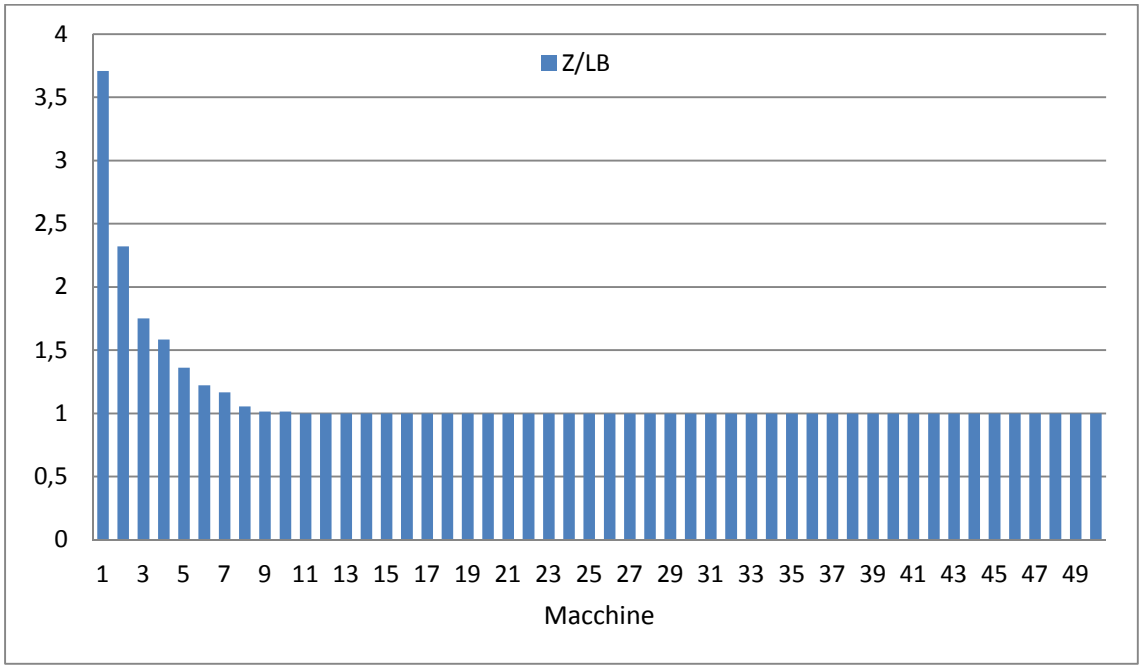
Per valutare, invece, la qualità del makespan ottenuto si studia, come già detto, il rapporto tra la soluzione euristica, ottenuta con m macchine, e il lower bound. Quando questo rapporto vale 1 (Z e LB coincidono) si è ottenuto il migliore risultato possibile.
Tuttavia questo non sempre è la migliore ottimizzazione possibile, infatti a volte, pure terminando il progetto in un tempo leggermente maggiore potrebbe essere possibile rispettare ugualmente i vincoli richiesti dal committente limitando le spese che è necessario sostenere. Il seguente grafico, sempre relativo all'esempio in questione, è un buon indice per valutare vantaggi e svantaggi:



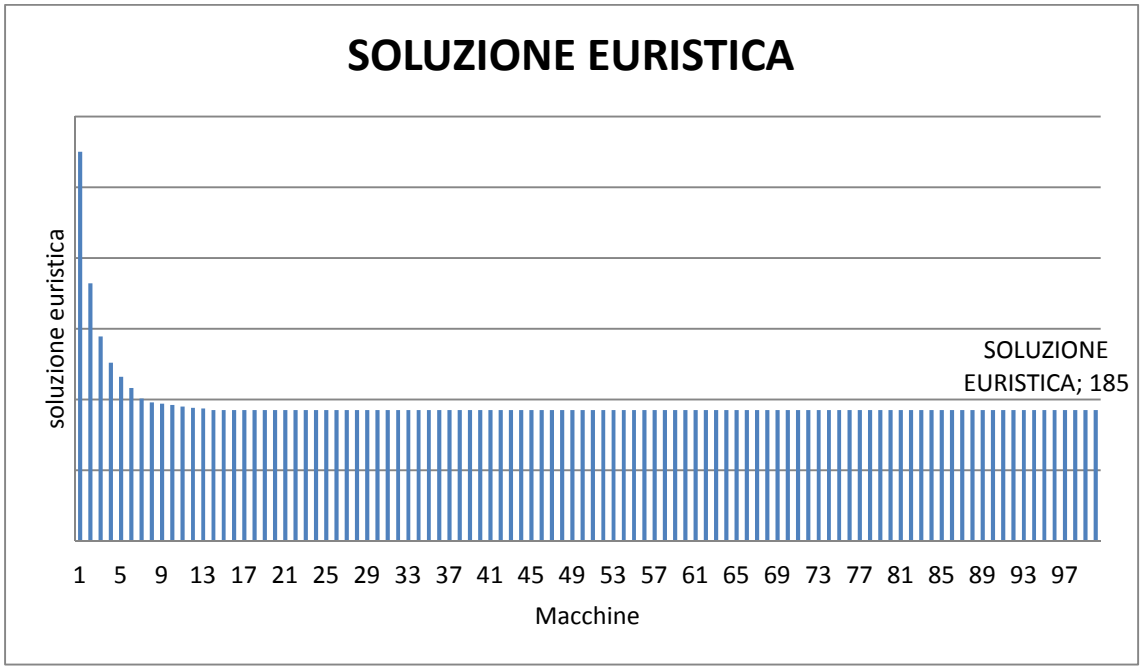
Di seguito vengono riportati unicamente i risultati ottenuti testando l'algoritmo su istanze di 50, 100 e 200 attività, sempre variando il numero di lavori paralleli ammessi.
Per ognuno di questi tre esempi vengono riportati: il grafico relativo alla soluzione euristica ottenuta e il grafico relativo al rapporto Z/LB.

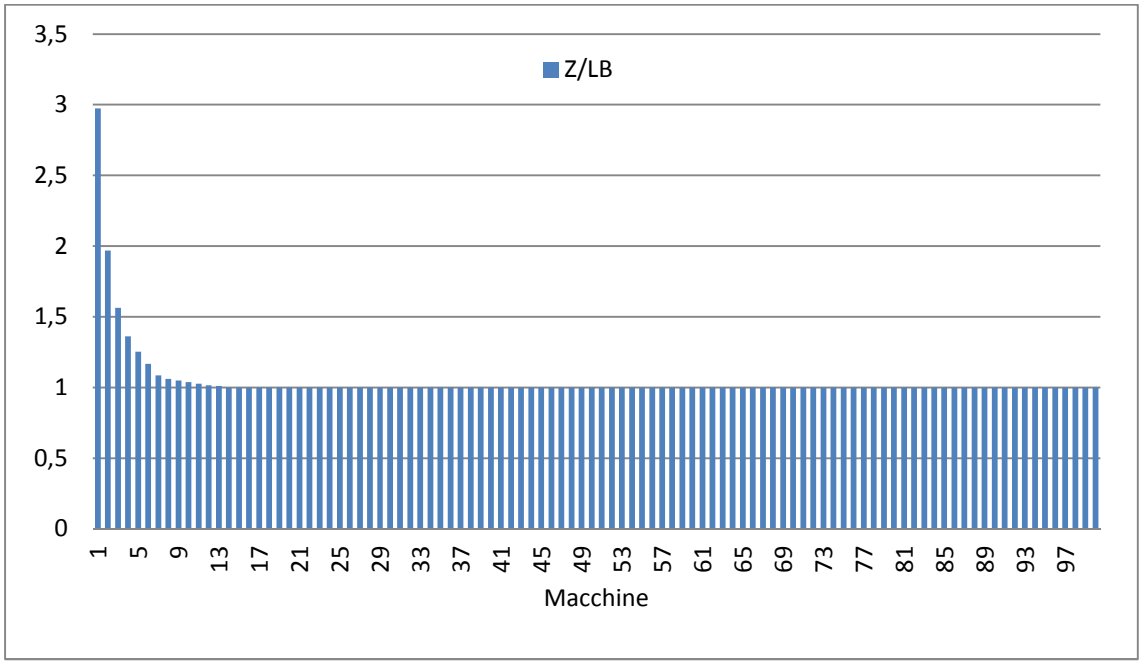
2) Applicando l'algoritmo su un'istanza di 50 attività raggiungo la soluzione ottima avendo a disposizione almeno 11 macchine.



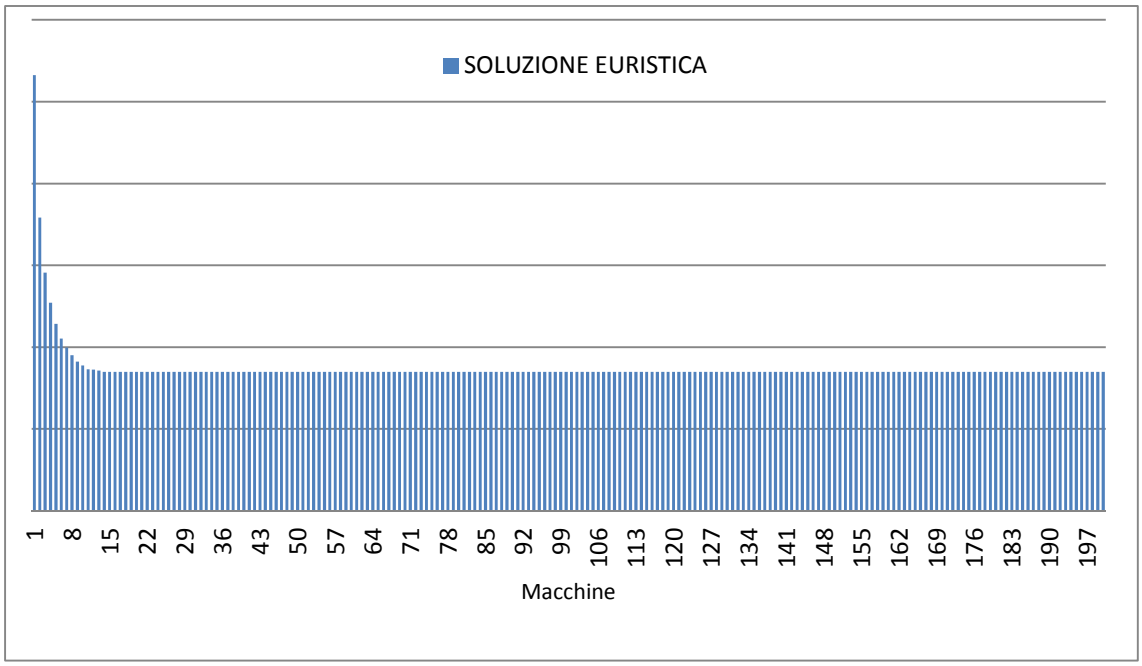


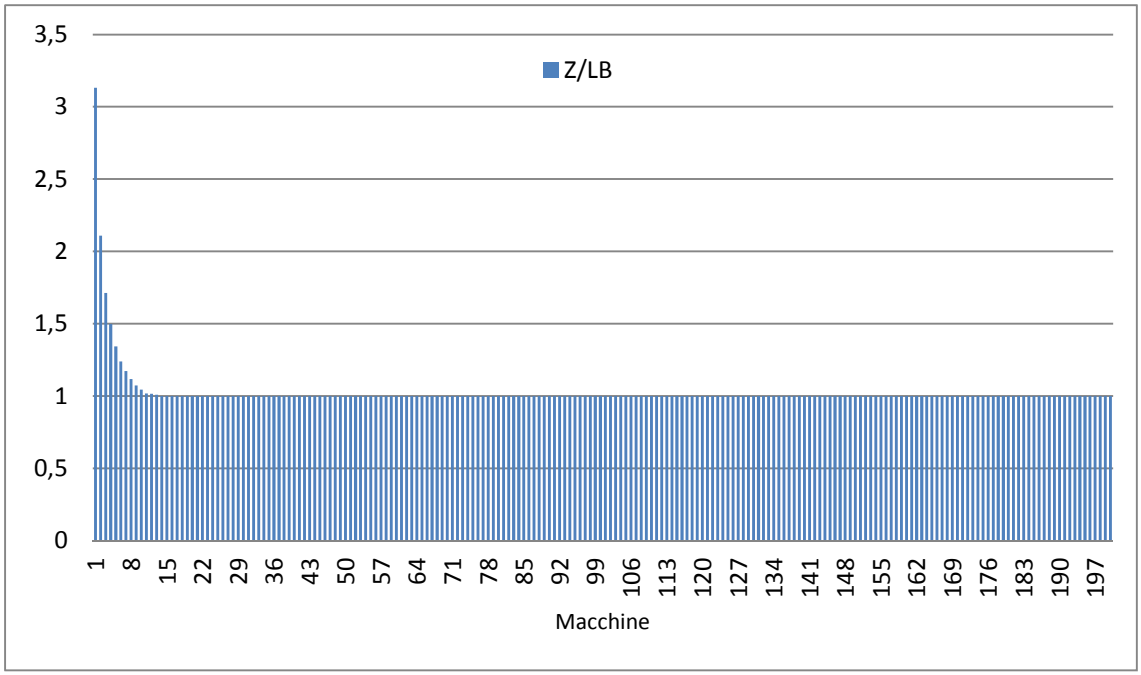
3) Applicando l'algoritmo su un'istanza di 100 attività raggiungo la soluzione ottima avendo a disposizione almeno 14 macchine.





4) Applicando l'algoritmo su un'istanza di 200 attività raggiungo la soluzione ottima avendo a disposizione almeno 15 macchine.

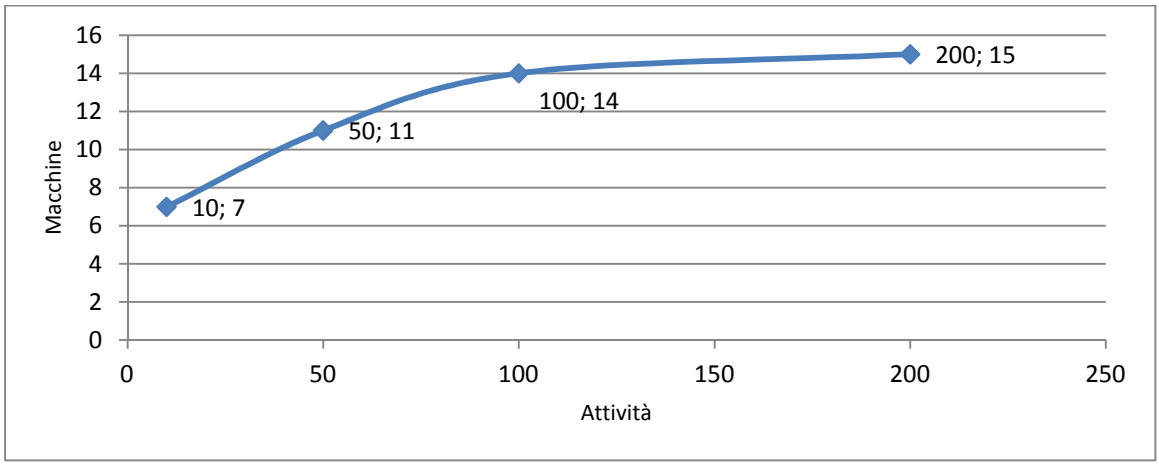




Una prima osservazione da fare sui risultati ottenuti riguarda il numero minimo di macchine necessarie per ottenere la migliore ottimizzazione possibile. Ricordando quanto ottenuto dagli esempi:

1) Per un'istanza di 10 attività sono state necessarie 7 macchine;
2) Per un'istanza di 50 attività sono state necessarie 11 macchine;
3) Per un'istanza di 100 attività sono state necessarie 14 macchine;
4) Per un'istanza di 200 attività sono state necessarie 15 macchine;

riassumendo il tutto in un grafico si ottiene:



Naturalmente questo schema è molto approssimativo, ma dimostra che all'aumentare del numero di attività che compongono un'istanza, il numero di macchine necessarie per raggiungere il migliore makespan possibile tende ad un valore limite. Il grafico sopra riportato è riferito ad un'istanza di prova con probabilità di precedenza tra i lavori che la compongono pari a *Pmin=0,2 (poche precedenze).*

Testando l'algoritmo su più problemi si è osservato che all'aumentare del valore *Pmin,* quindi all'aumentare della probabilità delle precedenze, il tempo di salita della curva ottenuta diminuisce notevolmente come pure il numero di macchine necessario per concludere il progetto nel minore tempo possibile.

3.6 PRESTAZIONI DEL SECONDO ALGORITMO AL VARIARE DEL NUMERO DI RUN

Si è testato l'algoritmo su delle istanze di prova variando il numero di run, ovvero ripetendo la tecnica vista nel paragrafo 3.2 un numero r di volte.
L'istanza di prova considerata è:

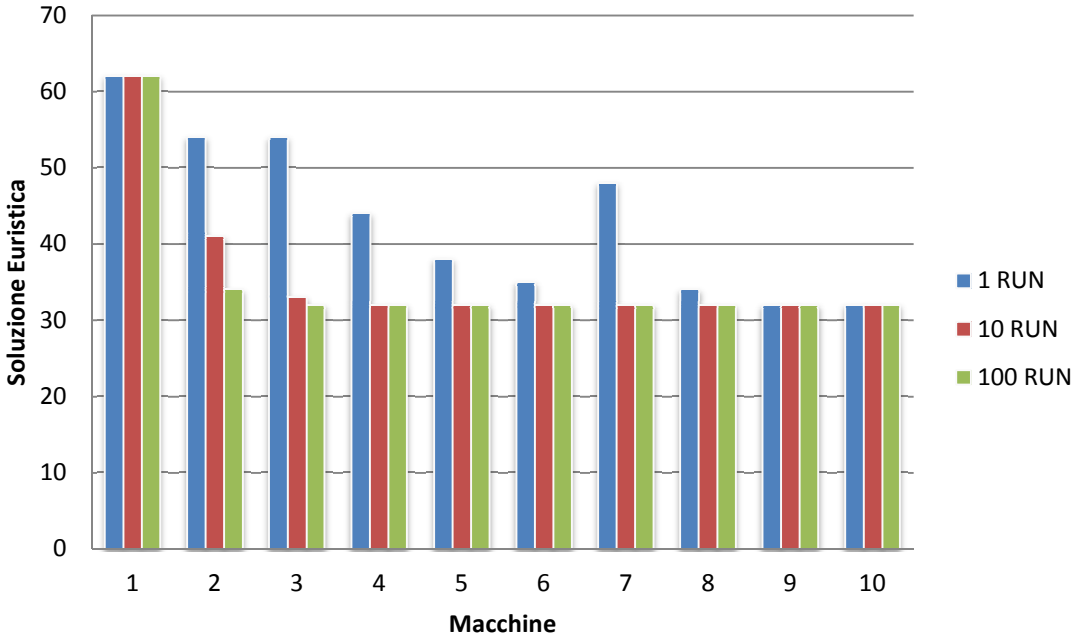
NOME ATTIVITA'	DURATA	PRECEDENZE
A	2	-
B	6	-
C	10	-
D	5	-
E	1	-
F	1	B, C, D
G	10	A, D, E
H	10	B, C
I	8	C, G
L	9	G, I

Questa è caratterizzata da $Pmin = 0{,}2$, ovvero da poche precedenze.
Su tale istanza si è variato sia il numero di run ponendolo pari a 1, 10 e 100; sia il numero di macchine a disposizione $m = 1,\ldots,10$. Questo ha permesso di vedere le prestazioni dell'algoritmo anche in rapporto al vincolo sul numero massimo di attività parallele ammesso.
I risultati ottenuti sono riportati nelle seguenti tabelle:

- Soluzione euristica (makespan):

NUMERO MACCHINE	1 RUN	10 RUN	100 RUN
1	62	62	62
2	54	41	34
3	54	33	32
4	44	32	32
5	38	32	32
6	35	32	32
7	48	32	32
8	34	32	32
9	32	32	32
10	32	32	32

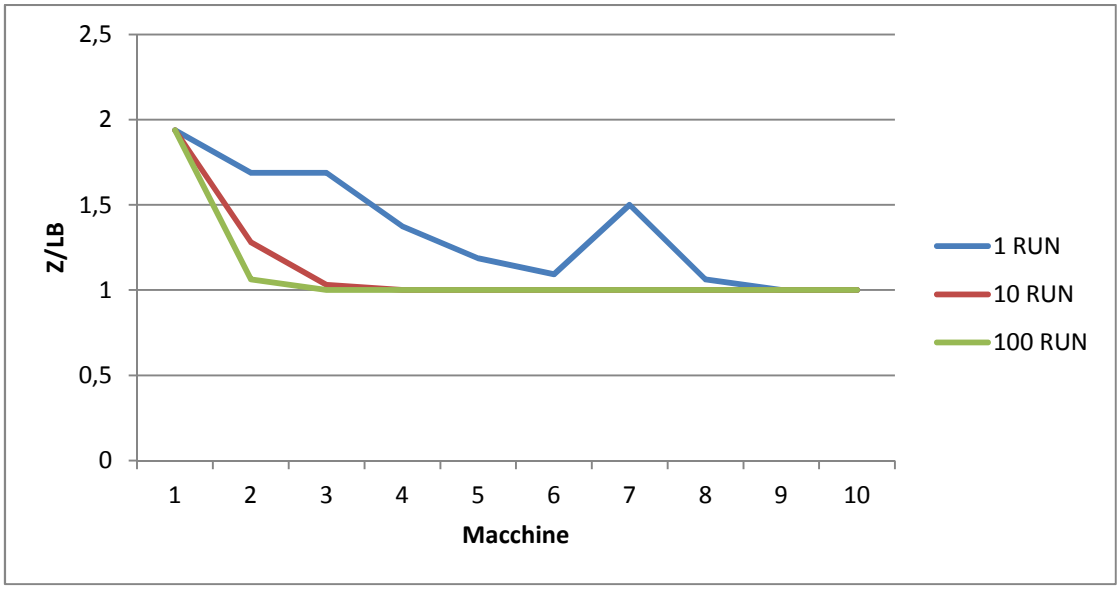
Nel seguente grafico sono riportate le soluzioni euristiche ottenute eseguendo 1, 10 e 100 run:



Come si era previsto l'algoritmo non risulta affidabile eseguendo un'unica run, ad esempio nel caso di 7 macchine è stato calcolato un pessimo makespan. Questo è dovuto al fatto che un organizzazione casuale delle attività può facilmente fallire.
Tuttavia i risultati ottenuti migliorano notevolmente all'aumentare del numero di run.
Ciò si può facilmente vedere studiando il rapporto tra la soluzione trovata e il rispettivo lower bound.

- Rapporto tra la soluzione euristica e il lower bound (Z/LB):

NUMERO MACCHINE	1 RUN	10 RUN	100 RUN
1	1,9375	1,9375	1,9375
2	1,6875	1,28125	1,0625
3	1,6875	1,03125	1
4	1,375	1	1
5	1,1875	1	1
6	1,09375	1	1
7	1,5	1	1
8	1,0625	1	1
9	1	1	1
10	1	1	1



Un buon indice per valutare la qualità della soluzione euristica ottenuta con m macchine, è dato dal rapporto tra la soluzione stessa e il lower bound (*paragrafo 2*).
Da tale grafico è possibile vedere che la curva tende più velocemente all'ottimo, ovvero al valore 1, all'aumentare del numero di run. In altre parole il makespan calcolato migliora notevolmente a parità del numero di macchine a disposizione: questo è molto chiaro se si osservano i risultati ottenuti imponendo il vincolo di massimo 2 attività parallele.
Testando l'algoritmo su istanze composte da un maggiore numero di attività, sempre caratterizzate da un valore $Pmin = 0{,}2$, sono stati ottenuti i medesimi risultati.

Si consideri invece la seguente istanza caratterizzata da $Pmin = 0{,}5$:

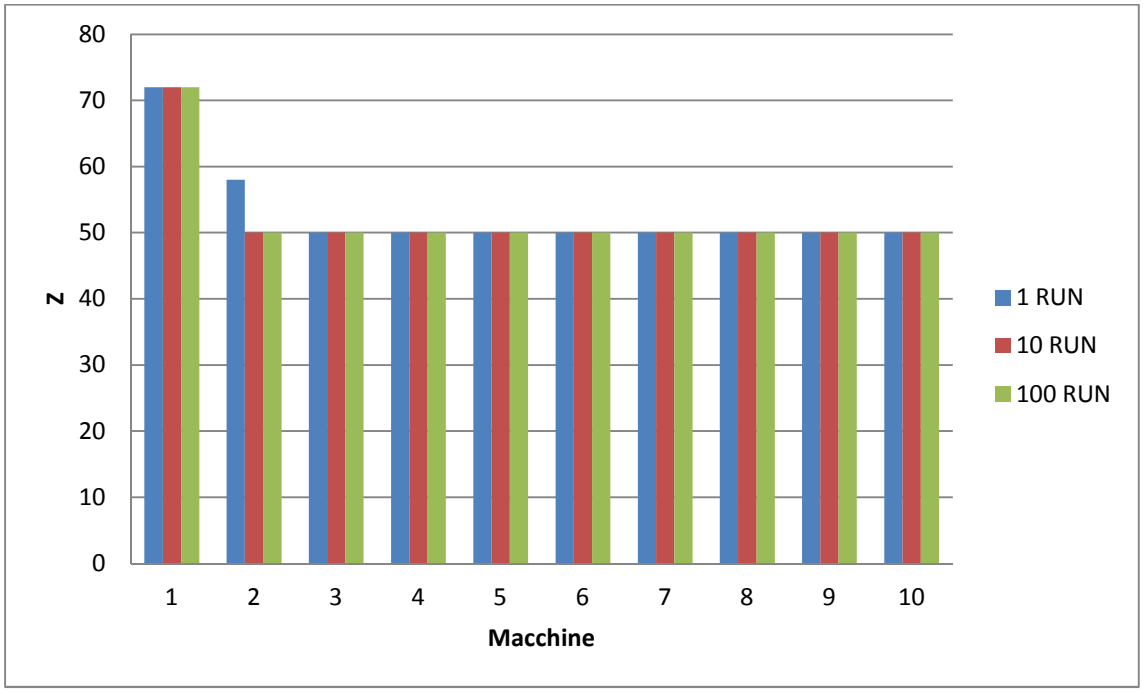
NOME ATTIVITA'	DURATA	PRECEDENZE
A	9	-
B	7	-
C	6	A, B
D	9	A
E	9	A, B, C, D
F	9	A, D, E
G	7	A, C, E
H	6	A, B, C, G
I	1	C, D, G, H
L	9	A, B, C, E, G, H, I

Eseguendo gli stessi test fatti per l'istanza precedente si ottengono i risultati:

- Soluzione euristica (makespan):

NUMERO MACCHINE	1 RUN	10 RUN	100 RUN
1	72	72	72
2	58	50	50
3	50	50	50
4	50	50	50
5	50	50	50
6	50	50	50
7	50	50	50
8	50	50	50
9	50	50	50
10	50	50	50

Aumentando, quindi, la probabilità di precedenza tra le attività è sufficiente un numero minore di run per ottenere buoni risultati, ad esempio in questa istanza eseguendo 10 o 100 run si ottiene la stessa soluzione euristica. Questa considerazione è molto importante per i tempi di calcolo dell'algoritmo: riducendo il numero di run diminuiscono i tempi di calcolo, ottenendo quindi un algoritmo più veloce.
Nel seguente grafico quanto appena detto è ben visibile:



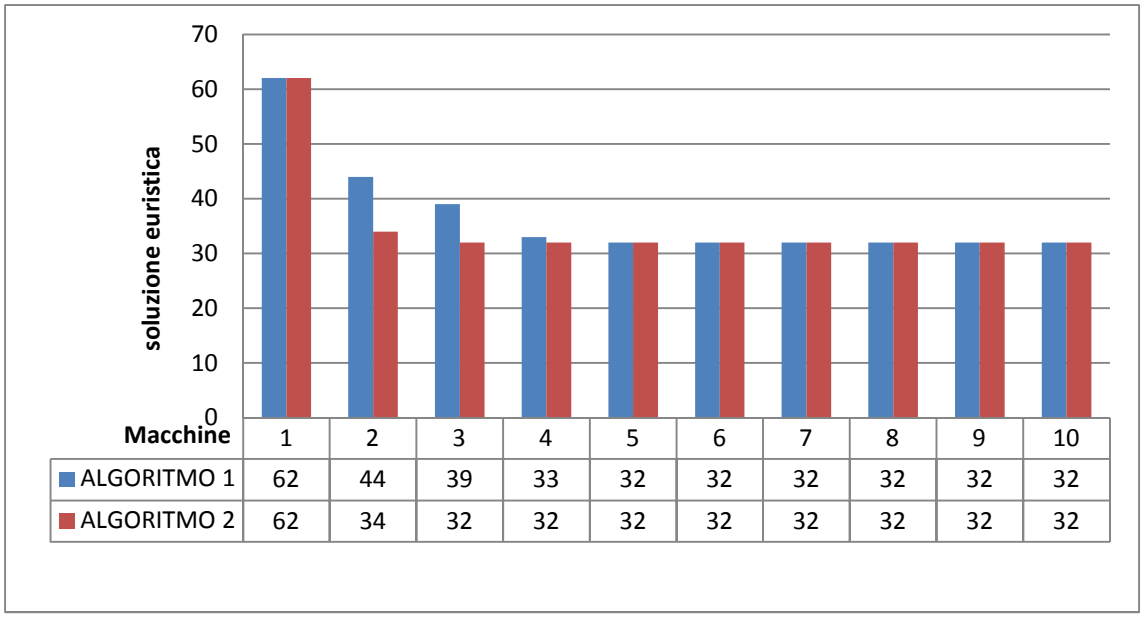
3.7 CONFRONTO TRA I 2 ALGORITMI

Si sono testati i due algoritmi sulle delle stesse istanze di prova variando il vincolo sul numero massimo di attività parallele ammesse. Questo ha permesso di confrontare le tecniche, individuando le circostanze in cui una risulta preferibile all'altra.

Viene ripresa l'istanza studiata nel paragrafo 3.6:

NOME ATTIVITA'	DURATA	PRECEDENZE
A	2	-
B	6	-
C	10	-
D	5	-
E	1	-
F	1	B, C, D
G	10	A, D, E
H	10	B, C
I	8	C, G
L	9	G, I

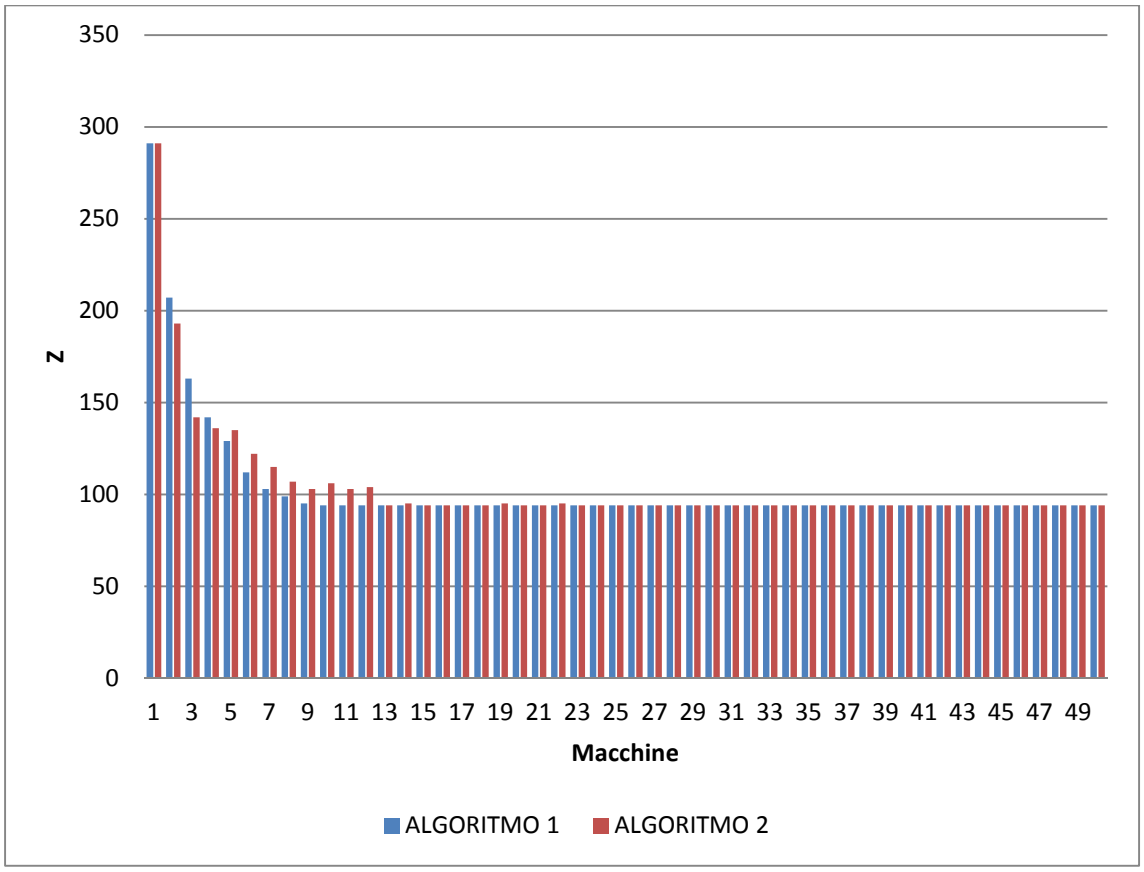
Il makespan calcolato dal primo e dal secondo algoritmo sono riportati nella seguente tabella. Per il secondo algoritmo sono state eseguite 100 run.



Macchine	1	2	3	4	5	6	7	8	9	10
ALGORITMO 1	62	44	39	33	32	32	32	32	32	32
ALGORITMO 2	62	34	32	32	32	32	32	32	32	32

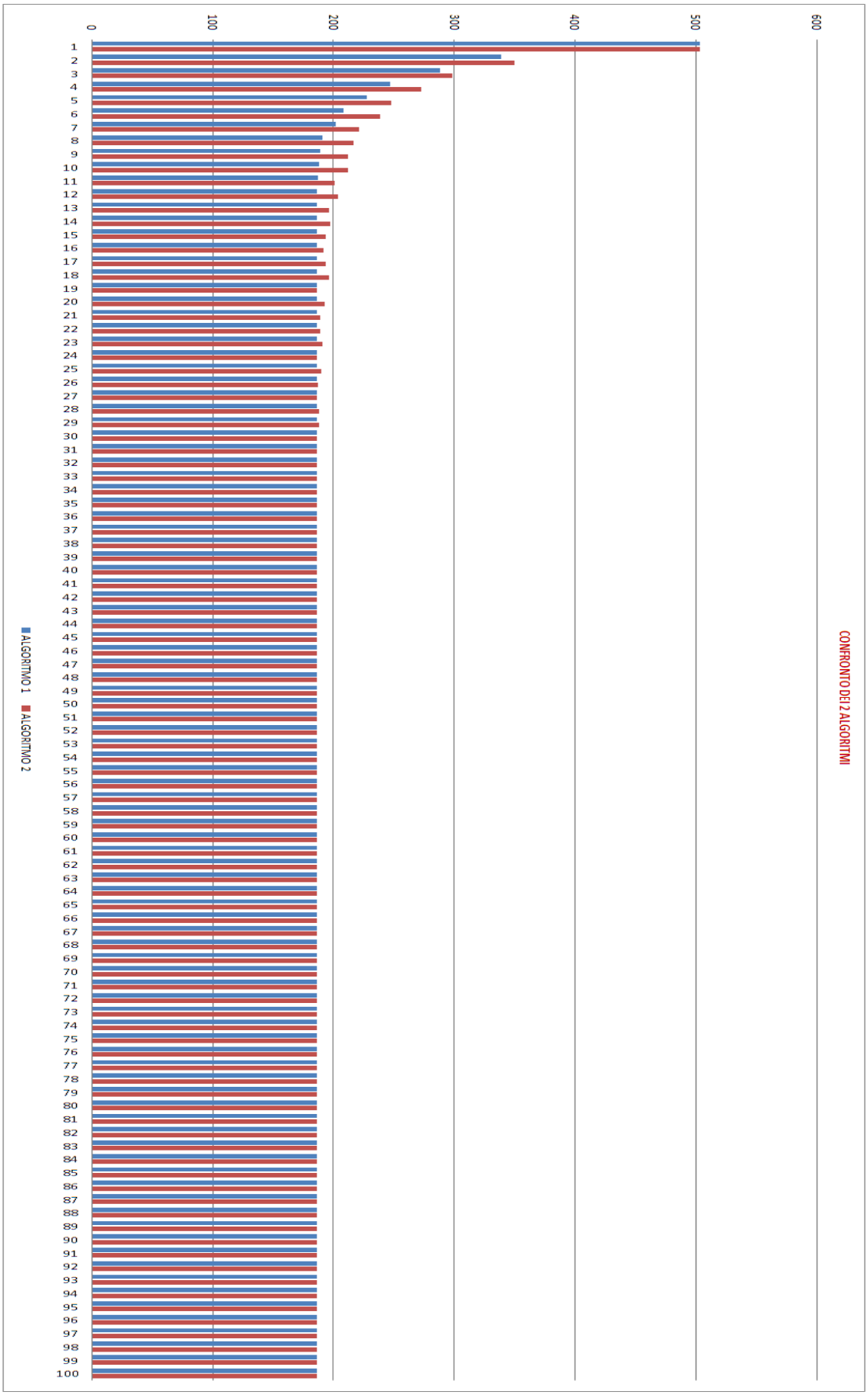
Si può concludere, osservando i risultati ottenuti, che il secondo algoritmo realizzato ha permesso di calcolare un migliore makespan rispetto al primo algoritmo. In altre parole la seconda tecnica ha realizzato una migliore organizzazioni dei lavori, permettendo di terminare il progetto in un tempo minore. Tali conclusioni sono state confermate da più test eseguiti su istanze di piccole dimensioni.

Considerando invece istanze composte da un maggior numero di attività sono stati riscontrati risultati diversi. Ad esempio considerando un'istanza di 50 job, si sono ottenuti i seguenti risultati:



Si nota che il secondo algoritmo risulta migliore rispetto al primo considerando un numero di macchine disponibili minore di 5; per valori maggiori è invece migliore la soluzione euristica calcolata applicando la prima tecnica.

Testando entrambi gli algoritmi su istanze ancora più grandi si osserva che è invece il primo algoritmo quello che permette di ottenere una migliore organizzazione delle attività.
Di seguito sono riportati i risultati ottenuti su un'istanza composta da 100 job:


0
100
200
300
400
500
600
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
ALGORITMO 1
ALGORITMO 2
CONFRONTO DEI 2 ALGORITMI

Si può quindi concludere che il primo algoritmo realizzato permette di calcolare migliori soluzioni su istanze composte da un numero grande di attività; Indicativamente è consigliato su istanze composte da più di 50 job. È invece il secondo algoritmo ad avere migliori prestazioni sulle istanze più piccole, permettendo di realizzare una migliore gestione del progetto.

4.

CONCLUSIONI

I test effettuati, ricapitolando, hanno permesso di concludere che entrambi gli algoritmi sviluppati sono in grado di ottenere soddisfacenti risultati sia per quanto riguarda il calcolo del makespan, sia per quanto riguarda il tempo di calcolo. In particolare il primo algoritmo è risultato più soddisfacente se applicato su istanze composte da molte attività; viceversa il secondo ha permesso di ottenere risultati migliori sulle istanze più piccole. Si può quindi pensare di implementare un unico algoritmo che pianifichi un progetto utilizzando entrambe le tecniche sviluppate, confrontando e scegliendo il minore makespan tra i due ottenuti. In questo modo è possibile ottenere su ogni problema il migliore risultato: la migliore pianificazione dei lavori. Si è stimato che il programma risultante avrà tempo di calcolo dell'ordine del microsecondo, in grado quindi di fornire la soluzione prontamente anche su problemi grandi. Un'altra soluzione potrebbe essere quella di valutare e scegliere quale delle due tecniche utilizzare in base al numero di attività che compongono il progetto. Dai test effettuati si è concluso che il secondo algoritmo si deve applicare su progetti composti da meno di 50 lavori; invece sui progetti più grandi offre migliori prestazioni il primo algoritmo. Il programma così realizzato ha il vantaggio di diminuire i tempi di calcolo a discapito di non garantire che la soluzione fornita sia la migliore ottenibile utilizzando le due tecniche sviluppate.

infine, grazie al vincolo sul numero massimo di attività parallele ammesse, il programma realizzato risulta applicabile e in grado di risolvere alcuni dei problemi di organizzazione aziendale che le aziende devono affrontare quotidianamente.

5.

SITOGRAFIA

[1] http://venus.unive.it/pesenti/Old/docwww/Didattica/Logistica/Log12sch.pdf

[2] http://www.federica.unina.it/ingegneria/ricerca-operativa-ing-2/Scheduling-macchina-singola-parallele/

[3] http://www.disp.uniroma2.it/user/giordani/didattica/MGSC/Dispense/4.SchMaccParallele.pdf

[4] http://www.dii.unisi.it/~agnetis/scheduling.pdf

[5] http://www.dmi.unisa.it/people/cerulli/www/OttimizzazionePages/OttFiles/Scheduling.pdf

[6] http://www.or.uni-bonn.de/lectures/ss10/scheduling_data/sched10_4.pdf

[7] http://www.waset.org/journals/waset/v59/v59-46.pdf

BIBIOGRAFIA

[8] Lorenzo Brunetta, Ricerca Operativa, De Agostini Scuola SpA , Novara, 2008.

[9] Matteo Fischetti, Lezioni di Ricerca Operativa, II edizione, Libreria Progetto, Padova, 1999.

[10] Cay Horstmann, Concetti di informatica e fondamenti di java, APOGEO, 2007.

6.

ALLEGATI

6.1 IMPLEMENTAZIONE PRIMO ALGORITMO IN JAVA

CLASSE LAVORO2

```
public class lavoro2
{
  private int numero_lavoro;
  private String nome;
  private int durata;
  private int[] precedenze1;
  private int numero_precedenze_1;
  private int[] precedenze2;
  private int numero_precedenze_2;
  private int Tmin;
  private int Tfine;

  public lavoro2(int a, int b)
  {
    numero_lavoro = a;
    durata = b;
    nome = "lavoro" + a;
    numero_precedenze_1 = 0;
    precedenze1 = new int[1];
    numero_precedenze_2 = 0;
    precedenze2 = new int[1];
    Tmin = 0;
    Tfine = 0;
  }

  //METODO CHE MI GENERA CASUALMENTE LE PRECEDENZE DEL LAVORO
  //c = probabilità di precedenza
  public void genera_precedenze(double c)
  {
    for(int i=0; i<numero_lavoro-1; i++)
    {
      double Pmin = Math.random();
      //il valore c indica la probabilità di precedenza
      if(Pmin < c)
      {
        precedenze1[numero_precedenze_1] = i+1;
        precedenze2[numero_precedenze_2] = i+1;
        numero_precedenze_1++;
```

```
        numero_precedenze_2++;

        //se serve ingrandisco l'array
        if(numero_precedenze_1 == precedenze1.length)
        {
          int[] tmp_1 = new int[precedenze1.length * 2];
          int[] tmp_2 = new int[precedenze2.length * 2];

          for(int j=0; j<numero_precedenze_1; j++)
          {
            tmp_1[j] = precedenze1[j];
            tmp_2[j] = precedenze2[j];
          }

          precedenze1 = tmp_1;
          precedenze2 = tmp_2;
        }
      }

    }
  }


  public String get_nome()
  {
    return nome;
  }


  public int get_numero_lavoro()
  {
    return numero_lavoro;
  }


  public int get_durata()
  {
    return durata;
  }


  //METODO CHE MI RESTITUISCE UNA STRINGA CONTENENTE TUTTE LE PRECEDENZE DEL
LAVORO
  public String get_precedenze()
  {
    String prec = "\r\n  ";
    int contatore = 0;
    for(int i=0; i< numero_precedenze_1; i++)
    {
      contatore++;
      //posso variare questo numero a seconda di quando vogli andare a capo
      if(contatore == 20)
      {
        prec = prec + " \r\n " + precedenze1[i] + "  ";
        contatore=0;
        continue;
      }
```

```
    prec = prec + precedenze1[i] + "  ";
  }
  return prec;
}


//metodo che mi cancelli una precedenza e aggiorni il numero di precedenze
public void cancello_precedenza(int a)
{
  int k = numero_precedenze_1;
  for(int i=0; i<k; i++)
  {
    if(precedenze2[i] == 0)
    {continue;}

    if(precedenze2[i] == a)
    {
      precedenze2[i] = 0;
      numero_precedenze_2 = numero_precedenze_2 - 1;

      break;
    }

  }
}

public int get_numero_precedenze_2()
{
  return numero_precedenze_2;
}


//metodo che mi restiruisce il numero di precedenze1
public int get_numero_precedenze_1()
{
  return numero_precedenze_1;
}

public int get_precedenza_i(int i)
{
  return precedenze1[i];
}


public int get_Tmin()
{
  return Tmin + 1;
}

//setto il giorno di inizio lavoro
public void set_Tmin(int tm)
{
  Tmin = tm;
}

//calcola il giorno in cui finisce il lavoro
public void set_Tfine()
```

```
  {
    Tfine = durata + Tmin;
  }

  //restituisce il giorno in cui è finito il lavoro
  public int get_Tfine()
  {
    return Tfine;
  }
}
```

CLASSE MACHINA2

```
public class macchina2
{
  private lavoro2[] lavori_svolti;
  private int numero_lavori_svolti;
  private int T_fine;

  public macchina2()
  {
    lavori_svolti = new lavoro2[1];
    numero_lavori_svolti = 0;
    T_fine = 0;
  }

  public void nuovo_lavoro_LB(lavoro2 n)
  {
    lavori_svolti[numero_lavori_svolti] = n;
    numero_lavori_svolti++;
    T_fine = T_fine + n.get_durata();

    if(numero_lavori_svolti == lavori_svolti.length)
    {
      lavoro2[] tmp = new lavoro2[lavori_svolti.length *2];
      for(int i=0; i<numero_lavori_svolti; i++)
      {
        tmp[i] = lavori_svolti[i];
      }
      lavori_svolti = tmp;

    }
  }

  //metodo che mi aggiunga un nuovo lavoro da fare
  public void nuovo_lavoro(lavoro2 n)
  {
    lavori_svolti[numero_lavori_svolti] = n;
    numero_lavori_svolti++;
```

```
    T_fine = n.get_Tfine();

    if(numero_lavori_svolti == lavori_svolti.length)
    {
      lavoro2[] tmp = new lavoro2[lavori_svolti.length *2];
      for(int i=0; i<numero_lavori_svolti; i++)
      {
        tmp[i] = lavori_svolti[i];
      }
      lavori_svolti = tmp;

    }
  }

  //metodo che mi restituisca il giorno di fine lavori
  public int giorno_fine_lavori()
  {
    return T_fine;
  }

  public int lavori_svolti()
  {
    return numero_lavori_svolti;
  }


  public String lavori_fatti_dalla_macchina()
  {
    String att = "";

    for(int i=0; i<numero_lavori_svolti; i++)
    {
      att = att + lavori_svolti[i].get_nome() + ", ";
    }

    return att;
  }


  public String specifiche_di_un_lavoro(int i)
  {
    String spec = "- il " + lavori_svolti[i].get_nome() + " inizia il giorno "
+ lavori_svolti[i].get_Tmin() + " e termina il giorno " +
lavori_svolti[i].get_Tfine();
    return spec;
  }
}
```

CLASSE “MAIN”

```
import java.util.Scanner;
import java.io.*;
```

```
public class programma2
{
  public static void main(String[] args) throws IOException
  {
    Scanner in = new Scanner(System.in);


    int durata_massima= 10;

    System.out.println("INSERIRE IL NOME DEL FILE DOVE VERRANNO SALVATI I DATI
DEL TEST: file_nome");
    String nome_file = in.next();
    nome_file = nome_file + ".txt";

    PrintWriter out = null;
    try
    {
      FileWriter file = new FileWriter(nome_file);
      out = new PrintWriter(file);
    }
    catch (IOException e)
    {
      System.out.println("Errore: " + e);
      System.exit(1);
    }


    System.out.println("INSERIRE IL NUMERO DI LAVORI CHE DESIDERA:");
    out.println("NUMERO DI LAVORI: ");
    int numero_lavori = in.nextInt();
    out.print(numero_lavori);
    System.out.println("");
    out.println("");

out.println("_____________________________________________________________________");
    System.out.println("INSERIRE LA PROBABILITA' DI PRECEDENZA (numero
compreso tra [0,1]):");
    out.println("PROBABILITA' DI PRECEDENZA (Pmin): ");
    double Pmin = in.nextDouble();
    out.print(Pmin);
    out.println("");

out.println("_____________________________________________________________________");
    System.out.println("INSERIRE IL NUMERO DI MACCHINE:");
    out.println("NUMERO DI MACCHINE: ");
    int numero_macchine = in.nextInt();
    out.print(numero_macchine);
    out.println("");

out.println("_____________________________________________________________________");

    int[] soluzione_euristica = new int[2];
    int contat = 0;

    long Tempo1;
    long Tempo2;
    long Tempo;
```

```
    // ritorna un long che e' il tempo (in millisecondi)
    // fra l'ora attuale e la mezzanotte del 1 gennaio 1970
    Tempo1=System.currentTimeMillis();



    lavoro2[] ATTIVITA = new lavoro2[numero_lavori];

    for(int i=0; i<numero_lavori; i++)
    {
      double d = durata_massima * Math.random();
      int durata = (int) d + 1;
      lavoro2 L = new lavoro2(i+1, durata);

      ATTIVITA[i] = L;

      ATTIVITA[i].genera_precedenze(Pmin);

      out.println("");
      out.println("- IL " + ATTIVITA[i].get_nome() + " HA DURATA " +
ATTIVITA[i].get_durata());
      out.println("LE ATTIVITA' PRECEDENTI SONO: " +
ATTIVITA[i].get_precedenze());
    }

    out.println("");

out.println("_____________________________________________________________________")
;
    //ora ho creato l'array ATTIVITA che mi contiene tutti i lavori con il
loro rispettivo nome,
    //durata e precedenze!!!


    //ORA ORDINO L'ARRAY IN MODO CHE LE ATTIVITA' SIANO DISPOSTE IN ORDINE
CRESCENTE

    for(int i=0; i<numero_lavori-1; i++)
    {
      int minimo = ATTIVITA[i].get_durata();

      for(int y=i+1; y<numero_lavori; y++)
      {
        if(ATTIVITA[y].get_durata() < minimo)
        {
          minimo = ATTIVITA[y].get_durata();
          lavoro2 tpm = ATTIVITA[i];
          ATTIVITA[i] = ATTIVITA[y];
          ATTIVITA[y] = tpm;
        }
      }
    }
    //ora gli elementi sono ordinati in ordine di durata crescente!!!


    out.println("");
```

```
    out.println("VENGONO PRIMA ORDINATE LE ATTIVITA' IN ORDINE CRESCENTE DI
DURATA:");

    for(int i=0; i<numero_lavori; i++)
    {
      out.println(ATTIVITA[i].get_nome() + " ---> " +
ATTIVITA[i].get_durata());
    }
    out.println("");

out.println("_____________________________________________________________________");


    //ORA DEVO ORDINARE L'ARRAY IN MODO CHE SIANO RISPETTATE LE PRECEDENZE!!

    lavoro2[] tpm = new lavoro2[numero_lavori];
    int contatore = 0;

    while(contatore < numero_lavori)
    {
      for(int i=0; i<numero_lavori; i++)
      {
        if(ATTIVITA[i] == null)
        {
          continue;
        }

        if(ATTIVITA[i].get_numero_precedenze_2() == 0)
        {
          tpm[contatore] = ATTIVITA[i];
          contatore++;
          ATTIVITA[i] = null;

          for(int k=0; k<numero_lavori; k++)
          {
            if(ATTIVITA[k] == null)
            {
              continue;
            }
            ATTIVITA[k].cancello_precedenza(tpm[contatore -
1].get_numero_lavoro());
          }

          break;
        }
      }
    }

    ATTIVITA = tpm;



    out.println("");
    out.println("ATTIVITA' ORDINATE IN MODO DA RISPETTARE LE PRECEDENZE:");

    for(int i=0; i<numero_lavori; i++)
    {
```

```
      out.println(ATTIVITA[i].get_nome() + " ---> " +
ATTIVITA[i].get_durata());
    }

    Tempo2=System.currentTimeMillis();

    //tempo in millisecondi
    Tempo=Tempo2-Tempo1;


    long TSecondo = 0;
    long TPrimo = 0;
    long Temp1 = 0;
    long Temp2 = 0;
    long T1 = 0;
    long T2 = 0;


    for(int u=1; u< numero_lavori+1; u++)
    {
      if(u==numero_macchine)
      {
       // ritorna un long che e' il tempo (in millisecondi)
       // fra l'ora attuale e la mezzanotte del 1 gennaio 1970
        Temp1=System.currentTimeMillis();
      }

      if(u==numero_lavori)
      {
        T1=System.currentTimeMillis();
      }

      if(u==numero_macchine || u==numero_lavori)
      {
        numero_macchine = u;

        macchina2[] azienda = new macchina2[numero_macchine];

        for(int i=0; i<numero_macchine; i++)
        {
          macchina2 uno = new macchina2();
          azienda[i] = uno;
        }

        for(int k=0; k< numero_lavori; k++)
        {
          //ora devo verificare il giorno di possibile inizio del nuovo lavoro
          //ed aggiornare T_min e T_fine del lavoro prima di inserirlo nella
macchina
          int fine_lavori_macchina = azienda[0].giorno_fine_lavori();

          if(ATTIVITA[k].get_numero_precedenze_1() == 0)
          {
            ATTIVITA[k].set_Tmin(fine_lavori_macchina);
//metodo nuovo_lavoro
            ATTIVITA[k].set_Tfine();
//mi inserisce l'attivita sulla macchina
```

```
            azienda[0].nuovo_lavoro(ATTIVITA[k]);
//e mi aggiorna il tempo di fine lavori
          }
          else
          {
            for(int j=0; j<ATTIVITA[k].get_numero_precedenze_1(); j++)
            {
              int p = ATTIVITA[k].get_precedenza_i(j);

              for(int y=0; y<numero_lavori; y++)
              {
                if(p == ATTIVITA[y].get_numero_lavoro())
                {
                  int t_fine = ATTIVITA[y].get_Tfine();
                  if(t_fine > fine_lavori_macchina)
                  {
                    fine_lavori_macchina = t_fine;
                    break;
                  }
                }
              }
            }
            ATTIVITA[k].set_Tmin(fine_lavori_macchina);
            ATTIVITA[k].set_Tfine();
            azienda[0].nuovo_lavoro(ATTIVITA[k]);
          }

          //ora devo ordinare le macchine in modo che la prima sia quella con
tempo di fine lavori minore
          int t = azienda[0].giorno_fine_lavori();
          for(int m=1; m<numero_macchine; m++)
          {
            if(azienda[m].giorno_fine_lavori() < t)
            {
              macchina2 tmp = new macchina2();
              tmp = azienda[0];
              azienda[0] = azienda[m];
              azienda[m] = tmp;
            }
          }
        }



        if(u==numero_macchine)
        {
          out.println("");

out.println("_____________________________________________________________________");
          out.println("");
          out.println("LE " + numero_macchine + " MACCHINE TERMINANO I LAVORI
I GIORNI:");
          for(int i=0; i<numero_macchine; i++)
          {
            int x = i+1;
            out.println("");
```

```
            out.println("--> LA MACCHINA " + x + " HA IMPIEGATO " +
azienda[i].giorno_fine_lavori() + " giorni; ");
            out.println("    HA SVOLTO " + azienda[i].lavori_svolti() + "
LAVORI: " + azienda[i].lavori_fatti_dalla_macchina());
            for(int w=0; w<azienda[i].lavori_svolti(); w++)
            {
              out.println("    " + azienda[i].specifiche_di_un_lavoro(w));
            }
          }
        }

        int end = 0;

        for(int m=0; m< numero_macchine; m++)
        {
          if(azienda[m].giorno_fine_lavori() > end)
          {
            end = azienda[m].giorno_fine_lavori();
          }
        }


        soluzione_euristica[contat] = end;


        contat++;


      }

      if(u==numero_macchine)
      {
        Temp2=System.currentTimeMillis();

        //tempo in millisecondi
        TPrimo=Temp2-Temp1;
      }

      if(u==numero_lavori)
      {
        T2=System.currentTimeMillis();

        //tempo in millisecondi
        TSecondo=T2-T1;
      }


    }

      out.println("");
      out.println("");
      out.println("i valori hanno il seguente significato:");
      out.println("- 1 ----> soluzione euristica");
      out.println("- 2 ----> lower bound");
      out.println("- 3 ----> z/LB");
      out.println("- 4 ----> tempo di calcolo LB");
      out.println("- 5 ----> tempo di calcolo soluzione euristica");
```

```
      out.println("");
      out.println("");

      out.println(soluzione_euristica[0]);
      out.println(soluzione_euristica[1]);
      out.println(soluzione_euristica[0]/soluzione_euristica[1]);
      out.println(Tempo + TSecondo);
      out.println(Tempo + TPrimo);


    out.close();
  }

}
```

6.2 IMPLEMENTAZIONE SECONDO ALGORITMO IN JAVA

CLASSE LAVORO2

```
public class lavoro2
{
  private int numero_lavoro;
  private String nome;
  private int durata;
  private int[] precedenze1;
  private int numero_precedenze_1;
  private int[] precedenze2;
  private int numero_precedenze_2;
  private int Tmin;
  private int Tfine;

  public lavoro2(int a, int b)
  {
    numero_lavoro = a;
    durata = b;
    nome = "lavoro" + a;
    numero_precedenze_1 = 0;
    precedenze1 = new int[1];
    numero_precedenze_2 = 0;
    precedenze2 = new int[1];
    Tmin = 0;
    Tfine = 0;
  }

  //METODO CHE MI GENERA CASUALMENTE LE PRECEDENZE DEL LAVORO
  //c = probabilità di precedenza
  public void genera_precedenze(double c)
  {
    for(int i=0; i<numero_lavoro-1; i++)
    {
      double Pmin = Math.random();
      //il valore c indica la probabilità di precedenza
      if(Pmin < c)
      {
        precedenze1[numero_precedenze_1] = i+1;
        precedenze2[numero_precedenze_2] = i+1;
        numero_precedenze_1++;
        numero_precedenze_2++;

        //se serve ingrandisco l'array
        if(numero_precedenze_1 == precedenze1.length)
        {
          int[] tmp_1 = new int[precedenze1.length * 2];
          int[] tmp_2 = new int[precedenze2.length * 2];

          for(int j=0; j<numero_precedenze_1; j++)
          {
            tmp_1[j] = precedenze1[j];
            tmp_2[j] = precedenze2[j];
          }
```

```
          precedenze1 = tmp_1;
          precedenze2 = tmp_2;
        }
      }

    }
  }


  public String get_nome()
  {
    return nome;
  }


  public int get_numero_lavoro()
  {
    return numero_lavoro;
  }


  public int get_durata()
  {
    return durata;
  }


  //METODO CHE MI RESTITUISCE UNA STRINGA CONTENENTE TUTTE LE PRECEDENZE DEL
LAVORO
  public String get_precedenze()
  {
    String prec = "\r\n  ";
    int contatore = 0;
    for(int i=0; i< numero_precedenze_1; i++)
    {
      contatore++;
      //posso variare questo numero a seconda di quando vogli andare a capo
      if(contatore == 20)
      {
        prec = prec + " \r\n " + precedenze1[i] + "  ";
        contatore=0;
        continue;
      }
      prec = prec + precedenze1[i] + "  ";
    }
    return prec;
  }


  //metodo che mi cancelli una precedenza e aggiorni il numero di precedenze
  public void cancello_precedenza(int a)
  {
    int k = numero_precedenze_1;
    for(int i=0; i<k; i++)
    {
      if(precedenze2[i] == 0)
```

```
    {continue;}

    if(precedenze2[i] == a)
    {
      precedenze2[i] = 0;
      numero_precedenze_2 = numero_precedenze_2 - 1;

      break;
    }

  }
}

public int get_numero_precedenze_2()
{
  return numero_precedenze_2;
}


//metodo che mi restiruisce il numero di precedenze1
public int get_numero_precedenze_1()
{
  return numero_precedenze_1;
}

public int get_precedenza_i(int i)
{
  return precedenze1[i];
}


public int get_Tmin()
{
  return Tmin + 1;
}

//setto il giorno di inizio lavoro
public void set_Tmin(int tm)
{
  Tmin = tm;
}

//calcola il giorno in cui finisce il lavoro
public void set_Tfine()
{
  Tfine = durata + Tmin;
}

//restituisce il giorno in cui è finito il lavoro
public int get_Tfine()
{
  return Tfine;
}

public void copio_precedenze(lavoro2 p)
{
  int num = p.get_numero_precedenze_1();
```

```
    numero_precedenze_1 = num;
    numero_precedenze_2 = num;
    precedenze1 = new int[num];
    precedenze2 = new int[num];

    for(int i=0; i<num; i++)
    {
      precedenze1[i]= p.get_precedenza_i(i);
      precedenze2[i]= p.get_precedenza_i(i);
    }
  }
}
```

CLASSE MACCHINA_ALG2

```
public class macchina_alg2
{
  private lavoro2[] lavori_da_svolgere;
  private lavoro2[] lavori_svolti;
  private int numero_lavori;
  private int numero_lavori_svolti;
  private int T_fine;

  public macchina_alg2()
  {
    lavori_da_svolgere = new lavoro2[1];
    numero_lavori = 0;
    lavori_svolti = new lavoro2[1];
    numero_lavori_svolti = 0;
    T_fine = 0;
  }


  //metodo per inserire un nuovo lavoro da svolgere
  public void inserire_lavoro(lavoro2 L)
  {
    lavori_da_svolgere[numero_lavori] = L;
    numero_lavori++;

    if(numero_lavori == lavori_da_svolgere.length)
    {
      lavoro2[] tmp = new lavoro2[lavori_da_svolgere.length *2];
      for(int i=0; i<numero_lavori; i++)
      {
        tmp[i] = lavori_da_svolgere[i];
      }
      lavori_da_svolgere = tmp;

    }
  }


  //metodo per verificare se la macchina ha ancora lavori da fare
```

```
public boolean ha_altri_lavori()
{
  if(numero_lavori == numero_lavori_svolti)
  {
    return false;
  }
  else
  {
    return true;
  }
}


//metodo per vedere se il primo lavoro è quello che devo fare
public boolean faccio_questo(int n)
{
  if(n == lavori_da_svolgere[0].get_numero_lavoro())
  {
    return true;
  }
  else
  {
    return false;
  }
}


//metodo per fare un lavoro
public void fai_lavoro(lavoro2 L)
{
  lavori_svolti[numero_lavori_svolti] = L;
  numero_lavori_svolti++;
  T_fine = L.get_Tfine();  //devo aggiornarlo prima di inserire il lavoro!!

  if(numero_lavori_svolti == lavori_svolti.length)
  {
    lavoro2[] tmp = new lavoro2[lavori_svolti.length *2];
    for(int i=0; i<numero_lavori_svolti; i++)
    {
      tmp[i] = lavori_svolti[i];
    }
    lavori_svolti = tmp;
  }

  lavori_da_svolgere[0] = null;
  int contatore = 1;
  while(true)
  {
    if(lavori_da_svolgere[contatore] == null)
    {
      break;
    }
    else
    {
      lavori_da_svolgere[contatore-1] = lavori_da_svolgere[contatore];
      lavori_da_svolgere[contatore] = null;
      contatore++;
```

```
      }
    }
  }


  //metodo che mi restituisce il giorno di fine lavori
  public int giorno_fine_lavori()
  {
    return T_fine;
  }


  public int lavori_svolti()
  {
    return numero_lavori_svolti;
  }


  public String lavori_fatti_dalla_macchina()
  {
    String att = "";

    for(int i=0; i<numero_lavori_svolti; i++)
    {
      att = att + lavori_svolti[i].get_nome() + ", ";
    }

    return att;
  }


  public String specifiche_di_un_lavoro(int i)
  {
    String spec = "- il " + lavori_svolti[i].get_nome() + " inizia il giorno "
+ lavori_svolti[i].get_Tmin() + " e termina il giorno " +
lavori_svolti[i].get_Tfine();
    return spec;
  }
}
```

CLASSE “MAIN”

```
import java.util.Scanner;
import java.io.*;

public class programma2
{
  public static void main(String[] args) throws IOException
  {
    Scanner in = new Scanner(System.in);


    int durata_massima= 10;
```

```
    System.out.println("INSERIRE IL NOME DEL FILE DOVE VERRANNO SALVATI I DATI
DEL TEST: file_nome");
    String nome_file = in.next();
    nome_file = nome_file + ".txt";

    PrintWriter out = null;
    try
    {
      FileWriter file = new FileWriter(nome_file);
      out = new PrintWriter(file);
    }
    catch (IOException e)
    {
      System.out.println("Errore: " + e);
      System.exit(1);
    }


    System.out.println("INSERIRE IL NUMERO DI LAVORI CHE DESIDERA:");
    out.println("NUMERO DI LAVORI: ");
    int numero_lavori = in.nextInt();
    out.print(numero_lavori);
    System.out.println("");
    out.println("");

out.println("_____________________________________________________________________");
    System.out.println("INSERIRE LA PROBABILITA' DI PRECEDENZA (numero
compreso tra [0,1]):");
    out.println("PROBABILITA' DI PRECEDENZA (Pmin): ");
    double Pmin = in.nextDouble();
    out.print(Pmin);
    out.println("");

out.println("_____________________________________________________________________");
    System.out.println("INSERIRE IL NUMERO DI MACCHINE:");
    out.println("NUMERO DI MACCHINE: ");
    int numero_macchine = in.nextInt();
    int numero_macchine_alg2 = numero_macchine;
    out.print(numero_macchine);
    out.println("");

out.println("_____________________________________________________________________");




    int[] soluzione_euristica = new int[2];
    int contat = 0;

    long Tempo1;
    long Tempo2;
    long Tempo;
    // ritorna un long che e' il tempo (in millisecondi)
    // fra l'ora attuale e la mezzanotte del 1 gennaio 1970
    Tempo1=System.currentTimeMillis();
```

```
    lavoro2[] ATTIVITA = new lavoro2[numero_lavori];
    lavoro2[] ATTIVITA_2 = new lavoro2[numero_lavori];

    for(int i=0; i<numero_lavori; i++)
    {
      double d = durata_massima * Math.random();
      int durata = (int) d + 1;
      lavoro2 L = new lavoro2(i+1, durata);

      ATTIVITA[i] = L;

      ATTIVITA[i].genera_precedenze(Pmin);

      out.println("");
      out.println("- IL " + ATTIVITA[i].get_nome() + " HA DURATA " +
ATTIVITA[i].get_durata());
      out.println("LE ATTIVITA' PRECEDENTI SONO: " +
ATTIVITA[i].get_precedenze());
    }

    for(int i=0; i<numero_lavori; i++)
    {
      lavoro2 K = new lavoro2(ATTIVITA[i].get_numero_lavoro(),
ATTIVITA[i].get_durata());
      ATTIVITA_2[i] = K;
      ATTIVITA_2[i].copio_precedenze(ATTIVITA[i]);
    }

    out.println("");

out.println("_____________________________________________________________________");
    //ora ho creato l'array ATTIVITA che mi contiene tutti i lavori con il
loro rispettivo nome,
    //durata e precedenze!!!


    //ORA ORDINO L'ARRAY IN MODO CHE LE ATTIVITA' SIANO DISPOSTE IN ORDINE
CRESCENTE

    for(int i=0; i<numero_lavori-1; i++)
    {
      int minimo = ATTIVITA[i].get_durata();

      for(int y=i+1; y<numero_lavori; y++)
      {
        if(ATTIVITA[y].get_durata() < minimo)
        {
          minimo = ATTIVITA[y].get_durata();
          lavoro2 tpm = ATTIVITA[i];
          ATTIVITA[i] = ATTIVITA[y];
          ATTIVITA[y] = tpm;
        }
      }
    }
    //ora gli elementi sono ordinati in ordine di durata crescente!!!
```

```
    out.println("");
    out.println("VENGONO PRIMA ORDINATE LE ATTIVITA' IN ORDINE CRESCENTE DI
DURATA:");

    for(int i=0; i<numero_lavori; i++)
    {
      out.println(ATTIVITA[i].get_nome() + " ---> " +
ATTIVITA[i].get_durata());
    }
    out.println("");

out.println("_____________________________________________________________________");


    //ORA DEVO ORDINARE L'ARRAY IN MODO CHE SIANO RISPETTATE LE PRECEDENZE!!

    lavoro2[] tpm = new lavoro2[numero_lavori];
    int contatore = 0;

    while(contatore < numero_lavori)
    {
      for(int i=0; i<numero_lavori; i++)
      {
        if(ATTIVITA[i] == null)
        {
          continue;
        }

        if(ATTIVITA[i].get_numero_precedenze_2() == 0)
        {
          tpm[contatore] = ATTIVITA[i];
          contatore++;
          ATTIVITA[i] = null;

          for(int k=0; k<numero_lavori; k++)
          {
            if(ATTIVITA[k] == null)
            {
              continue;
            }
            ATTIVITA[k].cancello_precedenza(tpm[contatore -
1].get_numero_lavoro());
          }

          break;
        }
      }
    }

    ATTIVITA = tpm;



    out.println("");
    out.println("ATTIVITA' ORDINATE IN MODO DA RISPETTARE LE PRECEDENZE:");
```

```
    for(int i=0; i<numero_lavori; i++)
    {
      out.println(ATTIVITA[i].get_nome() + " ---> " +
ATTIVITA[i].get_durata());
    }

    Tempo2=System.currentTimeMillis();

    //tempo in millisecondi
    Tempo=Tempo2-Tempo1;


//_____________________________________________________________________________
    long TSecondo = 0;
    long TPrimo = 0;
    long Temp1 = 0;
    long Temp2 = 0;
    long T1 = 0;
    long T2 = 0;


    for(int u=1; u< numero_lavori+1; u++)
    {
      if(u==numero_macchine)
      {
       // ritorna un long che e' il tempo (in millisecondi)
       // fra l'ora attuale e la mezzanotte del 1 gennaio 1970
        Temp1=System.currentTimeMillis();
      }

      if(u==numero_lavori)
      {
        T1=System.currentTimeMillis();
      }

      if(u==numero_macchine || u==numero_lavori)
      {
        numero_macchine = u;

        macchina2[] azienda = new macchina2[numero_macchine];

        for(int i=0; i<numero_macchine; i++)
        {
          macchina2 uno = new macchina2();
          azienda[i] = uno;
        }

        for(int k=0; k< numero_lavori; k++)
        {
          //ora devo verificare il giorno di possibile inizio del nuovo lavoro
          //ed aggiornare T_min e T_fine del lavoro prima di inserirlo nella
macchina
          int fine_lavori_macchina = azienda[0].giorno_fine_lavori();

          if(ATTIVITA[k].get_numero_precedenze_1() == 0)
          {
```

```
            ATTIVITA[k].set_Tmin(fine_lavori_macchina);
//metodo nuovo_lavoro
            ATTIVITA[k].set_Tfine();
//mi inserisce l'attivita sulla macchina
            azienda[0].nuovo_lavoro(ATTIVITA[k]);
//e mi aggiorna il tempo di fine lavori
          }
          else
          {
            for(int j=0; j<ATTIVITA[k].get_numero_precedenze_1(); j++)
            {
              int p = ATTIVITA[k].get_precedenza_i(j);

              for(int y=0; y<numero_lavori; y++)
              {
                if(p == ATTIVITA[y].get_numero_lavoro())
                {
                  int t_fine = ATTIVITA[y].get_Tfine();
                  if(t_fine > fine_lavori_macchina)
                  {
                    fine_lavori_macchina = t_fine;
                    break;
                  }
                }
              }
            }
            ATTIVITA[k].set_Tmin(fine_lavori_macchina);
            ATTIVITA[k].set_Tfine();
            azienda[0].nuovo_lavoro(ATTIVITA[k]);
          }

          //ora devo ordinare le macchine in modo che la prima sia quella con
tempo di fine lavori minore
          int t = azienda[0].giorno_fine_lavori();
          for(int m=1; m<numero_macchine; m++)
          {
            if(azienda[m].giorno_fine_lavori() < t)
            {
              macchina2 tmp = new macchina2();
              tmp = azienda[0];
              azienda[0] = azienda[m];
              azienda[m] = tmp;
            }
          }
        }



        if(u==numero_macchine)
        {
          out.println("");

out.println("_____________________________________________________________________");
          out.println("");
          out.println("LE " + numero_macchine + " MACCHINE TERMINANO I LAVORI
I GIORNI:");
          for(int i=0; i<numero_macchine; i++)
```

```
          {
            int x = i+1;
            out.println("");
            out.println("--> LA MACCHINA " + x + " HA IMPIEGATO " +
azienda[i].giorno_fine_lavori() + " giorni; ");
            out.println("    HA SVOLTO " + azienda[i].lavori_svolti() + "
LAVORI: " + azienda[i].lavori_fatti_dalla_macchina());
            for(int w=0; w<azienda[i].lavori_svolti(); w++)
            {
              out.println("    " + azienda[i].specifiche_di_un_lavoro(w));
            }
          }
        }

        int end = 0;

        for(int m=0; m< numero_macchine; m++)
        {
          if(azienda[m].giorno_fine_lavori() > end)
          {
            end = azienda[m].giorno_fine_lavori();
          }
        }


        soluzione_euristica[contat] = end;


        contat++;


      }

      if(u==numero_macchine)
      {
        Temp2=System.currentTimeMillis();

        //tempo in millisecondi
        TPrimo=Temp2-Temp1;
      }

      if(u==numero_lavori)
      {
        T2=System.currentTimeMillis();

        //tempo in millisecondi
        TSecondo=T2-T1;
      }


    }

      out.println("");
      out.println("");
      out.println("i valori hanno il seguente significato:");
      out.println("- 1 ----> soluzione euristica");
      out.println("- 2 ----> lower bound");
```

```
      out.println("- 3 ----> z/LB");
      out.println("- 4 ----> tempo di calcolo LB");
      out.println("- 5 ----> tempo di calcolo soluzione euristica");
      out.println("");
      out.println("");

      int end = soluzione_euristica[0];
      int LB = soluzione_euristica[1];


      double rapp = soluzione_euristica[0]/soluzione_euristica[1];

      out.println(soluzione_euristica[0]);
      out.println(soluzione_euristica[1]);
      out.println(rapp);
      out.println(Tempo + TSecondo);
      out.println(Tempo + TPrimo);

     //fine primo
algoritmo_____________________________________________________________
    out.println("");
    out.println("");
    out.println("RISULTATI SECONDO ALGORITMO:");
    out.println("");

    long Tempo_alg2_1;
    long Tempo_alg2_2;
    long Tempo_alg2;
    // ritorna un long che e' il tempo (in millisecondi)
    // fra l'ora attuale e la mezzanotte del 1 gennaio 1970
    Tempo_alg2_1=System.currentTimeMillis();

    long Time_10prove_1;
    long Time_10prove_2;
    long Time_10prove;
    // ritorna un long che e' il tempo (in millisecondi)
    // fra l'ora attuale e la mezzanotte del 1 gennaio 1970
    Time_10prove_1=System.currentTimeMillis();

    //ORA ORDINO L'ARRAY IN MODO CHE LE ATTIVITA' SIANO DISPOSTE IN ORDINE
CRESCENTE

    for(int i=0; i<numero_lavori-1; i++)
    {
      int minimo = ATTIVITA_2[i].get_durata();

      for(int y=i+1; y<numero_lavori; y++)
      {
        if(ATTIVITA_2[y].get_durata() < minimo)
        {
          minimo = ATTIVITA_2[y].get_durata();
          lavoro2 tpm_2 = ATTIVITA_2[i];
          ATTIVITA_2[i] = ATTIVITA_2[y];
          ATTIVITA_2[y] = tpm_2;
        }
      }
    }
```

```
    //ora gli elementi sono ordinati in ordine di durata crescente!!!



    //ORA DEVO ORDINARE L'ARRAY IN MODO CHE SIANO RISPETTATE LE PRECEDENZE!!

    lavoro2[] tpm_2 = new lavoro2[numero_lavori];
    int contatore_2 = 0;

    while(contatore_2 < numero_lavori)
    {
      for(int i=0; i<numero_lavori; i++)
      {
        if(ATTIVITA_2[i] == null)
        {
          continue;
        }

        if(ATTIVITA_2[i].get_numero_precedenze_2() == 0)
        {
          tpm_2[contatore_2] = ATTIVITA_2[i];
          contatore_2++;
          ATTIVITA_2[i] = null;

          for(int k=0; k<numero_lavori; k++)
          {
            if(ATTIVITA_2[k] == null)
            {
              continue;
            }
            ATTIVITA_2[k].cancello_precedenza(tpm_2[contatore_2 -
1].get_numero_lavoro());
          }

          break;
        }
      }
    }

    ATTIVITA_2 = tpm_2;

    Time_10prove_2=System.currentTimeMillis();

    //tempo in millisecondi
    Time_10prove=Time_10prove_2-Time_10prove_1;



    int test_alg2 = 100;

    //qui salverò i risultati ottenuti
    int[] soluzione_euristica_alg2 = new int[test_alg2];
    long[] tempo_di_calcolo = new long[test_alg2];
    int c = 0;


    for(int i=0; i<test_alg2; i++)
```

```
    {
      long T_singolaProva1;
      long T_singolaProva2;

      // ritorna un long che e' il tempo (in millisecondi)
      // fra l'ora attuale e la mezzanotte del 1 gennaio 1970
      T_singolaProva1=System.currentTimeMillis();



      //mi creo un arrey di macchine
      macchina_alg2[] azienda_alg2 = new macchina_alg2[numero_macchine_alg2];

      //creo le macchine
      for(int y=0; y<numero_macchine_alg2; y++)
      {
        macchina_alg2 u = new macchina_alg2();
        azienda_alg2[y] = u;
      }

      int j = i+1;
      out.println("-----------------------------------------------------------");
      out.println("PROVA " + j);
      out.println("");

      //assegno casualmente i lavori alle macchine
      for(int y=0; y<numero_lavori; y++)
      {
        double s = Math.random() * numero_macchine_alg2;
        int scelta = (int) s;     //mi restituisce un numero da 0....numero
macchine-1
        azienda_alg2[scelta].inserire_lavoro(ATTIVITA_2[y]);

        int h = scelta+1;
        out.println("lavoro" + ATTIVITA_2[y].get_numero_lavoro() + " viene
fatto dalla macchina " + h);

      }



      //faccio partire uno alla volta i lavori:
      for(int k=0; k<numero_lavori; k++)
      {


        //verifico quale macchina esegue il lavoro i-esimo
        for(int a=0; a<numero_macchine_alg2; a++)
        {
          //passo direttamente alla successiva se una macchina ha finito i
suoi lavori
          if(azienda_alg2[a].ha_altri_lavori() == false)
          {
            continue;
          }
          else
          {
```

```
            //se la macchina "a" deve fare il lavoro scelto esegue il ciclo
if, altrimenti salto alla macchina successiva!

if(azienda_alg2[a].faccio_questo(ATTIVITA_2[k].get_numero_lavoro()) == true)
            {
              //ora devo verificare il giorno di possibile inizio del nuovo
lavoro
              //ed aggiornare T_min e T_fine del lavoro prima di inserirlo
nella macchina
              int fine_lavori_macchina = azienda_alg2[a].giorno_fine_lavori();

              if(ATTIVITA_2[k].get_numero_precedenze_1() == 0)
              {
                ATTIVITA_2[k].set_Tmin(fine_lavori_macchina);
//metodo nuovo_lavoro
                ATTIVITA_2[k].set_Tfine();
//mi inserisce l'attivita sulla macchina
                azienda_alg2[a].fai_lavoro(ATTIVITA_2[k]);
//e mi aggiorna il tempo di fine lavori
              }
              else
              {
                for(int w=0; w<ATTIVITA_2[k].get_numero_precedenze_1(); w++)
                {
                  int p = ATTIVITA_2[k].get_precedenza_i(w);

                  for(int y=0; y<numero_lavori; y++)
                  {
                    if(p == ATTIVITA_2[y].get_numero_lavoro())
                    {
                      int t_fine = ATTIVITA_2[y].get_Tfine();
                      if(t_fine > fine_lavori_macchina)
                      {
                        fine_lavori_macchina = t_fine;
                        break;
                      }
                    }
                  }
                }
                ATTIVITA_2[k].set_Tmin(fine_lavori_macchina);
                ATTIVITA_2[k].set_Tfine();
                azienda_alg2[a].fai_lavoro(ATTIVITA_2[k]);
              }
            }
          }
        }
      }

      out.println("");

      out.println("");
      out.println("LE " + numero_macchine_alg2 + " MACCHINE TERMINANO I LAVORI
I GIORNI:");
      for(int z=0; z<numero_macchine_alg2; z++)
      {
        int x = z+1;
        out.println("");
```

```
        out.println("--> LA MACCHINA " + x + " HA IMPIEGATO " +
azienda_alg2[z].giorno_fine_lavori() + " giorni; ");
        out.println("    HA SVOLTO " + azienda_alg2[z].lavori_svolti() + "
LAVORI: " + azienda_alg2[z].lavori_fatti_dalla_macchina());
        for(int b=0; b<azienda_alg2[z].lavori_svolti(); b++)
        {
          out.println("    " + azienda_alg2[z].specifiche_di_un_lavoro(b));
        }
      }


      //ora la soluzione euristica sarà il T_fine maggiore
      //lo trovo e salvo il risultato sull'array
      int end_alg2 = 0;

      for(int m=0; m< numero_macchine_alg2; m++)
      {
        if(azienda_alg2[m].giorno_fine_lavori() > end_alg2)
        {
          end_alg2 = azienda_alg2[m].giorno_fine_lavori();
        }
      }

      soluzione_euristica_alg2[i] = end_alg2;

      T_singolaProva2=System.currentTimeMillis();


      if(i<10)
      {
        //tempo in millisecondi
        tempo_di_calcolo[i]=T_singolaProva2-T_singolaProva1;
      }
    }

    Tempo_alg2_2=System.currentTimeMillis();

    //tempo in millisecondi
    Tempo_alg2=Tempo_alg2_2-Tempo_alg2_1;


    out.println("");
    out.println("");
    out.println("DI SEGUITO VENGONO ELENCATE LA SOLUZIONI OTTENUTE NEI " +
test_alg2 + "TEST FATTI:");
    for(int t=0; t<test_alg2; t++)
    {
      int r = t+1;
      out.println("prova " + r + " ----> " + soluzione_euristica_alg2[t]);
    }

    out.println("");
    out.println("");

    int migliore_1 = soluzione_euristica_alg2[0];
    int migliore_10 = soluzione_euristica_alg2[0];
    int migliore_100 = soluzione_euristica_alg2[0];
```

```
    for(int t=1; t<10; t++)
    {
      if(soluzione_euristica_alg2[t] < migliore_10)
      {
        migliore_10 = soluzione_euristica_alg2[t];
      }
    }

    for(int t=1; t<test_alg2; t++)
    {
      if(soluzione_euristica_alg2[t] < migliore_100)
      {
        migliore_100 = soluzione_euristica_alg2[t];
      }
    }

    out.println("tra queste la migliore è: " + migliore_100);
    out.println("tempo di calcolo: " + Tempo_alg2);

    out.println("");

out.println("_____________________________________________________________________");

out.println("_____________________________________________________________________");

    if(end < migliore_100)
    {
      out.println("");
      out.println("LA SOLUZIONE EURISTICA MIGLIORE E' STATA OTTENUTA CON IL
PRIMO ALGORITMO :" + end + "!");
    }
    else if(end == migliore_100)
    {
      out.println("");
      out.println("CON ENTRAMBI GLI ALGORITMI HO OTTENUTO LA SOLUZIONE
EURISTICA: " + end + "!");
    }
    else
    {
      out.println("");
      out.println("LA SOLUZIONE EURISTICA MIGLIORE E' STATA OTTENUTA CON IL
SECONDO ALGORITMO :" + migliore_100 + "!");
    }

    long x = 0;
    for(int h=0; h<10; h++)
    {
      x = x + tempo_di_calcolo[h];
    }

    out.println("");
    out.println("");
    out.println("");
    out.println(LB);                                //lower bound
    out.println(end);                               //soluz algoritmo 1
    out.println(migliore_1);                          //soluzione alg 2  1run
```

```
    out.println(migliore_10);                          //soluzione alg 2  10run
    out.println(migliore_100);                          //soluzione alg 2
100run
    out.println(Tempo + TSecondo);                 //tempo lb
    out.println(Tempo + TPrimo);                   //tempo alg 1
    out.println(tempo_di_calcolo[0] + Time_10prove);    //tempo di calcolo
singola prova (alg 2)
    out.println(x + Time_10prove);                 //tempo alg 2 (10 prove)
    out.println(Tempo_alg2);                       //tempo alg 2 (100 prove)



    out.close();
  }

}
```

6.3 RISULTATI DEI TEST EFFETTUATI SUI DUE ALGORITMI

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	10
LOWER BOUND	42	24	37	15	22	28	MACCHINE =	3
Z (alg. 1)	42	33	38	26	29	33,6	Pmin=	0,2
Z (alg.2 con 1 RUN)	45	41	45	41	24	39,2		
Z (alg.2 con 10 RUN)	42	33	38	24	24	32,2		
Z (alg.2 con 100 RUN)	42	26	37	22	22	29,8		
Tempo di calcolo LB	0	0	0	0	0	0		
Tempo di calcolo alg. 1	0	0	0	0	0	0		
T. di calcolo alg. 2 (1 RUN)	0	0	0	0	16	3,2		
T. di calcolo alg. 2 (10 RUN)	16	16	0	16	16	12,8		
T. di calcolo alg. 2 (100 RUN)	32	31	15	31	32	28,2		
Z migliore	42	26	37	22	22	29,8		
Z/LB	1	1,083	1	1,466	1	1,11		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	10
LOWER BOUND	25	28	22	23	24	24,4	MACCHINE =	4
Z (alg. 1)	27	32	22	26	26	26,6	Pmin=	0,2
Z (alg.2 con 1 RUN)	28	44	29	25	37	32,6		
Z (alg.2 con 10 RUN)	26	30	22	25	24	25,4		
Z (alg.2 con 100 RUN)	25	28	22	23	24	24,4		
Tempo di calcolo LB	0	16	16	0	0	6,4		
Tempo di calcolo alg. 1	0	16	16	0	0	6,4		
T. di calcolo alg. 2 (1 RUN)	0	0	0	0	0	0		
T. di calcolo alg. 2 (10 RUN)	0	0	0	0	15	3		
T. di calcolo alg. 2 (100 RUN)	15	16	16	15	31	18,6		
Z migliore	25	28	22	23	24	24,4		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	10
LOWER BOUND	29	24	24	22	24	24,6	MACCHINE =	5
Z (alg. 1)	29	24	24	27	24	25,6	Pmin=	0,2
Z (alg.2 con 1 RUN)	36	29	24	32	47	33,6		
Z (alg.2 con 10 RUN)	29	24	24	22	24	24,6		
Z (alg.2 con 100 RUN)	29	24	24	22	24	24,6		
Tempo di calcolo LB	0	0	0	0	0	0		
Tempo di calcolo alg. 1	0	0	0	0	16	3,2		
T. di calcolo alg. 2 (1 RUN)	0	0	0	0	0	0		
T. di calcolo alg. 2 (10 RUN)	15	15	0	0	0	6		
T. di calcolo alg. 2 (100 RUN)	31	31	16	16	16	22		
Z migliore	29	24	24	22	24	24,6		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	10
LOWER BOUND	30	38	21	41	36	33,2	MACCHINE =	3
Z (alg. 1)	30	38	27	41	38	34,8	Pmin=	0,5
Z (alg.2 con 1 RUN)	30	38	26	44	51	37,8		
Z (alg.2 con 10 RUN)	30	38	26	41	38	34,6		
Z (alg.2 con 100 RUN)	30	38	21	41	38	33,6		
Tempo di calcolo LB	0	0	0	16	0	3,2		
Tempo di calcolo alg. 1	0	0	0	16	0	3,2		
T. di calcolo alg. 2 (1 RUN)	16	0	0	0	0	3,2		
T. di calcolo alg. 2 (10 RUN)	16	15	0	0	0	6,2		
T. di calcolo alg. 2 (100 RUN)	31	31	15	15	16	21,6		
Z migliore	30	38	21	41	38	33,6		
Z/LB	1	1	1	1	1,055	1,011		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	10
LOWER BOUND	43	49	24	45	42	40,6	MACCHINE =	4
Z (alg. 1)	43	49	27	45	42	41,2	Pmin=	0,5
Z (alg.2 con 1 RUN)	60	49	38	54	42	48,6		
Z (alg.2 con 10 RUN)	43	49	31	49	42	42,8		
Z (alg.2 con 100 RUN)	43	49	24	45	42	40,6		
Tempo di calcolo LB	0	0	0	0	15	3		
Tempo di calcolo alg. 1	0	0	0	0	15	3		
T. di calcolo alg. 2 (1 RUN)	16	0	0	0	0	3,2		
T. di calcolo alg. 2 (10 RUN)	16	15	15	0	0	9,2		
T. di calcolo alg. 2 (100 RUN)	32	31	31	16	16	25,2		
Z migliore	43	49	24	45	42	40,6		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	10
LOWER BOUND	44	47	34	42	62	45,8	MACCHINE =	5
Z (alg. 1)	44	47	34	42	62	45,8	Pmin=	0,5
Z (alg.2 con 1 RUN)	51	53	34	45	62	49		
Z (alg.2 con 10 RUN)	44	47	34	42	62	45,8		
Z (alg.2 con 100 RUN)	44	47	34	42	62	45,8		
Tempo di calcolo LB	0	0	0	0	0	0		
Tempo di calcolo alg. 1	0	0	0	0	0	0		
T. di calcolo alg. 2 (1 RUN)	0	0	0	15	0	3		
T. di calcolo alg. 2 (10 RUN)	15	15	16	15	0	12,2		
T. di calcolo alg. 2 (100 RUN)	47	31	31	31	31	34,2		
Z migliore	44	47	34	42	62	45,8		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	10
LOWER BOUND	68	53	49	40	50	52	MACCHINE =	3
Z (alg. 1)	68	53	49	40	50	52	Pmin=	0,8
Z (alg.2 con 1 RUN)	68	61	49	40	57	55		
Z (alg.2 con 10 RUN)	68	53	49	40	50	52		
Z (alg.2 con 100 RUN)	68	53	49	40	50	52		
Tempo di calcolo LB	0	0	0	0	0	0		
Tempo di calcolo alg. 1	0	0	0	0	0	0		
T. di calcolo alg. 2 (1 RUN)	0	15	0	16	0	6,2		
T. di calcolo alg. 2 (10 RUN)	0	15	0	16	0	6,2		
T. di calcolo alg. 2 (100 RUN)	16	31	31	32	31	28,2		
Z migliore	68	53	49	40	50	52		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	10
LOWER BOUND	47	62	34	38	36	43,4	MACCHINE =	4
Z (alg. 1)	47	62	34	38	36	43,4	Pmin=	0,8
Z (alg.2 con 1 RUN)	47	69	37	39	36	45,6		
Z (alg.2 con 10 RUN)	47	62	34	38	36	43,4		
Z (alg.2 con 100 RUN)	47	62	34	38	36	43,4		
Tempo di calcolo LB	0	0	0	0	16	3,2		
Tempo di calcolo alg. 1	0	0	0	0	16	3,2		
T. di calcolo alg. 2 (1 RUN)	0	15	0	0	0	3		
T. di calcolo alg. 2 (10 RUN)	0	15	16	16	0	9,4		
T. di calcolo alg. 2 (100 RUN)	16	31	31	32	15	25		
Z migliore	47	62	34	38	36	43,4		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	10
LOWER BOUND	37	41	37	55	61	46,2	MACCHINE =	5
Z (alg. 1)	37	41	37	55	61	46,2	Pmin=	0,8
Z (alg.2 con 1 RUN)	39	45	37	55	61	47,4		
Z (alg.2 con 10 RUN)	37	41	37	55	61	46,2		
Z (alg.2 con 100 RUN)	37	41	37	55	61	46,2		
Tempo di calcolo LB	16	0	0	0	0	3,2		
Tempo di calcolo alg. 1	16	0	0	0	0	3,2		
T. di calcolo alg. 2 (1 RUN)	0	0	0	0	0	0		
T. di calcolo alg. 2 (10 RUN)	0	16	0	0	16	6,4		
T. di calcolo alg. 2 (100 RUN)	31	31	15	32	31	28		
Z migliore	37	41	37	55	61	46,2		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	50
LOWER BOUND	72	133	106	89	97	99,4	MACCHINE =	13
Z (alg. 1)	73	133	106	94	102	101,6	Pmin=	0,2
Z (alg.2 con 1 RUN)	119	165	140	112	125	132,2		
Z (alg.2 con 10 RUN)	92	134	110	109	114	111,8		
Z (alg.2 con 100 RUN)	83	133	106	93	102	103,4		
Tempo di calcolo LB	46	0	15	0	16	15,4		
Tempo di calcolo alg. 1	46	0	15	16	16	18,6		
T. di calcolo alg. 2 (1 RUN)	0	0	0	0	0	0		
T. di calcolo alg. 2 (10 RUN)	32	16	32	31	31	28,4		
T. di calcolo alg. 2 (100 RUN)	110	94	94	94	109	100,2		
Z migliore	73	133	106	93	102	101,4		
Z/LB	1,013	1	1	1,044	1,051	1,022		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	50
LOWER BOUND	71	71	79	118	92	86,2	MACCHINE =	17
Z (alg. 1)	71	71	79	118	92	86,2	Pmin=	0,2
Z (alg.2 con 1 RUN)	78	82	106	118	119	100,6		
Z (alg.2 con 10 RUN)	78	82	87	118	110	95		
Z (alg.2 con 100 RUN)	72	76	80	118	92	87,6		
Tempo di calcolo LB	0	31	15	16	15	15,4		
Tempo di calcolo alg. 1	16	31	15	16	15	18,6		
T. di calcolo alg. 2 (1 RUN)	0	0	0	0	16	3,2		
T. di calcolo alg. 2 (10 RUN)	31	31	32	31	31	31,2		
T. di calcolo alg. 2 (100 RUN)	109	94	110	94	110	103,4		
Z migliore	71	71	79	118	92	86,2		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	50
LOWER BOUND	76	76	88	101	96	87,4	MACCHINE =	25
Z (alg. 1)	76	76	88	101	96	87,4	Pmin=	0,2
Z (alg.2 con 1 RUN)	100	89	88	138	130	109		
Z (alg.2 con 10 RUN)	87	85	88	101	96	91,4		
Z (alg.2 con 100 RUN)	76	76	88	101	96	87,4		
Tempo di calcolo LB	0	16	0	16	15	9,4		
Tempo di calcolo alg. 1	16	16	16	16	15	15,8		
T. di calcolo alg. 2 (1 RUN)	0	0	0	15	0	3		
T. di calcolo alg. 2 (10 RUN)	31	31	31	46	31	34		
T. di calcolo alg. 2 (100 RUN)	109	109	109	109	110	109,2		
Z migliore	76	76	88	101	96	87,4		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	50
LOWER BOUND	160	151	172	196	158	167,4	MACCHINE =	13
Z (alg. 1)	160	151	172	196	158	167,4	Pmin=	0,5
Z (alg.2 con 1 RUN)	174	164	175	198	161	174,4		
Z (alg.2 con 10 RUN)	163	158	175	196	158	170		
Z (alg.2 con 100 RUN)	160	151	172	196	158	167,4		
Tempo di calcolo LB	16	16	16	15	15	15,6		
Tempo di calcolo alg. 1	16	16	16	15	15	15,6		
T. di calcolo alg. 2 (1 RUN)	0	0	15	16	0	6,2		
T. di calcolo alg. 2 (10 RUN)	31	31	46	47	32	37,4		
T. di calcolo alg. 2 (100 RUN)	109	109	109	110	110	109,4		
Z migliore	160	151	172	196	158	167,4		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	50
LOWER BOUND	152	181	166	182	181	172,4	MACCHINE =	17
Z (alg. 1)	152	181	166	182	181	172,4	Pmin=	0,5
Z (alg.2 con 1 RUN)	180	186	173	190	189	183,6		
Z (alg.2 con 10 RUN)	152	184	167	183	181	173,4		
Z (alg.2 con 100 RUN)	152	181	166	182	181	172,4		
Tempo di calcolo LB	15	0	16	15	16	12,4		
Tempo di calcolo alg. 1	15	16	16	15	16	15,6		
T. di calcolo alg. 2 (1 RUN)	0	0	16	0	15	6,2		
T. di calcolo alg. 2 (10 RUN)	32	31	47	31	46	37,4		
T. di calcolo alg. 2 (100 RUN)	110	109	109	109	125	112,4		
Z migliore	152	181	166	182	181	172,4		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	50
LOWER BOUND	202	135	194	148	175	170,8	MACCHINE =	25
Z (alg. 1)	202	135	194	148	175	170,8	Pmin=	0,5
Z (alg.2 con 1 RUN)	207	152	194	148	185	177,2		
Z (alg.2 con 10 RUN)	202	135	194	148	175	170,8		
Z (alg.2 con 100 RUN)	202	135	194	148	175	170,8		
Tempo di calcolo LB	15	16	15	0	16	12,4		
Tempo di calcolo alg. 1	15	16	15	16	16	15,6		
T. di calcolo alg. 2 (1 RUN)	16	16	0	0	0	6,4		
T. di calcolo alg. 2 (10 RUN)	47	47	47	31	31	40,6		
T. di calcolo alg. 2 (100 RUN)	125	109	110	109	125	115,6		
Z migliore	202	135	194	148	175	170,8		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	50
LOWER BOUND	257	211	235	250	248	240,2	MACCHINE =	13
Z (alg. 1)	257	211	235	250	248	240,2	Pmin=	0,8
Z (alg.2 con 1 RUN)	257	211	237	250	248	240,6		
Z (alg.2 con 10 RUN)	257	211	235	250	248	240,2		
Z (alg.2 con 100 RUN)	257	211	235	250	248	240,2		
Tempo di calcolo LB	16	0	16	16	0	9,6		
Tempo di calcolo alg. 1	16	15	16	16	15	15,6		
T. di calcolo alg. 2 (1 RUN)	16	0	15	0	0	6,2		
T. di calcolo alg. 2 (10 RUN)	47	32	47	31	31	37,6		
T. di calcolo alg. 2 (100 RUN)	125	110	125	109	110	115,8		
Z migliore	257	211	235	250	248	240,2		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	50
LOWER BOUND	249	256	215	239	194	230,6	MACCHINE =	17
Z (alg. 1)	249	256	215	239	194	230,6	Pmin=	0,8
Z (alg.2 con 1 RUN)	255	257	217	248	194	234,2		
Z (alg.2 con 10 RUN)	249	256	215	239	194	230,6		
Z (alg.2 con 100 RUN)	249	256	215	239	194	230,6		
Tempo di calcolo LB	0	32	0	0	16	9,6		
Tempo di calcolo alg. 1	16	32	15	16	16	19		
T. di calcolo alg. 2 (1 RUN)	0	0	0	0	16	3,2		
T. di calcolo alg. 2 (10 RUN)	31	31	32	31	47	34,4		
T. di calcolo alg. 2 (100 RUN)	109	109	110	109	125	112,4		
Z migliore	249	256	215	239	194	230,6		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	50
LOWER BOUND	248	203	216	246	224	227,4	MACCHINE =	25
Z (alg. 1)	248	203	216	246	224	227,4	Pmin=	0,8
Z (alg.2 con 1 RUN)	259	203	221	250	224	231,4		
Z (alg.2 con 10 RUN)	248	203	216	246	224	227,4		
Z (alg.2 con 100 RUN)	248	203	216	246	224	227,4		
Tempo di calcolo LB	0	31	15	16	16	15,6		
Tempo di calcolo alg. 1	16	31	15	16	16	18,8		
T. di calcolo alg. 2 (1 RUN)	0	0	16	0	0	3,2		
T. di calcolo alg. 2 (10 RUN)	31	32	47	31	31	34,4		
T. di calcolo alg. 2 (100 RUN)	109	110	125	109	109	112,4		
Z migliore	248	203	216	246	224	227,4		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	100
LOWER BOUND	157	173	140	205	215	178	MACCHINE =	25
Z (alg. 1)	157	173	140	205	215	178	Pmin=	0,2
Z (alg.2 con 1 RUN)	202	198	212	233	239	216,8		
Z (alg.2 con 10 RUN)	174	177	164	220	223	191,6		
Z (alg.2 con 100 RUN)	165	173	140	205	215	179,6		
Tempo di calcolo LB	31	31	32	31	16	28,2		
Tempo di calcolo alg. 1	31	31	32	31	32	31,4		
T. di calcolo alg. 2 (1 RUN)	15	0	0	15	15	9		
T. di calcolo alg. 2 (10 RUN)	62	47	46	62	62	55,8		
T. di calcolo alg. 2 (100 RUN)	156	141	140	140	140	143,4		
Z migliore	157	173	140	205	215	178		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	100
LOWER BOUND	165	168	209	140	192	174,8	MACCHINE =	34
Z (alg. 1)	165	168	209	140	192	174,8	Pmin=	0,2
Z (alg.2 con 1 RUN)	196	197	216	174	209	198,4		
Z (alg.2 con 10 RUN)	175	178	209	142	204	181,6		
Z (alg.2 con 100 RUN)	167	173	209	140	192	176,2		
Tempo di calcolo LB	31	16	15	16	31	21,8		
Tempo di calcolo alg. 1	31	32	31	31	31	31,2		
T. di calcolo alg. 2 (1 RUN)	0	0	15	0	0	3		
T. di calcolo alg. 2 (10 RUN)	47	46	62	47	47	49,8		
T. di calcolo alg. 2 (100 RUN)	141	140	140	141	141	140,6		
Z migliore	165	168	209	140	192	174,8		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	100
LOWER BOUND	192	158	172	174	182	175,6	MACCHINE =	50
Z (alg. 1)	192	158	172	174	182	175,6	Pmin=	0,2
Z (alg.2 con 1 RUN)	223	191	185	201	204	200,8		
Z (alg.2 con 10 RUN)	192	167	173	184	193	181,8		
Z (alg.2 con 100 RUN)	192	158	172	174	182	175,6		
Tempo di calcolo LB	32	31	15	31	0	21,8		
Tempo di calcolo alg. 1	32	31	31	31	15	28		
T. di calcolo alg. 2 (1 RUN)	0	0	16	0	16	6,4		
T. di calcolo alg. 2 (10 RUN)	62	47	62	63	63	59,4		
T. di calcolo alg. 2 (100 RUN)	156	156	156	141	157	153,2		
Z migliore	192	158	172	174	182	175,6		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	100
LOWER BOUND	351	297	351	350	345	338,8	MACCHINE =	25
Z (alg. 1)	351	297	351	350	345	338,8	Pmin=	0,5
Z (alg.2 con 1 RUN)	377	314	364	350	359	352,8		
Z (alg.2 con 10 RUN)	354	304	351	350	350	341,8		
Z (alg.2 con 100 RUN)	351	297	351	350	345	338,8		
Tempo di calcolo LB	32	31	16	31	31	28,2		
Tempo di calcolo alg. 1	47	47	31	31	31	37,4		
T. di calcolo alg. 2 (1 RUN)	0	16	16	16	16	12,8		
T. di calcolo alg. 2 (10 RUN)	94	63	78	62	78	75		
T. di calcolo alg. 2 (100 RUN)	203	157	172	172	156	172		
Z migliore	351	297	351	350	345	338,8		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	100
LOWER BOUND	302	386	343	319	390	348	MACCHINE =	34
Z (alg. 1)	302	386	343	319	390	348	Pmin=	0,5
Z (alg.2 con 1 RUN)	302	386	347	319	401	351		
Z (alg.2 con 10 RUN)	302	386	343	319	390	348		
Z (alg.2 con 100 RUN)	302	386	343	319	390	348		
Tempo di calcolo LB	31	31	30	32	16	28		
Tempo di calcolo alg. 1	31	47	46	32	31	37,4		
T. di calcolo alg. 2 (1 RUN)	16	15	16	15	16	15,6		
T. di calcolo alg. 2 (10 RUN)	63	62	63	78	78	68,8		
T. di calcolo alg. 2 (100 RUN)	157	156	157	172	172	162,8		
Z migliore	302	386	343	319	390	348		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	100
LOWER BOUND	342	366	307	365	361	348,2	MACCHINE =	50
Z (alg. 1)	342	366	307	365	361	348,2	Pmin=	0,5
Z (alg.2 con 1 RUN)	362	371	337	368	371	361,8		
Z (alg.2 con 10 RUN)	343	366	308	365	361	348,6		
Z (alg.2 con 100 RUN)	342	366	307	365	361	348,2		
Tempo di calcolo LB	16	31	31	16	31	25		
Tempo di calcolo alg. 1	32	31	47	32	47	37,8		
T. di calcolo alg. 2 (1 RUN)	15	15	0	15	16	12,2		
T. di calcolo alg. 2 (10 RUN)	78	78	62	78	63	71,8		
T. di calcolo alg. 2 (100 RUN)	171	172	156	187	156	168,4		
Z migliore	342	366	307	365	361	348,2		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	100
LOWER BOUND	490	511	465	431	486	476,6	MACCHINE =	25
Z (alg. 1)	490	511	465	431	486	476,6	Pmin=	0,8
Z (alg.2 con 1 RUN)	491	512	467	431	490	478,2		
Z (alg.2 con 10 RUN)	490	511	465	431	486	476,6		
Z (alg.2 con 100 RUN)	490	511	465	431	486	476,6		
Tempo di calcolo LB	46	31	46	32	32	37,4		
Tempo di calcolo alg. 1	62	47	62	47	47	53		
T. di calcolo alg. 2 (1 RUN)	16	31	16	16	15	18,8		
T. di calcolo alg. 2 (10 RUN)	63	78	63	62	62	65,6		
T. di calcolo alg. 2 (100 RUN)	188	188	172	187	172	181,4		
Z migliore	490	511	465	431	486	476,6		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	100
LOWER BOUND	407	408	484	491	469	451,8	MACCHINE =	34
Z (alg. 1)	407	408	484	491	469	451,8	Pmin=	0,8
Z (alg.2 con 1 RUN)	407	409	489	497	469	454,2		
Z (alg.2 con 10 RUN)	407	408	484	491	469	451,8		
Z (alg.2 con 100 RUN)	407	408	484	491	469	451,8		
Tempo di calcolo LB	31	47	31	32	31	34,4		
Tempo di calcolo alg. 1	47	62	47	47	47	50		
T. di calcolo alg. 2 (1 RUN)	15	16	16	16	15	15,6		
T. di calcolo alg. 2 (10 RUN)	62	63	63	62	62	62,4		
T. di calcolo alg. 2 (100 RUN)	187	172	172	187	172	178		
Z migliore	407	408	484	491	469	451,8		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	100
LOWER BOUND	470	489	414	525	486	476,8	MACCHINE =	50
Z (alg. 1)	470	489	414	525	486	476,8	Pmin=	0,8
Z (alg.2 con 1 RUN)	470	489	414	528	490	478,2		
Z (alg.2 con 10 RUN)	470	489	414	525	486	476,8		
Z (alg.2 con 100 RUN)	470	489	414	525	486	476,8		
Tempo di calcolo LB	31	31	31	31	31	31		
Tempo di calcolo alg. 1	47	47	47	47	47	47		
T. di calcolo alg. 2 (1 RUN)	15	15	15	15	16	15,2		
T. di calcolo alg. 2 (10 RUN)	78	62	62	62	63	65,4		
T. di calcolo alg. 2 (100 RUN)	203	172	172	171	188	181,2		
Z migliore	470	489	414	525	486	476,8		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	200
LOWER BOUND	328	423	332	337	369	357,8	MACCHINE =	50
Z (alg. 1)	328	423	332	337	369	357,8	Pmin=	0,2
Z (alg.2 con 1 RUN)	360	459	366	363	399	389,4		
Z (alg.2 con 10 RUN)	336	427	344	346	372	365		
Z (alg.2 con 100 RUN)	330	424	343	340	369	361,2		
Tempo di calcolo LB	77	78	78	77	78	77,6		
Tempo di calcolo alg. 1	109	109	109	109	110	109,2		
T. di calcolo alg. 2 (1 RUN)	0	0	0	0	0	0		
T. di calcolo alg. 2 (10 RUN)	31	31	31	31	31	31		
T. di calcolo alg. 2 (100 RUN)	266	266	250	266	250	259,6		
Z migliore	328	423	332	337	369	357,8		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	200
LOWER BOUND	354	326	301	383	416	356	MACCHINE =	67
Z (alg. 1)	354	326	301	383	416	356	Pmin=	0,2
Z (alg.2 con 1 RUN)	413	347	360	396	430	389,2		
Z (alg.2 con 10 RUN)	367	337	323	383	428	367,6		
Z (alg.2 con 100 RUN)	354	326	306	383	416	357		
Tempo di calcolo LB	78	79	78	79	61	75		
Tempo di calcolo alg. 1	109	110	109	110	93	106,2		
T. di calcolo alg. 2 (1 RUN)	0	0	0	0	16	3,2		
T. di calcolo alg. 2 (10 RUN)	31	31	32	31	32	31,4		
T. di calcolo alg. 2 (100 RUN)	266	265	266	265	266	265,6		
Z migliore	354	326	301	383	416	356		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	200
LOWER BOUND	328	355	318	324	382	341,4	MACCHINE =	100
Z (alg. 1)	328	355	318	324	382	341,4	Pmin=	0,2
Z (alg.2 con 1 RUN)	341	355	361	329	414	360		
Z (alg.2 con 10 RUN)	332	355	318	324	382	342,2		
Z (alg.2 con 100 RUN)	328	355	318	324	382	341,4		
Tempo di calcolo LB	78	78	63	63	78	72		
Tempo di calcolo alg. 1	109	110	94	110	109	106,4		
T. di calcolo alg. 2 (1 RUN)	31	0	16	0	0	9,4		
T. di calcolo alg. 2 (10 RUN)	109	31	47	31	31	49,8		
T. di calcolo alg. 2 (100 RUN)	375	281	297	265	281	299,8		
Z migliore	328	355	318	324	382	341,4		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	NUMERO =	200
LOWER BOUND	730	708	698	743	663	708,4	MACCHINE =	50
Z (alg. 1)	730	708	698	743	663	708,4	Pmin=	0,5
Z (alg.2 con 1 RUN)	747	708	705	773	664	719,4		
Z (alg.2 con 10 RUN)	730	708	698	743	663	708,4		
Z (alg.2 con 100 RUN)	730	708	698	743	663	708,4		
Tempo di calcolo LB	62	93	95	93	78	84,2		
Tempo di calcolo alg. 1	141	156	157	156	125	147		
T. di calcolo alg. 2 (1 RUN)	15	16	0	0	16	9,4		
T. di calcolo alg. 2 (10 RUN)	62	63	47	47	47	53,2		
T. di calcolo alg. 2 (100 RUN)	484	469	468	453	453	465,4		
Z migliore	730	708	698	743	663	708,4		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	200
LOWER BOUND	697	701	711	697	616	684,4	MACCHINE =	67
Z (alg. 1)	697	701	711	697	616	684,4	Pmin=	0,5
Z (alg.2 con 1 RUN)	719	720	718	706	635	699,6		
Z (alg.2 con 10 RUN)	699	704	711	697	616	685,4		
Z (alg.2 con 100 RUN)	697	701	711	697	616	684,4		
Tempo di calcolo LB	93	94	94	94	94	93,8		
Tempo di calcolo alg. 1	156	156	156	156	156	156		
T. di calcolo alg. 2 (1 RUN)	16	15	16	16	15	15,6		
T. di calcolo alg. 2 (10 RUN)	47	47	47	63	47	50,2		
T. di calcolo alg. 2 (100 RUN)	453	469	453	469	469	462,6		
Z migliore	697	701	711	697	616	684,4		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	200
LOWER BOUND	697	697	652	631	705	676,4	MACCHINE =	100
Z (alg. 1)	697	697	652	631	705	676,4	Pmin=	0,5
Z (alg.2 con 1 RUN)	699	705	667	631	705	681,4		
Z (alg.2 con 10 RUN)	697	697	652	631	705	676,4		
Z (alg.2 con 100 RUN)	697	697	652	631	705	676,4		
Tempo di calcolo LB	78	94	79	109	79	87,8		
Tempo di calcolo alg. 1	141	156	141	156	141	147		
T. di calcolo alg. 2 (1 RUN)	15	16	16	16	15	15,6		
T. di calcolo alg. 2 (10 RUN)	62	63	62	47	62	59,2		
T. di calcolo alg. 2 (100 RUN)	468	469	484	453	469	468,6		
Z migliore	697	697	652	631	705	676,4		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	200
LOWER BOUND	946	937	862	955	982	936,4	MACCHINE =	50
Z (alg. 1)	946	937	862	955	982	936,4	Pmin=	0,8
Z (alg.2 con 1 RUN)	946	937	863	956	982	936,8		
Z (alg.2 con 10 RUN)	946	937	862	955	982	936,4		
Z (alg.2 con 100 RUN)	946	937	862	955	982	936,4		
Tempo di calcolo LB	187	95	78	93	94	109,4		
Tempo di calcolo alg. 1	234	204	156	171	172	187,4		
T. di calcolo alg. 2 (1 RUN)	32	0	15	16	16	15,8		
T. di calcolo alg. 2 (10 RUN)	94	62	78	79	78	78,2		
T. di calcolo alg. 2 (100 RUN)	672	656	672	657	656	662,6		
Z migliore	946	937	862	955	982	936,4		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA	LAVORI =	200
LOWER BOUND	939	1018	904	991	954	961,2	MACCHINE =	67
Z (alg. 1)	939	1018	904	991	954	961,2	Pmin=	0,8
Z (alg.2 con 1 RUN)	939	1021	906	1005	957	965,6		
Z (alg.2 con 10 RUN)	939	1018	904	991	954	961,2		
Z (alg.2 con 100 RUN)	939	1018	904	991	954	961,2		
Tempo di calcolo LB	94	78	78	79	94	84,6		
Tempo di calcolo alg. 1	172	156	172	157	172	165,8		
T. di calcolo alg. 2 (1 RUN)	16	16	16	15	16	15,8		
T. di calcolo alg. 2 (10 RUN)	78	79	78	78	78	78,2		
T. di calcolo alg. 2 (100 RUN)	641	641	656	625	641	640,8		
Z migliore	939	1018	904	991	954	961,2		
Z/LB	1	1	1	1	1	1		

	TEST_1	TEST_2	TEST_3	TEST_4	TEST_5	MEDIA
LOWER BOUND	919	1031	923	971	944	957,6
Z (alg. 1)	919	1031	923	971	944	957,6
Z (alg.2 con 1 RUN)	919	1034	923	972	944	958,4
Z (alg.2 con 10 RUN)	919	1031	923	971	944	957,6
Z (alg.2 con 100 RUN)	919	1031	923	971	944	957,6
Tempo di calcolo LB	94	94	94	93	94	93,8
Tempo di calcolo alg. 1	172	172	172	171	172	171,8
T. di calcolo alg. 2 (1 RUN)	16	16	0	0	0	6,4
T. di calcolo alg. 2 (10 RUN)	63	63	63	63	63	63
T. di calcolo alg. 2 (100 RUN)	625	609	625	594	610	612,6
Z migliore	919	1031	923	971	944	957,6
Z/LB	1	1	1	1	1	1

LAVORI =	200
MACCHINE =	100
Pmin=	0,8