Università degli Studi di Padova
Corso di Laurea Magistrale in Ingegneria Informatica

Interval scheduling problem
analisi teorica e test

Relatori: Prof. Giorgio Romanin Jacur
Prof. Matteo Fischetti
Studente: Luca Zoggia
Matricola: 603638-IF
Anno Accademico: 2009/2010

Indice

Sommario

La tesi ha avuto come obiettivi

- analisi di quanto presente in letteratura relativamente all'*interval scheduling problem*;
- test di modelli matematici per situazioni riconducibili al servizio di sosta degli aerei in aeroporto.

La fase di analisi teorica è partita da un inquadramento generale dell'interval scheduling problem, dei problemi derivabili e dei relativi algoritmi. Successivamente ci si è concentrati su documentazione relativa ad applicazioni alle realtà aeroportuali. L'obiettivo è di massimizzare il guadagno dato dallo svolgere le operazioni di rampa. Il guadagno è dato dalla somma dei profitti economici del servire gli aerei. Quindi per ogni aereo sarà specificato il relativo profitto. Tutti gli aerei devono essere serviti senza eccedere la disponibilità delle varie tipologie di piazzole di sosta.

La parte operativa della tesi si è concentrata nel testare i modelli relativi al problema di interesse. Sono state analizzate situazioni sia feasible che infeasible. La principale misura di riferimento è stato il tempo di esecuzione. Per i rilassamenti al modello e le soluzioni euristiche si è analizzato il valore della soluzione ricavata. Le soluzioni euristiche sono date da esecuzioni dell'algoritmo *simulated annealing*. Tutti i modelli delle istanze testate sono stati processati ricorrendo all'installazione di *Cplex* (versione 7.5.0) nel server dell'Università degli Studi di Padova a Vicenza.

Capitolo 1

Introduzione

Il traffico aereo presenta da anni una tendenza all'aumento, nonostante periodi di flessione in seguito alla caduta delle Twin Towers o alla recente crisi economica. L'incremento riguarda sia il trasporto di persone che quello delle merci.

Gli investimenti nelle infrastrutture e nei sistemi di gestione degli aeroporti spesso non sono stati adeguati alla crescente richiesta di voli. Il risultato è stato un aumento della congestione all'interno degli aeroporti.

Purtroppo la creazione di nuove risorse spesso non è possibile in tempi brevi, ad esempio per cause finanziarie o tecniche. Per limitare la congestione e i suoi effetti si è quindi reso necessario sfruttare nel modo più accurato possibile le risorse disponibili. Questo è quanto accade nella fase di parcheggio degli aerei.

La tesi si occupa della ricerca della migliore schedulazione delle piazzole di sosta ai vari aerei che transitano per un aeroporto. La scelta della piazzola deve rispettare vincoli di natura

- *fisica*: una piazzola ha portata e dimensione limitate;
- *tecnica*: la piazzola prescelta deve consentire di servire l'aereo per tutte le operazioni necessarie;
- *contrattuale*: la piazzola prescelta deve consentire di servire l'aereo per tutte le operazioni richieste dalla compagnia aerea.

La sosta di un aereo in una piazzola comporta un guadagno per l'ente aeroportuale. In base alle caratteristiche della piazzola, del contratto e degli obiettivi dell'ente, sono previsti differenti guadagni per ogni coppia sosta - piazzola. L'interesse dell'ente è quello di massimizzare il guadagno globale, cioè la somma dei guadagni dalle soste effettuate da tutti gli aerei transitati presso l'aeroporto.

In letteratura il problema è stato ampiamente studiato anche a partire da diversi punti di vista. Infatti sono stati formulati e studiati molti problemi e metodi risolutivi. Alcuni studi relativi a ammissibilità, modellazione, tecniche risolutive esatte o approssimate sono stati analizzati prima di procedere ai test relativi alla singola situazione di interesse.

Solo una parte di quanto presente in letteratura fa riferimento a situazioni reali. Questa tesi ha cercato di coniugare aspetti teorici ad una situazione che ricalcasse quella reale.

Indice ragionato

Nel capitolo 2 ci si sofferma nell'inquadrare il problema reale e di come viene trattato dalla comunità scientifica. Si riscontra che l'assegnazione di piazzole di sosta ad aerei è modellata tramite l'*interval scheduling problem*. Si vedrà inoltre come tale tipologia di problemi viene spesso utilizzata nell'ambito aeroportuale tramite opportune varianti che coinvolgono vincoli e/o funzione obiettivo.

Nel capitolo 3 verrà presentato il modello relativo alla situazione di interesse. Si proporrà la formulazione matematica e le relative modifiche, anche dettate da questioni implementative.

I dettagli circa l'implementazione degli algoritmi verranno presentati nel capitolo 4. Ampio spazio sarà dato alla generazione di istanze. Inoltre verrà presentato il *simulated annealing*, algoritmo metaeuristico utilizzato nei test.

I capitoli 5, 6 e 7 presenteranno i risultati dei test e i relativi commenti.

Le considerazioni finali e i possibili sviluppi successivi alla tesi verranno discussi nel capitolo 8.

Capitolo 2

Introduzione al problema

2.1 Interesse del problema

Il problema dell'assegnazione delle piazzole di sosta in un'aeroporto ha notevole importanza. Infatti durante le soste vengono compiute le operazioni di rampa (*ground handling operations*). Queste sono sempre comprese tra due voli successivi. Si compongono di:

- guida dell'aeromobile al parcheggio;
- posizionamento tacchi e connessione al gruppo energetico esterno;
- posizionamento delle scale o del loading bridge;
- sbarco passeggeri;
- scaricamento stive;
- verifiche tecniche;
- preparazione dei documenti per il volo in partenza;
- rifornimento carburante;
- pulizie di bordo;
- allestimento catering;
- caricamento delle stive;
- imbarco dei passeggeri;
- allontanamento delle scale e/o del loading bridge;

- pushback e/o avviamento motore.

La durata della sosta dipende dalle caratteristiche dell'aereo, del volo appena effettuato, di quello da effettuare e delle operazioni previste. In quest'arco di tempo vengono eseguite le varie operazioni.

Gli accordi tra le compagnie e la società gestore dell'aeroporto prevedono che quest'ultima riceva compensi in base al tipo di operazioni effettuate. Esistono varie tipologie di piazzole, ognuna con proprie caratteristiche come dimensioni e operazioni effettuabili. Per ogni tipologia di piazzole è quindi previsto un compenso.

Il gestore dell'aeroporto è interessato a ricavare il massimo guadagno, tenendo conto delle possibilità delle piazzole a disposizione e del fatto che debbono essere serviti tutti gli aerei.[1]

A questo scopo ben si adatta una modellazione tipica della ricerca operativa. Il problema da affrontare appartiene all'ambito del *job scheduling problem*, nel caso specifico si andrà a trattare l'*interval scheduling problem*. La caratteristica distintiva di quest'ultimi è che i tempi di inizio e di terminazione dei job sono conosciuti a priori e, se il job viene schedulato, devono essere rispettati.

2.2 Letteratura presente

2.2.1 Interval scheduling problem

Il problema della schedulazione di job a machine è di vitale importanza per molte organizzazioni e aziende. E' anche per questo motivo che la comunità scientifica affronta da molto tempo il problema. Estremamente importante è la situazione *non preemptable*, in cui non è possibile interrompere l'esecuzione di un job per processarne un altro sempre tramite la stessa machine. La naturale formulazione nei termini tipici della ricerca operativa ha portato già negli anni '50 nomi come Dantzig e Fulkerson [1] ad interessarsene. Qualche anno più tardi Ford e Fulkerson [2] risolsero utilizzando il teorema di Dilworth il *basic interval scheduling problem*. Tale problema viene così espresso:

Problema 2.2.1 (Basic interval scheduling problem) *Siano*

- $I = \{[s_i; f_i)\, i = 1, \ldots, n\}$ *l'insieme dei job (s_i e f_i indicano gli istanti discreti di inizio e fine del job i)*
- J *l'insieme delle machine.*

[1] In alcuni casi non trattati in questa tesi alcune operazioni possono non essere eseguite riducendo così i tempi di servizio.

Trovare una schedulazione dei job alle machine tale che:

- *il numero di machine richieste (cioè $|J|$) sia minimo*
- *job con tempi di esecuzione intersecati, cioè con istanti di inizio precedenti a quelli di terminazione di entrambe, non siano eseguiti nella stessa machine.*

Definizione 2.2.1 (Job overlap) *Si definisce* job overlap *la cardinalità del più grande sottoinsieme di job tale che tutti i job in esso contenuto abbiano i tempi di esecuzione intersecati tra di loro.*

Nell'esempio mostrato nella figura 2.2.1 il *job overlap* vale 4 e si ha nell'intervallo di tempo $[2; 3)$. Infatti durante tale intervallo i job 1, 2, 3 e 4 sono contemporaneamente in esecuzione.

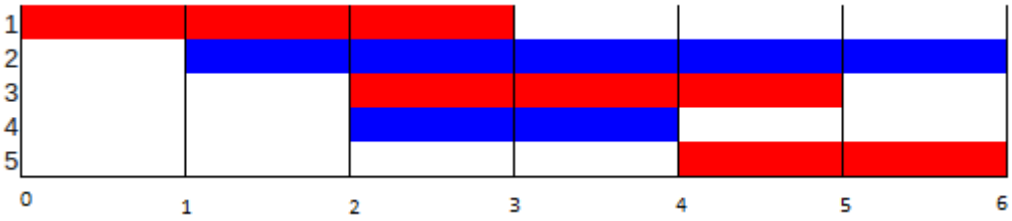

Figura 2.1: Esempio di job overlap.

Il risultato di Ford e Fulkerson fu di dimostrare che il *job overlap* per il problema 2.2.1 non è solo un lower bound, ma è il valore ottimo. La dimostrazione è basata principalmente sul teorema di Dilworth.

2.2.2 Varianti del basic interval scheduling problem[3]

Modificando vincoli e/o funzione obiettivo del problema 2.2.1 sono stati sviluppati differenti problemi. Questi hanno l'obiettivo di rispondere a particolari esigenze, come ad esempio la formazione di team.

Alcune varianti sono:

Problema 2.2.2 (Interval scheduling with required jobs) *Minimizzare il costo di schedulazione a patto che tutti i job vengano eseguiti. Il costo di schedulazione è dato dalla somma dei costi delle assegnazioni dei job alle machine, quindi per ogni coppia job - machine è previsto un apposito costo. Le machine sono differenti tra di loro.*

Le differenze tra machine possono essere di differenti tipologie. Ad esempio possono variare per l'intervallo di disponibilità delle varie machine (*Interval scheduling with machine availability*[3]) o per il tipo di job che possono essere processati. Caso appartenente a quest'ultima tipologia è il *Hierarchical interval scheduling*[3]. Le machine vengono ordinate per numero crescente da 1 a $|J|$ e il job i può essere eseguito solo dalle machine con indice non superiore a $m_i \in \{1, \ldots, |J|\}$.

Problema 2.2.3 (Interval scheduling with given machine) *Il numero di machine è fissato e per ogni coppia job - machine vi è un specifico profitto. Si deve massimizzare il profitto totale dato dall'assegnazione dei job alle machine. Non è necessario che tutti i job vengano schedulati.*

E' stato provato da Arkin e Silverberg [4] che se alcuni job non possono essere schedulati su alcune machine (cioè si è in presenza di *incompatibilità*) allora il problema è NP-hard.

Il problema 2.2.3 è interessante perchè un rilassamento del problema trattato nei test ne è un caso particolare.

Problema 2.2.4 (Discrete interval scheduling) *Rispetto al problema 2.2.1 si differenzia per il fatto che per ogni job è disponibile un insieme finito di possibili inizi. Cioè il job i può iniziare ad essere eseguito solo a partire da un istante* $s \in S_i = \{s_{1_i}, \ldots, s_{k_i}\}$

2.2.3 Applicazioni alle realtà aeroportuali

Kolen e Kroon [7] hanno modellato il problema in termini di classi di job e classi di machine. Il tipo di schedulazione rimane non preemptive e una machine può processare solo un job alla volta. L'utilizzo delle classi è più conveniente in questo studio che riguarda l'assegnazione di ingegneri ad aerei per le operazioni di manutenzione. Gli ingegneri possiedono varie tipologie di licenze e quindi possono operare solo su determinati tipi di aerei. Gli ingegneri vengono quindi modellati come machine mentre la machine class coincidono con le tipologie di aerei sulle quali un ingegnere con determinate qualifiche può operare. Agli intervalli in cui operare su di un specifico aereo vengono invece fatti corrispondere i job.

I problemi interessati dallo studio presentato in [7] sono due.

Problema 2.2.5 (CS(L)) *Siano*

- *J il numero di job*

- $I = \{(s_i, f_i, a_i)\, i = 1, \ldots, J\}$ *insieme di job dove* a_i *indica la classe del job* i
- C *machine class in cui la machine class* $c = 1, \ldots, C$ *ha* M_c *machine.*
- L *matrice delle compatibilità tra i job della classe* a *e le machine della classe* c *dove*

$$L_{ac} = \begin{cases} 1 & \textit{se } a \textit{ compatibile con } c \\ 0 & \textit{altrimenti} \end{cases}$$

Verificare se esiste uno scheduling non preemptive di tutti i job.

Problema 2.2.6 (MCS(L)) *Siano*

- J *il numero di job*
- $I = \{(s_i, f_i, a_i, v_i)\, i = 1, \ldots, J\}$ *insieme di job dove* a_i *indica la classe del job* i *e* v_i *indica il profitto collegato al job* i
- C *machine class in cui la machine class* $c = 1, \ldots, C$ *ha* M_c *machine.*
- L *matrice delle compatibilità tra i job della classe* a *e le machine della classe* c*dove*

$$L_{ac} = \begin{cases} 1 & \textit{se } a \textit{ compatibile con } c \\ 0 & \textit{altrimenti} \end{cases}$$

Trovare lo soluzione di massimo profitto, anche a scapito di non schedulare alcuni job.

Il problema 2.2.5 è volto alla fattibilità della schedulazione di tutti i job. E' un caso particolare del problema 2.2.2 dove la differenza tra le machine è data dalle diverse compatibilità con un job.

L'obiettivo del problema 2.2.6 è invece di massimizzare il profitto anche a scapito di non schedulare alcuni job. Tale problema è un caso particolare del problema 2.2.3 poichè vi sono sempre le incompatibilità.

Arkin e Silverberg [4] hanno dimostrato come, sotto certe condizioni, il problema 2.2.5 sia NP-completo. Per farlo si sono ricondotti ad un problema similare in cui inizio e fine dei job sono fissati e le compatibilità sono tra job e machine anzichè tra job class e machine class. Inoltre hanno presentato un algoritmo di programmazione dinamica che in tempo $O(J^{\sum_{c=1,\ldots,C} M_c+1})$ fornisce la soluzione ottima per il problema 2.2.6.

Complessità di CS(L)

Definizione 2.2.2 (L riducibile) *La matrice delle compatibilità L si dice riducibile se sussiste almeno una delle seguenti condizioni.*

1. *Esiste una colonna o una riga di L formata solamente da zeri.*
2. *Esistono due colonne o due righe di L identiche.*
3. *Attraverso una permutazione su righe e colonne è possibile riscrivere L come*
$$\begin{bmatrix} L_x & 0 \\ 0 & L_y \end{bmatrix}$$
con L_x sottomatrice di L_y

Definizione 2.2.3 (L irriducibile) *Se L non è riducibile allora è irriducibile.*

Teorema 2.2.1 (Complessità di CS(L)) *Sia L irriducibile. CS(L) è NP-completo $\iff$ L ha almeno tre colonne.*

Dimostrazione

only if La matrice
$$L_2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{pmatrix}$$
è, a meno di permutazioni su righe e colonne, la più grande matrice irriducibile con al più due colonne. Essendo CS(L_2) risolvibile in tempo polinomiale [7] allora la conseguenza logica è dimostrata.

if

Teorema 2.2.2 *Sia L irriducibile con almeno tre colonne. Attraverso permutazioni su righe e colonne si ricava che almeno uno tra L_3, L_4 e L_5 è sottomatrice di L.[7]*

Teorema 2.2.3 *CS(L_3), CS(L_4) e CS(L_5) sono NP-completi.[7]*

Lemma 2.2.1 *Date L_x e L_y tali che L_x è sottomatrice di L_y attraverso permutazioni su righe e colonne allora se CS(L_x) è NP-completo allora CS(L_y) è NP-completo. [7]*

Per i teoremi 2.2.2 e 2.2.3 e il lemma 2.2.1 risulta pertanto che se la matrice L ha almeno tre colonne allora CS(L) è NP-completo.

Complessità di MCS(L)

Teorema 2.2.4 (Complessità di MCS(L)) *Sia L irriducibile. MCS(L) è NP-hard* $\iff$ *L ha almeno due colonne.*

Dimostrazione

only if Se L ha solo una colonna allora

$$L = L_0 = (1)\,.$$

Poichè MCS(L_0) può essere risolta in tempo polinomiale ([4]) allora questa parte del teorema è dimostrata.

if se L ha almeno due colonne allora è possibile ricavare tramite permutazioni su righe e colonne la matrice

$$L_1 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}.$$

Poichè valgono il teorema 2.2.5 e il lemma 2.2.2 allora il problema è NP-hard.

Teorema 2.2.5 *MCS(L_1) è NP-hard.[7]*

Lemma 2.2.2 *Date L_x e L_y tali che L_x è sottomatrice di L_y attraverso permutazioni su righe e colonne allora se MCS(L_x) è NP-hard allora MCS(L_y) è NP-hard. [7]*

Studio sul *tactical fixed interval scheduling problem*[8]

Problema 2.2.7 (Tactical fixed interval scheduling problem) *Minimizzare il numero di machine per schedulare un insieme di job appartenenti a determinate classi. Devono essere rispettati i seguenti vincoli*

- *Ogni machine non può processare più job nello stesso istante.*
- *Ogni machine può operare solo su determinate classi di job.*
- *Non è consentita preeemption.*

Il problema 2.2.7 si differenzia dal 2.2.1 per il fatto che quest'ultimo, essendo il caso più generale possibile, non prevede le classi di job.

Teorema 2.2.6 *Se il numero di machine class è superiore a 2 allora il problema 2.2.7 è NP-hard, a meno di casi particolari.[9]*

Kroon et al. [8] hanno applicato il problema 2.2.7 per la gestione dei team di lavoro in ambito aereoportuale.

Tipologia di istanze Lo studio presenta i risultati dei test compiuti su:

- istanze create a partire da procedure non deterministiche;
- istanze che rappresentino situazioni vicine alla realtà.

La generazione delle istanze è basata principalmente sui seguenti parametri.

- Numero di job class.
- Numero totale di job.
- Durata massima dei job.
- Scelta delle job class e degli istati di inizio e fine. Sono state testate due tipologie di creazioni.
  - Per ogni job, la job class e l'istante d'inizio vengono scelti a partire da due distribuzioni uniformi.
  - Per ogni job, la job class è determinata tramite una distribuzione uniformemente distribuita mentre l'istante di inizio è determinato a partire da una normale con parametri determinati dal tipo di job class scelta.
- Disponibilità delle machine class. Sono state testate due tipi di disponibilità:
  - inesistenza di incompatibilità;
  - le machine di una classe possono processare solo due classi di job, machine class differenti hanno in comune al più una classe di job sulla quale possono operare e per ogni job class ci sono solo due classi di machine che la possono operare.

Test e risulati Sono stati approntati vari tipi di rilassamenti e di soluzioni euristiche. I rilassamenti sono quasi tutti per eliminazione di vincoli sul problema, solo uno è un rilassamento lagrangiano ed interessa il vincolo di esecuzione di un job su di una sola machine class[2]. I rilassamenti per eliminazione escludono totalmente dal modello uno o più vincoli, quelli lagrangiani spostano il vincolo, pesato secondo dei moltiplicatori, nella funzione obiettivo. Dalla definizione dei rilassamenti e a parità di vincolo trattato, si ricava che:

[2]Per come è stato definito il problema il vincolo garantisce anche che il job sia eseguito su di una sola machine

- il rilassamento per eliminazione è un caso particolare di quello lagrangiano in cui i moltiplicatori sono tutti nulli;

- il rilassamento lagrangiano fornisce risultati non peggiori di quello per eliminazione.

Le soluzioni euristiche studiate invece sono due. Una è basata su di un algoritmo greedy, l'altra sull'utilizzo di più coperture feasible di sottoinsiemi di job. Una schedulazione si dice feasible per il problema in oggetto se rispetta i tre vincoli espressi nella definizione del problema stesso. In generale una soluzione viene detta feasible se rispetta i vincoli del modello.

Il risultato più importante dello studio è stato dimostrare che la programmazione lineare, in combinazione con una procedura branch and bound, risulta essere efficace sia come qualità dei risultati che come tempi di esecuzione se la taglia delle istanze è di media dimensione. Tale situazione rispecchia quella reale, pertanto tali procedure possono essere inserite nei DSS di aeroporti e compagnie aeree.

Studio sul *operational fixed interval scheduling problem*[10]

Problema 2.2.8 (Operational fixed interval scheduling problem) *Si massimizzi il profitto dato dalla schedulazione di un insieme di job o di una parte di essi rispettando i vincoli seguenti.*

- *Ogni machine non può processare più job nello stesso istante.*

- *Le machine possono operare solo su determinate classi di job.*

- *Non è consentita preeemption.*

In [10] sono stati compiuti dei test sia su istanze random che su istanze che fossero simili alle realtà aeroportuali. Il test ha interessato sia lower bound che upper bound del problema.

I valori registrati come tempi e qualità delle soluzioni sono stati pienamente soddisfacenti quando il carico di lavoro e il numero di machine erano basse. Nelle situazioni più vicine alla realtà i test hanno dato esiti più soddisfacenti rispetto all'altra tipologia di istanze. Le soluzioni risolvendo il problema come se fosse di programmazione lineare hanno dato buoni risultati, infatti generalmente coincidevano con l'ottimo intero.

2.2.4 On-line scheduling problem

Tutti i problemi presentati prima sono *off-line*. I problemi *on-line* si differenziano per il fatto che lo scheduler viene a conoscenza dell'esistenza di un job al momento di inizio di tale job. Lipton e Tomkins [5] hanno studiato il caso in cui si cerchi di massimizzare il tempo di occupazione delle machine con i seguenti vincoli:

- job con intervalli di esecuzione intersecati non siano eseguiti nella stessa macchina
- un job schedulato non può essere interrotto.

Lo studio si è concentrato sulla approssimabilità dell'ottimo dimostrando che non è possibile di riuscire a fare meglio di un fattore $O(log\Delta)$. Δ è il rapporto tra il massimo intervallo di esecuzione e il minimo intervallo di esecuzione tra i job da schedulare.

Le versioni preemptive dei problemi *on-line* prevedono che ogni qual volta ad un job i in esecuzione venga sottratta la machine che sta usando per consentire ad un altro job di essere eseguito, allora il job i deve essere subito eseguito su di un'altra machine. Questo ha l'obiettivo di impedire che il job i non venga completato entro il termine previsto.

Studi nell'ambito dell'on-line scheduling sono stati compiuti anche da Fischetti et al. [6].

2.3 Estensioni date dal progetto

Nel lavoro di tesi ci si è riferiti a concetti per lo più conosciuti. Il lavoro si differenzia da altri studi per l'applicazione di tali concetti al specifico caso di interesse. La generazione delle istanze aveva proprio l'obiettivo di ricondursi a situazioni tipiche della realtà aeroportuale di riferimento. L'obiettivo era ottenere situazioni in cui vi fosse una gamma sufficientemente ampia di scelte per ogni job nonostante quest'ultimi fossero in numero elevato. Le istanze generate sono poi state oggetto di test.

Tutto il lavoro di tesi, ed ancor di più i test, è stato compiuto sullo stile visto in [8]. Le differenza di obiettivo con questo studio consente solo un limitato confronto tra i risulati.

Capitolo 3

Analisi del problema

Nel capitolo viene presentato il modello relativo al problema di riferimento. Ne viene discussa la complessità e analizzata una tecnica per diminuire la dimensione del modello. Nella parte conclusiva si introducono modifiche al modello dovute alle necessità di implementazione su supporto informatico.

3.1 Il modello

Ci si riconduce all'interval scheduling problem facendo corrispondere:

- soste degli aeromobili ai job;
- classi di parcheggi alle machine class.

Si deve massimizzare il profitto dato dall'assegnazione dei job a una machine di una classe compatibile. Machine della stessa classe vengono considerate come indistinguibili. Tale obiettivo viene garantito dalla formula 3.1.

Si deve imporre che:

1. tutti i job vengano eseguiti;
2. la richiesta di machine di una determinata classe non superi mai la disponibilità massima di tale classe. Quindi in ogni istante il numero di job che viene eseguito da machine di una determinata classe non deve superare il numero totale di machine della classe.

Il rispetto dei due vincoli viene garantito rispettivamente dalle formule 3.2 e 3.3.

3.2 Formulazione matematica

$$
\begin{aligned}
max \quad & \sum_{i \in I,\, j \in J_i} c_{ij} x_{ij} && && (3.1)\\
& \sum_{j \in J_i} x_{ij} = 1 && \forall i \in I && (3.2)\\
& \sum_{i' \in I | [s_{i'}; f_{i'}) \cap [s_i; f_i) \neq \emptyset} x_{i'j} \leq p_j && \forall i \in I,\, j \in J_i && (3.3)\\
& x_{ij} \in \{0, 1\} && \forall i \in I,\, j \in J_i && (3.4)
\end{aligned}
$$

dove

- I insieme dei job
- $\forall i \in I$
  - J_i insieme delle machine class compatibili con il job i
- J insieme delle machine class
- $\forall j \in J$
  - I_j insieme dei job compatibili con la machine class j
  - p_j numero di machine della classe j
- $\forall i \in I, j \in J$
  - c_{ij} guadagno ottenuto assegnando il job i ad una machine della classe j
    $c_{ij} = 0 \iff$ i e j non sono compatibili
  - x_{ij} variabile binaria che indica se il job i è stato assegnato ad una machine della classe j

$$
x_{ij} = \begin{cases} 1 & \text{se } i \text{ assegnato ad una machine della classe } j \\ 0 & \text{altrmenti} \end{cases}
$$

3.3 Complessità

Il modello presentato nella sezione 3.2 presenta

- $O(|I||J|)$ variabili, poichè vi sono $|I|$ job e per ognuno vi sono $O(|J|)$ machine class compatibili.
- $O(|I||J|)$ vincoli, poichè vi sono
  - $|I|$ vincoli di tipo 3.2;

- $|I|O(|J|)$ vincoli di tipo 3.3;
- $|I|O(|J|)$ vincoli di tipo 3.4.

Il modello risulta quindi essere polinomiale nella taglia dell'istanza.

In [7] viene riportato che se le machine class sono più di una allora la ricerca della soluzione di massimo guadagno è NP-hard. Inoltre nello studio si forniva una estensione per il caso in cui un insieme di job sia già allocato. Anche in questa situazione se le machine class sono più di una allora il problema è NP-hard.

3.4 Introduzione dell'essential partition nel modello

Definizione 3.4.1 (Aircraft set) *Si definisce* aircraft set *un insieme AS che è sottoinsieme dell'insieme dei job, cioè I. L'aircraft set associato all'intervallo* $[t_1; t_2)$ *è l'insieme di job* $AS(t_1, t_2) | \forall i \in AS(t_1, t_2)\ [s_i; f_i) \cap [t_1; t_2) \neq \emptyset$.

Definizione 3.4.2 (Essential partition) *Si definisce* essential partition *una partizione* $X = \{[s_0 = 0; f_0), [s_1; f_1), \ldots, [s_n; f_n = T)\}$ *di* $[0; T)$ *tale che* $\forall k \in \{1, \ldots, n\}$

1. $[s_a; f_a) \cap [s_b; f_b) \neq \emptyset\ \forall a, b \in AS(s_k, f_k)$
2. $\neg\exists j \in \{1, \ldots, n\} \setminus \{k\} |\ AS(s_j, f_j) \subseteq AS(s_k, f_k)$

Definizione 3.4.3 (Intervallo essenziale) *Gli intervalli temporali* $[s_k; f_k)\ k \in \{1, \ldots, n\}$ *che compongono l'essential partition X sono detti* intervalli essenziali.

Teorema 3.4.1 *Data* $X = \{[s_0 = 0; f_0), [s_1; f_1), \ldots, [s_n; f_n = T)\}\ \forall k \in \{1, \ldots, n-1\}$ *si ha che:*

1. $|AS(s_k, f_k) \setminus AS(s_{k+1}, f_{k+1})| \geq 1$
2. $|AS(s_{k+1}, f_{k+1}) \setminus AS(s_k, f_k)| \geq 1$

Dimostrazione Se il teorema non valesse allora l'aircraft set di uno dei due intervalli sarebbe contenuto in quello di un altro intervallo. Questo violerebbe la condizione 2 della definizione di essential partition.

3.4.1 Vantaggi

L'introduzione dell'essential partition consente di diminuire il numero di vincoli da prendere in considerazione. Il vincolo 3.3 è relativo ad ogni job mentre il vincolo

$$\sum_{i \in I_j \cap AS(s,f)} x_{ij} \leq p_j \, \forall [s; f) \in X, \, j \in J \tag{3.5}$$

è relativo ad ogni intervallo dell'essential partition. Essendo tali intervalli mediamente in quantità minore dei job si può pertanto affermare che la modifica al modello ne riduce la dimensione.

Al caso peggiore il numero di intervalli essenziali equivale a quello dei job. Tale situazione è da considerarsi triviale poichè corrisponde al caso in cui nessun job si interseca con un altro. La soluzione si ottiene quindi assegnando ad ogni job una postazione dalla machine class che comporta il maggiore guadagno per quel job.

3.4.2 Procedura di identificazione

Il riconoscimento degli intervalli essenziali dell'essential partition viene effettuato con una procedura di complessità nel caso peggiore quadratica nel numero di job.

I job vengono controllati per valore d'inizio crescente, nel caso di job ad uguale inizio viene controllato il job che termina prima. Per ogni job j si verifica l'esistenza di altri job che terminino prima del suo inizio. Questi job e tutti quelli da eseguire durante l'inizio di j verranno assegnati ad un nuovo intervallo. L'ultimo intervallo dell'essential partition è composto dal job che parte per ultimo e da tutti quelli da eseguire durante il suo inizio.

Pseudocodice Dato una lista di job vengono identificati gli intervalli dell'essential partition.

```
1 Input:
2       jobList←job ordinati secondo l'ordinamento:
            i < j ⟺ s_i < f_j ∨ s_i = s_j ∧ f_i < f_j
3 Corpo:
4       jobInEsecuzione← ∅
5       ∀ job∈ jobList
6             newIntervallo=false
7             ∀ j∈jobInEsecuzione
8                   if j.end≤job.start
9                         if ¬newIntervallo
```

```
10                                  crea(nuovoIntervallo)
11                                  AS_nuovoIntervallo ←jobInEsecuzione
12                                  newIntervallo=true
13                              jobInEsecuzione.remove(j)
14                      jobInEsecuzione.add(job)
15              crea(nuovoIntervallo)
16              AS_nuovoIntervallo ←jobInEsecuzione
```

Complessità Al caso peggiore nessun job termina prima che altri job siano iniziati. Questo comporta che all'i-esimo job si dovrà ripetere la verifica della terminazione per tutti gli $i-1$ job precedenti. Pertanto la complessità risulta:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \tag{3.6}$$

L'ordinamento dei job viene garantito nella procedura di creazione dei job. L'inserimento dei job nella lista risulta eseguibile con ordine di complessità non superiori a quella dell'esecuzione dell'algoritmo di identificazione. Infatti già nell'inefficiente caso in cui si controllasse ogni job da inserire con quelli già inseriti si otterrebbe il risultato della formula 3.6. Pertanto l'ordine di complessità rimane quadratico.

3.5 Elaborazioni del modello

Gli insiemi relativi alle compatibilità job - machine class, cioè $J_i(i \in I)$ e $I_j(j \in J)$, vengono rappresentati con la stessa matrice *admit*. Essa viene riempita secondo la seguente regola

$$admit_{ij} = \begin{cases} 1 & \text{se } i \text{ compatibile con } j \\ 0 & \text{altrimenti} \end{cases}$$

Le variabili binarie x_{ij} trattate dai programmi per la risoluzione del modello sono in numero maggiore rispetto a quanto definito nel modello. Infatti vi è una variabile per ogni coppia job - machine class. Quindi il vincolo 3.4 diventa

$$x_{ij} \in \{0,1\} \, \forall i \in I, \, j \in J \tag{3.7}$$

Avendo così definito le variabili x negli altri vincoli dato il job i non ci si limiterà alle machine class $j \in J_i$ ma ci si estenderà a quelle incompatibili. Questo comunque non compromette la correttezza del risultato ottenuto. Infatti per definizione se vi è incompatibilità tra il job i e la machine class j allora $x_{ij} = 0$.

Capitolo 4

Algoritmi risolutivi

In questo capitolo vengono esposti gli algoritmi e gli strumenti utlizzati per la risoluzione dei modelli. La parte di creazione delle istanze è quella dove si è concentrata la maggior parte dell'attività di implementazione.

Per la risoluzione dei modelli si è ricorsi a strumenti già presenti nel mercato, tranne per il simulated annealing. Infatti per quest'ultimo si è ricorsi ad una implementazione usata in precedenti lavori e riaddattata per il problema di riferimento. Viene pertanto presentato l'algoritmo e i parametri su cui si può agire. I dettagli più specifici relativi alla configurazione verranno affrontati nella sezione 6.1.3 dove si esporranno i test eseguiti.

Nel capitolo non è stata inserita una sezione relativa al post-processing vista la tipologia di test e il tipo di misure effettuate. In linea di principio si è semrpe cercato di raccogliere i dati necessari al singolo test dai file di output generati dalle esecuzioni di *Cplex* o del simulated annealing. Successivamente i dati sono stati analizzati e rappresentati tramite fogli elettronici.

4.1 Preprocessing

4.1.1 Generazione istanze

Parametri per la generazione di istanze

Ogni istanza viene generata a partire dai seguenti parametri.

1. Numero di machine class.
2. Numero minimo di machine per ogni machine class.
3. Numero massimo di machine per ogni machine class da aggiungere.
4. Numero di blocchi di job.

5. Numero minimo di job per blocco.
6. Numero massimo di job per blocco da aggiungere.
7. Distanza massima tra gli inizi di due job dello stesso blocco.
8. Probabilità che un job sia compatibile con una machine class.
9. Durata minima di un job.
10. Durata massima di un job.
11. Massimo guadagno dato dall'assegnazione di un job ad una machine class.[1]

Utilizzo del parametro 8 Dati

- x valore del parametro;
- $X \sim U(0;1)$

il job i è compatibile con la machine class $j \iff X \leq x$.

Per ogni coppia job - machine class viene generato un nuovo valore per X.

Istanze feasible

```
1  ∀ machine class
2        assegnazione id
3        assegnazione numero di machine
4  ∀ blocco di job
5        settaggio numero job nel blocco
6        ∀ job in blocco
7              assegnazione id
8              assegnazione start e finish
9              assegnazione machine class compatibili
10             ∀ machine class compatibile
11                   asssegnazione income
12 cancellazione indipendent subproblem
```

[1] Il guadagno minimo tra job e machine class compatibili per default vale 1.

Cancellazione indipendent subproblem Nella definizione del problema si deve verificare che non vi siano parti di esso che possano essere trattate singolarmente. Ad esempio ammettiamo che nell'intervallo $[0; 10)$ i job con numero d'ordine $i \in \{1, \ldots, 6\}$ abbiano il loro intervalo di esecuzione. Se i job 1, 2 e 3 possono essere eseguiti solo su machine di classe $c \in \{1, \ldots, 3\}$ mentre i job 4, 5 e 6 solo su machine di class $c^{'} \in \{4, \ldots, 7\}$ allora è possibile suddividere il problema in due sottoproblemi distinti (*indipendent subproblem*). Infatti la schedulazione dei primi tre job non influenza quellla degli altri tre e viceversa.

Definizione 4.1.1 (Indipendent suproblem) *Sia IS una istanza di* interval scheduling problem *con I insieme dei job. Se $\exists X \in 2^I | \forall Y \in X | \neg \exists Z \in X \setminus \{Y\} | \exists y \in Y, z \in Z | [s_y; f_y) \cap [s_z; f_z) \neq \emptyset \wedge J_y \cap J_z \neq \emptyset$ allora è possibile definire per ogni insieme Y di X un indipendent subproblem IS_Y che avrà come insieme dei job Y stesso.*

Ogni indipendent subproblem può essere trattato singolarmente. Pertanto si otterranno due o più problemi di taglia più piccola.

Descrizione della cancellazione Poichè interessano istanze senza indipendent subproblem si è resa necessaria la loro cancellazione nel caso la procedura di creazione istanze li rilevasse. Questa fase si compie in due passi.

1. Garantire che vi sia sempre almeno un job in esecuzione. Il tutto viene garantito dalla regola:
   Dati i job i e $i^{'} | \, f_i \leq s_{i'}$
   se $\neg \exists$ job $i^{''} | \, [f_i; s_{i'}) \cap [s_{i''}; f_{i''}) \neq \emptyset \Longrightarrow s_{i'} = f_i - 1$.
   La regola impedisce che ci siano job che non si intersechino con un altro job. Così facendo è sempre vero che ci sono almeno due job in esecuzione. Senza perdita di generalità, ci si riferisce a istanti discreti perciò il decremento di uno garantisce l'intersezione tra i job i e $i^{'}$.

2. Garantire la connessione del grafo $G = (V; E)$ con $V = I$ e $E = \{(i, i^{'}) | \, [s_i; f_i) \cap [s_{i'}; f_{i'}) \neq \emptyset \wedge J_i \cap J_{i'} \neq \emptyset\}$. L'insieme E è quindi formato da archi in cui due nodi, che corrispondono a job, sono collegati se sono vere le due condizioni seguenti

   - hanno tempi di esecuzione intersecati;
   - potrebbero essere eseguiti su machine della stessa classe.

   Il tutto viene garantito dalla seguente procedura:

```
1  Input:
2       G=(V,E) rappresentato come una matrice M di
           dimensione |I| × |I| dove M_{i,i'} = 1 ⟺ (i, i') ∈ E e
            0 altrimenti
3
4  Corpo:
5       //verifica che tutti i nodi siano
           raggiungibili, ovvero che ogni job possa
           avere almeno un conflitto con un altro job
6       crea nuova coda A
7       A.put(primo job a partire)
8       do{
9            i ← A.pop()
10           segnala i come visitato
11           ∀j|(i, j) ∈ E ∧ j non segnalato come visitato
12                A.put(j)
13      }while A ≠ ∅
14      //se tutti i nodi sono raggiungibili allora
           termina la procedura dato che non vi sono
           indipendent subproblem
15      if ¬∃ nodi non visitati
16           termina la procedura
17      //se non tutti i nodi sono raggiungibili rende
            raggiungibili quelli non raggiunti e
           collegabili a nodi raggiunti
18      ∀i ∈ V|i non visitato ∧∃i' ∈ nodi visitati
           |(i, i') ∈ E
19           ∀ machine class j|i e j non compatibili
20                rendi i e j compatibili con guadagno
                       1
21      //la procedura deve essere ripetuta dato che
           non tutti gli indipendent subproblem posson
            esser stati cancellati
22      ripeti la procedura sull'istanza modificata
```

Complessità La complessità è dominata dal passo 2 della procedura di cancellazione indipendent subproblem. Questa potrebbe essere ripetuta un numero non superiore a $|I|$ di volte. Per la tipologia di istanze da generare comunque non ha impedito la generazione delle istanze. Una esecuzione singola della procedura comporta una complessità $O(|I|^2|J|)$. A incidere è la

parte di verifica ed eventuale correzione delle compatibilità tra job e machine class. Infatti per ogni job si verifica che non sia raggiungibile, se non lo è si cercano i job a lui collegati e per ogni machine class si crea una compatibilità. La prima parte della procedura invece viene compiuta in tempo $O(|I|^2)$.

Quanto presentato fa riferimento ad una implementazione non estremamente efficiente in modo da ottenere un bound pessimistico. Utilizzando opportune strutture dati, ad esempio liste, per memotizzare le relazioni di job e machine si possono apportare miglioramenti allla complessità.

Considerazioni Per la tipologia di istanze da generare, le condizioni raramente non sono rispettate. Pertanto si può ricorrere alle azioni risolutive presentate senza che la feasibility e i tempi di esecuzione ne risentano.

La restante parte della procedura di generazione dell'istanza comporta una complessità $O(|I||J|)$ dovuta al fatto che per ogni coppia job - machine class ne viene decisa la compatibilità. Essendo rari i casi in cui si manifestano indipendent subproblem si può pertanto affermare che la procedura di generazione istanze è realmente dominata da questa fase.

Istanze infeasible

Per le istanze infeasible si ricorre all'inserimento di un specifico insieme di job. Le intersezioni tra i vari job e le compatibilità con le machine class saranno tali da rendere infeasible l'istanza. L'inserimento dei job avviene tra due blocchi di job feasible scelti casualmente. La casualità della scelta è dovuta al fatto che tali blocchi non determinano la infeasibilty.

Sono state creati degli insiemi di job specifici per testare differenti situazioni. Principalmente si hanno due categorie di insiemi:

1. insiemi senza pattern ripetibile;

2. insiemi a pattern ripetibile.

La categoria 2 è contraddistinta dalla presenza un sottoinsieme dei job ripetibile contiguamente una o più volte. Nell'altra categoria invece non è presente un pattern ripetibile.

Le machine class interessate nell'infeasibility possono essere compatibili anche con altri job.

4.1.2 Generazione file eseguibili per GAMS

GAMs è un software utilizzato in ambito matematico principalmente per problemi di ottimizzazione. Si avvale di un compilatore e di un solver. Il

compilatore consente di tradurre il file di input del problema in un file utilizzabile dal solver previsto. Questo consente di scrivere il problema in un modo il più ad alto livello possibile. Sarà poi il compilatore a tradurre nel modo più conveniente per l'esecuzione, perciò si ha la massima indipendenza dal solver utilizzato. Infatti GAMS è integrabile con molti solver. Questo consente di risolvere problemi di taglia e complessità elevata in tempi più previ di quanto possa fare il solver standard.

Nel corso della tesi si è utilizzata l'installazione di GAMS presso il server dell'Università degli Studi di Padova a Vicenza. Essa era integrata con l'installazione di *Cplex* installata nello stesso server.

Data un'istanza vengono generati due file eseguibili per GAMS. Uno è relativo al modello senza essential partition, l'altro al modello che ne fa uso. Per la creazione di quest'ultimo viene utilizzata la procedura presentanta nella sezione 3.4.2.

Modello che non usa essential partition Prevede:

- Indici
  - i indice dei job;
  - j indice delle machine class.
- Parametri
  - p_j numero di machine dalla classe j;
  - $admit_{ij}$ indica se il job i può essere elaborato in una machine della classe j;
  - $income_{ij}$ guadagno dato dall'assegnazione del job i ad una macchine della classe j;[2]
  - $overlap_{ii'}$ indica se alla partenza del job i il job i' è in esecuzione;[3]
- Variabili
  - z valore della funzione obiettivo
  - x_{ij} variabile binaria che indica se il job i è stato assegnato ad una machine della classe j
- Equazioni

[2]Per quanto indicato nella sezione 3.2 $income_{ij} = 0 \iff admit_{ij} = 0$

[3]Per come sono state impostate le equazioni si deve porre $overlap_{ii} = 1$

- *funzione obiettivo* da massimizzare;
- $esecuzioneJob_i$ garantisce che il job sia assegnato ad una sola machine;
- $disponibilità_{ij}$ impedisce che durante l'esecuzione del i la richiesta effettiva di machine della classe j ne superi la disponibilità massima;

Modello che usa essential partition Rispetto all'altro modello si differenzia per:

- aggiunta dell'indice r per scorrere gli essential partition
- aggiunta del parametro eas_{ri} per indicare se l'intervallo essenziale r contiene il job i
- eliminazione del parametro $overlap_{ii'}$ poichè tramite i parametri di tipo eas si ottiene l'indicazione temporale di intersezione tra job
- modifica all'equazione *disponibilità* con la sostituzione dell'indice i con l'indice r riducendosi così a controllare il non superamento delle disponibilità di machine per i soli intervalli essenziali dell'essential partition.

4.2 Processing

4.2.1 Cplex

Cplex è un software proprietario di ottimizzazione attualmente gestito da IBM. E' capace i risolvere problemi di ottimizzazione lineare anche intera e di dimensioni elevate. Più recentemente è stata aggiunta la possibilità di risolvere problemi di ottimizzazione quadratica. Sono previste delle interfacce verso linguaggi di programmazione e altri software di ottimizzazione.

Si è utilizzata la versione 7.5.0 di *Cplex* installata presso il server dell'Università degli Studi di Padova a Vicenza. Attualmente l'ultima versione stabile di Cplex è la 12.2. Si prevede che, a parità di modello, ricorrendo a quest'ultima i tempi di esecuzione possano ridursi di almeno la metà.

Tutti i test relativi a istanze feasible, sia nel caso del modello che fornisce la soluzione ottima che per i rilassamenti, e instanze infeasible sono stati compiuti con questo solver.

4.2.2 Simulated annealing

E' stato utilizzato a scopo di confronto dei risultati nel caso di ricerca di soluzioni euristiche. La differenza rispetto alla soluzione con il modello matematico è che il simulated annealing può essere interamente sviluppato ed eseguito senza ricorrere a soluzioni a pagamento. Infatti l'implementazione presentata in questa tesi è stata realizzata ricorrendo alle sole principali librerie previste per il linguaggio di programmazione Java.

Analogia con il processo di ricottura dei metalli[11]

Il simulated annealing è basato sulla statistica meccanica e sull'analogia con il processo di ricottura dei metalli. Fu sviluppato originariamente da Kirkpatrick et al. [12] per risolvere problemi di ottimizzazione combinatoria e discreta.

E' nato come metodo di simulazione della ricottura dei solidi. L'annealing è il processo con il quale un solido, portato allo stato fluido mediante riscaldamento ad alte temperature, viene riportato poi di nuovo allo stato solido o cristallino, e quindi a temperature basse, controllando e riducendo gradualmente la temperatura. Ad alte temperature, gli atomi nel sistema si trovano in uno stato altamente disordinato e quindi l'energia del sistema è elevata. Per portare tali atomi in una configurazione cristallina (statisticamente) altamente ordinata, deve essere abbassata la temperatura del sistema. Riduzioni veloci della temperatura possono causare difetti nel reticolo cristallino con conseguente metastabilità, con fessurazioni e fratture del reticolo stesso (stress termico).

L'annealing evita questo fenomeno procedendo ad un graduale raffreddamento del sistema, portandolo ad una struttura globalmente ottima e stabile. Il sistema si dice essere in equilibrio termico alla temperatura T se la probabilità $P(E_i)$ di uno stato avente energia E_i è governata dalla distribuzione di Boltzmann

$$P(\mathbf{E_i} = E_i) = \frac{1}{Z(T)} e^{-\frac{E_i}{KT}} \tag{4.1}$$

ove T è la temperatura, K è la costante di Boltzmann e $Z(T)$ è un fattore di normalizzazione dipendente da T.

Si noti che ad alte temperature tutti gli stati di energia sono probabilmente possibili, mentre a basse temperature il sistema si trova sicuramente in stati di minima energia.

Metropolis nel 1953 [13] sviluppò un algoritmo per simulare il comportamento di una collezione di atomi in equilibrio termico ad una particolare temperatura. Questo algoritmo ha un ruolo fondamentale per l'applicazio-

SISTEMA FISICO	PROBLEMA DI OTTIMIZZAZIONE
Stato	Soluzione ammissibile
Stato fondamentale	Soluzione ottima
Energia	Costo
Raffreddamento	Ricerca locale

Tabella 4.1: Corrispondenza tra gli elementi del sistema fisico e quelli del problema di ottimizzazione

ne del simulated annealing a problemi di ottimizzazione. Senza perdita di generalità, si consideri un problema di minimizzazione. La caratteristica essenziale dell'algoritmo di Metropolis è che genera un insieme di configurazioni a ogni temperatura T con la proprietà che le energie delle differenti configurazioni possono essere rappresentate dalla distribuzione di Boltzmann. Il metodo comincia da una assegnata configurazione iniziale degli atomi in un sistema con energia E_0. Vengono quindi generate successive configurazioni con piccole perturbazioni casuali della configurazione corrente. Viene deciso se accettare o rigettare la configurazione in base alla differenza fra l'energia della configurazione corrente e quella della nuova configurazione (detta configurazione candidata). Tale decisione è influenzata dal fatto che le energie del sistema delle configurazioni accettate devono formare una distribuzione di Boltzmann se si è raggiunto l'equilibrio termico. L'algoritmo di Metropolis accetta sempre una soluzione candidata la cui energia E_j è inferiore a quella della configurazione corrente (E_i). Per contro se l'energia E_j della configurazione candidata è più grande di quella della configurazione corrente, allora il candidato è accettato con la seguente probabilità:

$$P = e^{-\frac{\Delta E}{T}} \tag{4.2}$$

dove $\Delta E = E_j - E_i$ e T è la temperatura.

Questo processo di *raffreddamento rapido* può essere visto come analogo all'ottimizzazione locale. Gli stati del sistema fisico corrispondono alle soluzioni di un problema di ottimizzazione combinatoria; l'energia di uno stato corrisponde al costo di una soluzione e la minima energia, o stato fondamentale corrisponde ad una soluzione ottima.

La gradualità del processo di raffreddamento consente di arrivare ad una struttura globalmente ottima e stabile, mentre un'eccessiva rapidità è in analogia con l'ottimizzazione locale.

La temperatura non ha alcun diretto analogo in ottimizzazione. Serve meramente come parametro di controllo che implicitamente definisce la regione dello spazio di stato esplorato dall'algoritmo in un particolare stadio. Ad

alte temperature, l'algoritmo può attraversare quasi tutto lo spazio di stato poiché pessime soluzioni vengono facilmente accettate. Quindi il comportamento si avvicina ad una random search poichè la ricerca salta da un punto all'altro dello spazio delle soluzioni. Lo scopo è di individuare le direzioni o le aree in cui è più probabile individuare l'ottimo globale. Successivamente, abbassando il valore della temperatura, l'algoritmo viene confinato in regioni sempre più ristrette dello spazio delle soluzioni dato che la distribuzione di Boltzmann collassa, avendo così basse probabilità di accettazione. A basse temperature il simulated annealing è simile ai metodi *steepest descent*. Le soluzioni vengono localizzate nella zona del dominio maggiormente promettente. Quando la temperatura è, teoricamente, allo zero assoluto nessuna transizione di stato può portare verso uno stato a più alta energia. Quindi, come nell'ottimizzazione locale, sono proibiti movimenti in salita e le conseguenze di ciò possono essere indesiderabili.

Caratteristiche dell'algoritmo

L'algoritmo presenta le seguenti caratteristiche.

- Adatto sia a problemi continui che a problemi discreti.
- *Di tipo local search*, ovvero ad ogni iterazione si concentra su di un'unica soluzione candidata. L'esecuzione dell'algoritmo viene associata ad una traiettoria che ricalca l'ordine con cui le possibili soluzioni vengono considerate.[14]
  Tale traiettoria dipende da:
  - implementazione dell'algoritmo;
  - rappresentazione del problema;
  - istanza del problema.
- *Memory less*, ovvero non fa riferimento a risultati di iterazioni precedenti. Ad ogni iterazione la successiva soluzione da valutare dipende solo dalla soluzione attuale e dal suo vicinato, pertanto il comportamento dell'algoritmo è descrivibile da una processo di Markov[15]. La teoria di tali processi consente di concludere che l'algoritmo converge alla soluzione ottima ma purtroppo solo asintoticamente[11]. Tuttavia esistono delle implementazioni alternative che ottengono speed-up dall'uso di informazione pregressa, pagando però un prezzo maggiore in quanto ad utilizzo di memoria.

Il simulated annealing è un algoritmo *metaeuristico*. Questa tipologia di algoritmi lavora in contesti di assenza di informazione sul dominio del problema. L'obiettivo principale è di fornire un'approssimazione ammissibile dell'ottimo in tempi contenibili. Si caratterizzano per

- *Esplorazione intelligente dello spazio di ricerca*. Tentando di evitare di terminare in un minimo locale, tipico degli algoritmi di ricerca locale, lo spazio delle soluzioni viene esplorato anche ammettendo alcune mosse peggiorative. Nasce così un trade-off tra:
  - *intensificazione*, ovvero la ricerca deve concentrarsi nelle aree dov'è maggiormente presumibile che si possa trovare un minimo locale;
  - *diversificazione*, ovvero la ricerca deve garantire di valutare, anche indirettamente, il più ampio numero di soluzioni.
- *Proprietà any time*. Il tempo di esecuzione incide sulla qualità del risultato, infatti all'aumentare di questo il risultato può diventare più accurato. Matematicamente si dimostra che tali algoritmi convergono asintoticamente all'ottimo.
- *Algoritmi stocastici*. L'utilizzo di funzioni probabilistiche comporta che differenti esecuzioni, a partire dalle stesse condizioni iniziali, ritornino soluzioni differenti. Tali funzioni vengono usate ad esempio nelle fasi iniziali di scelta di soluzioni candidate o durante l'aggiornamento dei parametri nelle varie iterazioni dei cicli.

Pseudocodice

```
1 s←soluzione iniziale da generare
2 best←s
3 do
4       nearSolution←s.generaSoluzioneIntorno()
5       if (condizione di aggiornamento)
6             s←nearSolution
7       aggiornamento temperatura secondo la cooling
          function
8 while(¬ condizione di terminazione)
9 return best
```

Le funzioni e i valori dei parametri dipendono fortemente dal problema e dalle caratteristiche delle istanze testate. Pertanto ci si può orientare verso

codice poco raffinato ma che permette di risparmiare tempo nella sua scrittura, senza comunque compromettere qualità e tempi delle esecuzioni degli algoritmi.

Generazione soluzione iniziale Ad ogni job viene assegnato casualmente una machine da una delle classi ad esso compatibili. Successivamente viene verificato che per ogni machine class non vi siano istanti in cui la richiesta superi la disponibilità massima. Nel caso venga superata si ricorre a rigenerare interamente la soluzione utilizzando sempre la stessa procedura stocastica.

La soluzione presentata comunque offre un buon compromesso tra velocità di esecuzione e bontà della soluzione generata visto il tipo di istanze trattate. Il metodo potrebbe portare a ricorsione infinita, ad esempio per ogni istanza infeasible il metodo non terminerà mai. Per questo l'implementazione di simulated annealing realizzata per questa tesi è stata applicata a sole istanze feasible.

Temperatura iniziale Si fissa il valore T_0 della temperatura iniziale in modo che la probabilità iniziale di accettare mosse peggiorative (p_0) sia uguale a 0.8.[12] Tale valore è quello usato nella maggior parte delle implementazioni dell'algoritmo. La formula di riferimento è:

$$T_0 = \frac{\epsilon_m}{\ln p_0} \tag{4.3}$$

dove ϵ_m rappresenta la media delle mosse peggioranti iniziali[16]. Questa viene calcolata a partire da alcune mosse peggioranti nell'intorno della soluzione iniziale. Per calcolarla nel codice si utilizza un metodo apposito. Questo richiama più volte il metodo per la generazione di soluzioni nell'intorno della soluzione fornita come parametro di input e memorizza in un vettore solamente le soluzioni che hanno un valore della funzione di fitness superiore a quella fornita.

Raggiungimento dell'equilibrio Consiste nell'insieme di iterazioni che l'algoritmo compie senza aggiornare la temperatura. Questo corrisponde, nell'analogia con la fisica, alla fase di progressivo avvicinamento alla distribuzione di Boltzmann per quel valore di temperatura.

Definizione 4.2.1 (Epoch) *Si definisce* epoch *un insieme di iterazioni alla stessa temperatura.*

Le epoch indicano quando aggiornare la temperatura. Un'epoch termina quando sono state accettate un numero minimo di soluzioni.

Definizione 4.2.2 (Costo di un epoch) *Si definisce* costo di un epoch *il valore della funzione di fitness dell'ultima soluzione generata che appartiene all'epoch.*

La temperatura viene aggiornata se sono state eseguite un numero massimo di iterazioni, oppure se il costo dell'ultima epoch risulta entro una certa distanza dal costo di un'epoch precedente, in quanto si presume di aver raggiunto l'equilibrio per quella temperatura.

Generazione soluzioni in un intorno Viene scelto casualmente un job. Questo verrà riassegnato ad una machine di una machine class anch'essa scelta casualmente e differente da quella attualmente assegnata. successivamente viene ricontrollato che la nuova assegnazione non superi i limiti di disponibilità. In tal caso viene ripetuta l'intera procedura di generazione di soluzione in un intorno.

E' stato verificato che modificare una sola assegnazione per ogni invocazione della procedura non limita il numero di iterazioni dell'intero algoritmo e non compromette la bontà della soluzione finale.

Cooling schedule Stabilisce il criterio di variazione della temperatura. Nella pratica si ricorre ad una semplice cooling function del tipo:

$$T_{k+1} = \alpha T_k \tag{4.4}$$

L'implementazione realizzata ha usato tale cooling function. α è stata posta a 0.95, valore generalmente usato nella maggior parte delle esecuzioni dell'algoritmo che la utilizzano.[12][16]

Iterazioni dell'algoritmo Ad ogni iterazione dell'algoritmo la soluzione attualmente considerata (s) viene confrontata con alcune tra le soluzioni ammissibili del suo vicinato. s viene aggiornata al valore di $nearSolution \neq s$ con probabilità:

$$\begin{cases} 1 & \text{se } f_{nearSolution} > f_s \\ e^{-\frac{|f_{nearSolution} - f_s|}{T}} & \text{se } f_{nearSolution} < f_s \end{cases} \tag{4.5}$$

Pertanto la probabilità di effettuare mosse peggiorative cala al diminuire della temperatura e all'aumentare del peggioramento indotto da $nearSolution$. Il tutto segue l'analogia con la fisica. Infatti al calare della temperatura si tende ad avvicinarsi all'ottimo locale della zona in cui la soluzione si trova, invece quando la temperatura è elevata è più probabile spostarsi verso altre zone.

La formula 4.5 è valida nel caso di problemi di massimo, come appunto l'interval scheduling. Per problemi di minimo è sufficiente invertire le condizioni.

Terminazione (o congelamento) La terminazione dell'algoritmo si ha quando il valore della temperatura scende al di sotto del valore di temperatura minima passato come parametro. Il valore addottato è di 0.5, valore estremamente più basso della temperatura iniziale generalmente calcolata. Il motivo è di consentire un numero sufficientemente elevato di iterazioni.

Capitolo 5

Risultati dei test su istanze feasible

Uno dei parametri fondamentali in tutti i test è stato la velocità di esecuzione.

Ci si è interessati anche a quanto la matrice dei coefficienti fosse sparsa. Per farlo si è ricorsi ad un indice di non sparsità calcolato come:

$$non\,sparsità = \frac{\sharp \text{ coefficienti non nulli}}{\sharp \text{ coefficienti}} \tag{5.1}$$

Tale misura è stata introdotta in questa tesi per verificare se la sparsità della matrice incidesse nell'esecuzione degli algortimi risolutivi.

Nel corso dei test si è sempre valutato l'apporto degli essential partition nelle diverse situazioni.

Note

- I dati presentati, se non diversamente specificato, sono relativi al valore medio calcolato come media algebrica dei valori dei risultati dei specifici test. Per ogni configurazione dei parametri analizzati è stato eseguito un numero di test non inferiore a 50.
- I dati relativi ai tempi di esecuzione sono espressi in secondi.
- Se non espressamente indicato nei grafici:
  - in blu i dati relativi ai test di modelli che non ricorrono all'essential partition.
  - in arancione i dati relativi ai test di modelli che ricorrono all'essential partition.

5.1 Variazione probabilità compatibilità job - machine class

Tempo di esecuzione

Generalmente la variazione della probabilità, e quindi del numero di machine compatibili per ogni job, non incide sul tempo di esecuzione. Questo permette di affermare che il numero di compatibilità non ha grande influenza sul tempo di esecuzione.

L'utilizzo degli essential partition consente di ridurre significativamente il tempo di esecuzione. Infatti è più breve di almeno il 20% con una punta di quasi il 37%. Nella maggior parte dei casi tale riduzione rimane compresa tra il 20 e il 30%. I tempi delle esecuzioni senza essential partition presentano maggiore variabilità rispetto al caso senza essential partition.

Probabilità	senza EP	con EP
0,1	0,0813	0,0607
0,2	0,0747	0,0480
0,3	0,0773	0,0547
0,4	0,0780	0,0620
0,5	0,0800	0,0620

Tabella 5.1: Tempo di esecuzione al variare della probabilità delle compatibilità job-machine.

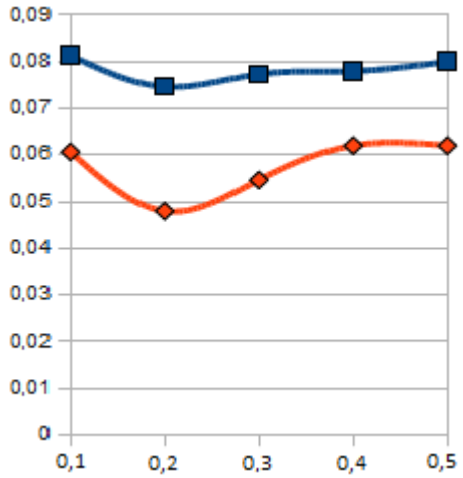

Figura 5.1: Tempo di esecuzione al variare della probabilità delle compatibilità job-machine.

Non sparsità

Aumentando la probabilità gli zeri nelle matrici dei coefficienti tendono a diminuire dato che le compatibilità aumentano.

Si è notato che la differenza tra la percentuale di non zero nella versione con essential partition e la percentuale di non zero nella versione senza essential partition tende ad aumentare con l'aumento della probabilità. In generale tale differenza non va al di sotto del 9%.

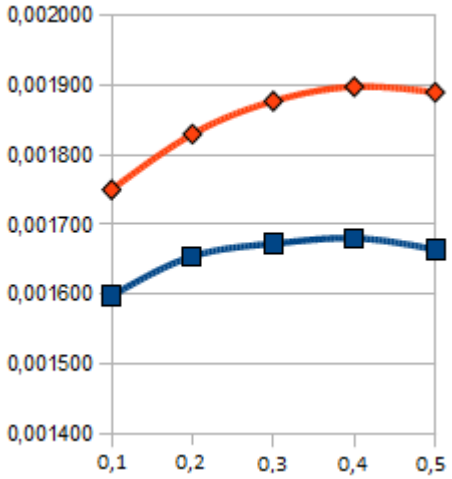

Figura 5.2: Non sparsità al variare della probabilità delle compatibilità job-machine.

5.2 Variazione numero machine class

Tempo di esecuzione

L'aumento di machine class comporta un innalzamento dei tempi di esecuzione. L'aumento generalmente si fa più significativo quando non si ricorre agli essential partition. Pertanto i benefici dati dal loro inserimento nel modello si hanno quando la taglia del problema si fa più elevata. Comunque anche nelle taglie più piccole il risparmio di tempo che riescono a dare è superiore al 25%.

	Machine per machine class			
	da 3 a 5		da 4 a 7	
Numero machine class	senza EP	con EP	senza EP	con EP
5	0,0653	0,0467	0,0780	0,0547
7	0,0933	0,0653	0,1133	0,0733
8	0,1240	0,0827	0,1207	0,0847
10	0,1620	0,1100	0,1633	0,1040

Tabella 5.2: Tempo di esecuzione al variare del numero di machine class.

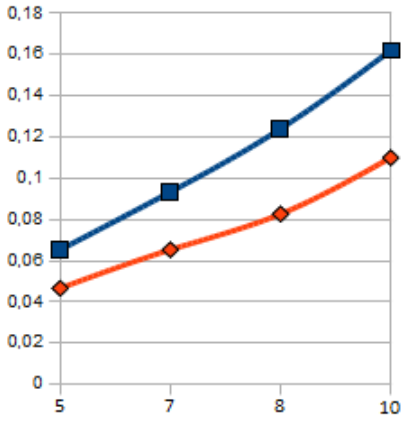

Figura 5.3: Tempo di esecuzione al variare del numero di machine class. Situazione con numero di machine per machine class che varia tra 3 e 5.

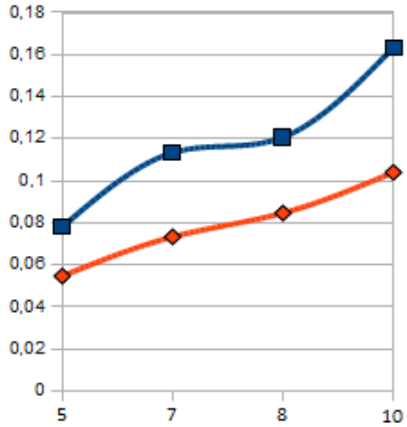

Figura 5.4: Tempo di esecuzione al variare del numero di machine class. Situazione con numero di machine per machine class che varia tra 4 e 7.

Non sparsità

Nonostante aumenti, il numero di machine class è nettamente inferiore di quello dei job. Questo comporta che la percentuale di non zero diminuisce quando il numero di machine class aumenta.

Quel che è più importante è che l'introduzione degli essential partition continua a rendere meno sparsa la matrice. La differenza rispetto al caso senza essential partition non va al di sotto del 4%.

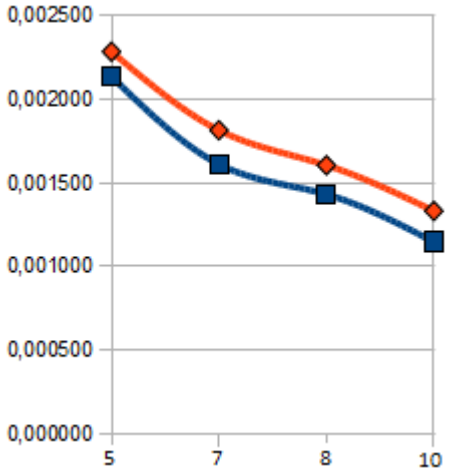

Figura 5.5: Non sparsità al variare del numero di machine class. Situazione con numero di machine per machine class che varia tra 3 e 5.

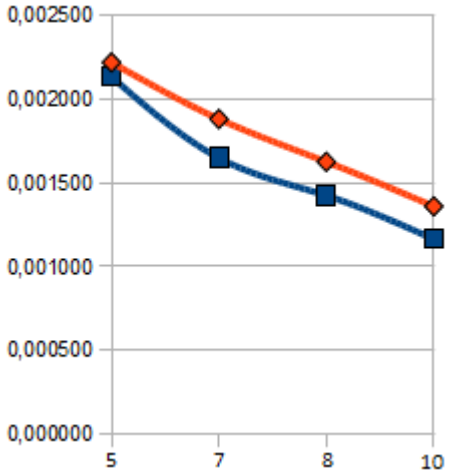

Figura 5.6: Non sparsità al variare del numero di machine class. Situazione con numero di machine per machine class che varia tra 4 e 7.

5.3 Variazione concentrazione dei job

Per ottenere l'effetto di variazione della concentrazione dei job si è ricorsi a esecuzioni della procedura di generazione di istanze feasible con differenti valori di massimo istante di inizo di un job all'interno di un blocco di job. Il numero di blocchi necessario è stato fissato a 1 mentre il numero di job per blocco oscillava tra i 550 e i 600 job. L'effetto di variazione della concentrazione è quindi dato dal modificare l'ampiezza temporale dei blocchi. Per ogni blocco viene definito un intervallo di tempo nel quale i job che vi appartengono possono essere eseguiti. Variandone l'ampezza si riesce a modellare la variazione di concentrazione.

Tempo di esecuzione

Indipendentemente dal numero di machine class, il diradare i job comporta una diminuzione nel tempo di esecuzione. Questo è giustificato dal fatto che i conflitti, e quindi i vincoli relativi, sono ristretti ad un numero minore di job.

Ampiezza temporale blocco	senza EP	con EP
250	0,0420	0,0260
300	0,0447	0,0247
350	0,0360	0,0253
400	0,0340	0,0213
450	0,0340	0,0233
500	0,0353	0,0247

Tabella 5.3: Tempo di esecuzione al variare della concentrazione dei job.

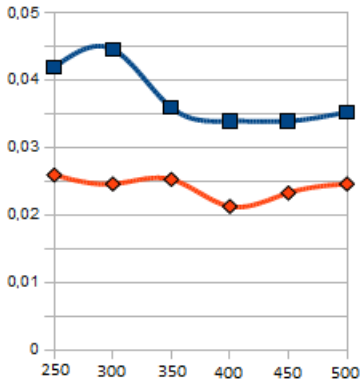

Figura 5.7: Tempo di esecuzione al variare della concentrazione dei job.

Si è notato che gli essential partition tendenzialmente sono più convenienti quando i job sono più concentrati. Infatti la differenza tra i tempi tende ad avere andamento decrescente con il diradarsi dei job. Il motivo è che l'utilizzo degli essential partition comporta una maggiore indipendenza del tempo di esecuzione dalle caratteristiche dell'istanza. Se si utilizza invece il modello senza essential partition le differenze si fanno più accentuate.

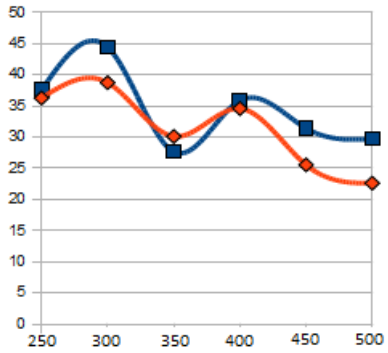

Figura 5.8: Risparmio sul tempo di esecuzione dato dall'utilizzo degli essential partition al variare della concentrazione dei job. In blu il caso di scarsità di machine e machine class, in arancione il caso con un numero maggiore di machine e machine class.

Non sparsità

Diradando i job la matrice dei coefficienti diviene più sparsa dato che aumenta il numero di vincoli ma il numero di job per ciascun vincolo è minore. Gli essential partition consentono di diminuire in modo sostanziale la sparsità della matrice, a patto che il numero di machine class disponibili non sia troppo basso.

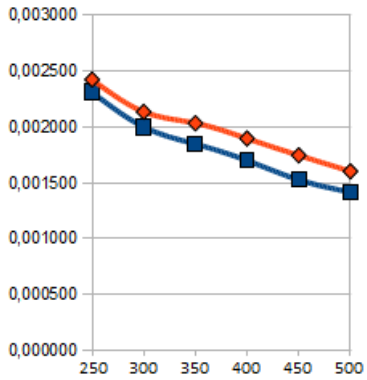

Figura 5.9: Non sparsità al variare della concentrazione dei job.

5.4 Variazione numero di job

Tempo di esecuzione

Aumentando il numero di job gli essential partition danno un apporto crescente alla riduzione di tempi di esecuzione. Infatti si passa da un risparmio dell'ordine del 15 - 20% con 450 - 500 job ai 30 - 35% con circa 1250 job. Si è registrato un andamento a gradino nei risparmi di tempo. Infatti passando dai 650 - 700 job ai 800 - 850 il guadagno quasi raddoppia.

Numero di job	senza EP	con EP
464	0,0600	0,0489
664	0,0878	0,0733
842	0,1256	0,0811
960	0,1467	0,0967
1260	0,1967	0,1300

Tabella 5.4: Tempo di esecuzione al variare del numero di job.

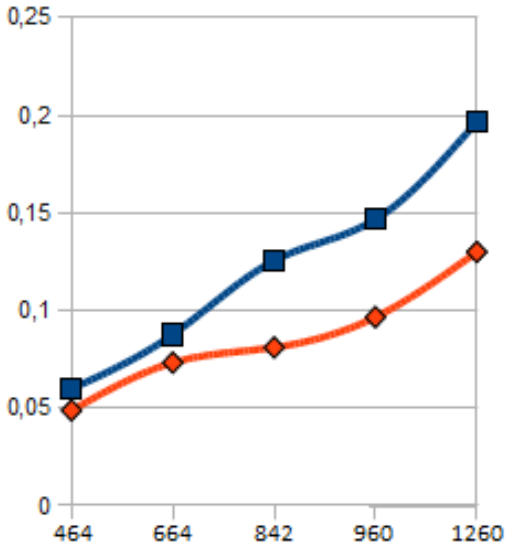

Figura 5.10: Tempo di esecuzione al variare del numero di job.

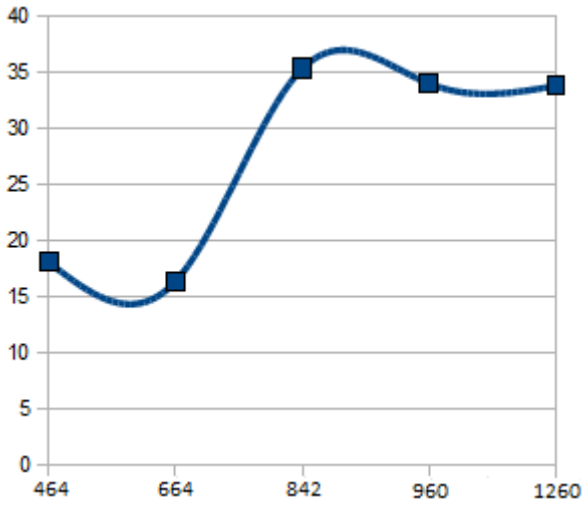

Figura 5.11: Risparmio sul tempo di esecuzione dato dall'utilizzo degli essential partition al variare del numero di job.

Capitolo 6

Risultati dei test di rilassamenti e algoritmi eurisitici su istanze feasible

Si sono analizzate delle tecniche di approssimazione della soluzione del problema. Le istanze di riferimento erano tutte feasible. Questo presupponeva che una soluzione ottima di riferimento esistesse. Lo studio si è concentrato nel capire di quanto fosse l'errore commesso dai metodi approssimati e dei relativi tempi di esecuzione.

Nota I dati presentati, se non diversamente specificato, sono relativi al valore medio calcolato come media algebrica dei valori dei risultati dei specifici test. Per ogni configurazione dei parametri analizzati è stato eseguito un numero di test non inferiore a 50.

6.1 Approssimazioni testate

Il problema oggetto di tutti i test è di massimo. Pertanto dati:

- z^* soluzione ottima
- z^H soluzione data dall'esecuzione di un algoritmo euristico
- z^R soluzione data dall'esecuzione di un rilassamento

è sempre vero che $z^H \leq z^* \leq z^R$.

Sono stati utilizzati due tipi di rilassamenti:

1. rilassamento vincolo 3.2;

2. rilassamento vincolo non riassegnabilità job in differenti intervalli essenziali.

Come algoritmo euristico di riferimento è stato utilizzato il simulated anealing.

6.1.1 Rilassamento vincolo 3.2

Il vincolo diviene

$$\sum_{j \in J_i} x_{ij} \leq 1 \, \forall i \in I \tag{6.1}$$

Così facendo si consente ai job di non essere schedulati. Essendo le variabili x_{ij} di tipo binario, per ogni job saranno quindi possibili due opzioni:

- essere schedulato su una sola machine;
- non essere schedulato.

Rispetto alla situazione non soggetta a rilassamento i cambiamenti saranno di due tipi:

- alcuni job non verranno schedulati
- altri verranno schedulati su machine differenti.

6.1.2 Rilassamento vincolo non riassegnabilità job in differenti intervalli essenziali

La modifica al modello presentata nella sezione 3.4 prevede comunque che un job non possa essere riassegnato con l'inizio di un nuovo intervallo essenziale. Rilassando questo vincolo l'assegnazione può invece variare nel tempo. Quindi è possible che nell'intervallo essenziale r il job i sia eseguito dalla machine di classe c, mentre nell'intervallo essenziale $r+1$ viene essere eseguito da una machine di classe $c^{'} \neq c$.

I motivi per cui il job i sia rischedulato possono essere:

1. è terminato un job j che occupava una machine di una classe che garantiva un guadagno maggiore;
2. è iniziato un job j che garantisce un maggior guadagno se occupa la machine attualmente è assegnata a i.

Entrambe le rischedulazione accadono quando una classe è nella situazione di massima occupazione.

Nella situazione corrispondente al motivo 2 si può avere un protrarsi degli effetti su altri job. Infatti se il job i viene spostato di machine questo potrebbe occupare la machine (di una classe differente dalla precedente) che è assegnata al job k. Tale ragionamento vale anche per quest'ultimo job avendo così un effetto a catena.

6.1.3 Simulated annealing

La configurazione utilizzata per i test è riportata di seguito.

- Cooling function di tipo *a rapporto di variazione di temperatura costante* e pari a 0.95.
- Condizione di terminazione basata sul *raggiungimento della temperatura minima* posta a 0.5. Tale valore è molto più basso di quello della temperatura iniziale che nei test si vedeva solitamente calcolata al fine di eseguire un alto numero di iterazioni.
- Numero massimo di iterazioni alla stessa temperatura è stato posto a 2000.
- Probabilità di accettare una mossa peggiorativa è stata posta pari a 0.8.
- Numero minimo di transizioni accettate in un'epoch pari a 100.
- Distanza minima tra due soluzioni di un'epoch è stata posta a 1. Pertanto l'epoch viene interrotta solo se si trova una soluzione che dia income che differisce al più di uno. Tale valore è relativamente basso per non compromettere la qualità della soluzione ritornata. Infatti ricorrendo a valori maggiori il numero di iterazioni sarebbe stato eccessivamente limitato.

6.2 Istanze di riferimento

Le istanze generate erano di taglia grande. Avevano tra i 1200 e i 1300 job che venivano generati a partire da 3 blocchi. Le machine class erano 6 e il numero di machine per classe oscillava tra 4 e 7. La probabilità di una compatibilità job - machine class era pari a 0.3.

La procedura di generazione aveva come obiettivo quello di ricondursi ad un caso con un numero di scelte per ogni job sufficientemente alto.

6.3 Risultati

6.3.1 Qualità delle soluzioni

I rilassamenti hanno generalmente dato risultati molto vicini a quelli del problema originario. L'errore percentuale non è mai andato oltre il 0.1%. Il rilassamento 1 ha dato risultati migliori dell'altro. Infatti non è mai andato oltre il 0.01%.

Il simulated annealing ha mostrato invece margini d'errore ben più ampi. Le soluzioni ritornate erano in un range di errore che andava dal 15 al 20%. Rispetto ai rilassamenti si nota anche una maggiore variabilità nella qualità che dipende da istanza a istanza.

Rispetto a quanto ottenuto in [8] i rilassamenti forniscono risultati più vicini all'ottimo. La situazione si ribalta nel confronto tra metodi euristici.

Metodo di approssimazione			Distanza % dall'ottimo
Rilassamenti	Vincolo 3.2	senza EP	0.002%
		con EP	0.002%
	Vincolo non riassegnabilità job in differenti intervalli essenziali		0.05%
Algoritmi euristici	Simulated annealing	Best cases	16.60%
		Worst cases	17.47%

Tabella 6.1: Qualità delle soluzioni per rilassamenti e algoritmi euristici.

6.3.2 Tempo di esecuzione

Sul fronte dei tempi di esecuzione il simulated annealing veniva risolto generalmente tra i 14 e i 15 secondi, anche se per qualche istanza si terminava entro i 14 secondi. Tendenzialmente le istanze con tempi di esecuzione bassi presentavano un errore maggiore nel valore della soluzione ritornata. Questo dimostra che il tempo di esecuzione è fondamentale per tale tipologia di algoritmi.

I tempi registrati per il simulated annealing sono leggermente minori a quelli del rilassamento 2. La costruzione delle istanze impiegava al più due secondi in più dell'esecuzione del simulated annealing. La risoluzione veniva invece completata in tempi decisamente più brevi, cioè nell'ordine dei 5 centesimi di secondo.

Il rilassamento 1 è quello che ha meglio figurato anche per i tempi di esecuzione. Infatti il tempo per la costruzione delle istanze era identico dato che non vengono introdotte nuove variabili, prezzo da pagare con l'altro rilassamento.

Metodo di approssimazione			Tempo di esecuzione rispetto all'ottimo
Rilassamenti	Vincolo 3.2	senza EP	Pochi decimi di secondo. Valore comparabile con l'ottimo.
		con EP	Pochi decimi di secondo. Valore comparabile con l'ottimo.
	Vincolo non riassegnabilità job in differenti intervalli essenziali		16-17 s. Molto più alto l'ottimo.
Algoritmi euristici	Simulated annealing		14-15 s. Molto più alto l'ottimo.

Tabella 6.2: Tempo di esecuzione per rilassamenti e algoritmi euristici.

Capitolo 7

Risultati dei test su istanze infeasible

I test compiuti per queste istanze si sono concentrati unicamente sui tempi di esecuzione. L'obiettivo è quello di capire di quanto tempo si necessita per determinare la infeasibility e per trovare una soluzione alternativa.

Note

- I dati presentati, se non diversamente specificato, sono relativi al valore medio calcolato come media algebrica dei valori dei risultati dei specifici test. Per ogni configurazione dei parametri analizzati è stato eseguito un numero di test non inferiore a 50.
- I dati relativi ai tempi di esecuzione sono espressi in secondi.

7.1 Caratteristiche dei test

Per tale categoria di istanze si è voluto verificare il tempo necessario al riconoscimento della infeasibility e quello necessario per eseguire un rilassamento. Il vincolo da rilassare è il 3.2. Nella versione rilassata si è consentito di ricercare una soluzione nella quale non fosse necessario che tutti i job fossero schedulati. Pertanto il vincolo 3.2 è stato modificato in

$$\sum_{j \in J_i} x_{ij} \leq 1 \, \forall i \in I \tag{7.1}$$

La taglia delle istanze prevede un numero di job compreso tra 1250 e 1300 generati a partire da 3 blocchi contigui di job. Un blocco di job è costituito da job che iniziano entro un determinato istante. Le machine class

disponibili sono 6 ciascuna con un numero di machine che varia da 4 a 7. La probabilità di una compatibilità tra un job e una machine class è del 30%. L'impostazione dei parametri garantisce che per ogni job ci siano un numero relativamente ampio di scelte.

Il metodo di generazione prevedeva di inserire tra i blocchi un gruppo di job le cui compatibilità portavano ad infeasibility. Avendo tre blocchi le possibili posizioni di inserimento sono quattro. Nel corso dei test ci si è riferiti a quattro distinti gruppi di job. I primi tre interessano un numero limitato di job, l'ultimo invece riguarda un numero ben più ampio di job. Si noti come l'infeasibility sia dovuta alla compatibilità dei job del gruppo, in quantità ben minore rispetto agli altri job.

Gruppo 1

E' composto da sette job tutti di durata 10. Gli istanti di inizio tra due job consecutivi distano una unità. Le machine class compatibili con tali job sono tre (0, 1 e 2) e hanno 2 machine ciascuna. Di seguito si riporta per ogni job le machine class compatibili.

1. 0, 1 e 2.
2. 1 e 2.
3. 0 e 2.
4. 1.
5. 0 e 1.
6. 1 e 2.
7. 0.

Gruppo 2

E' composto da tre job tutti di durata 5. Gli istanti di inizio tra due job consecutivi distano una unità. La machine class compatibile con tali job è solo una (0) e ha 2 machine.

Gruppo 3

E' composto da sette job tutti di durata 10. Gli istanti di inizio tra due job consecutivi distano una unità. Le machine class compatibili con tali job sono

tre (0, 1 e 2) e hanno 2 machine ciascuna. Di seguito si riporta per ogni job le machine class compatibili.

1. 0 e 1.
2. 1 e 2.
3. 0 e 2.
4. 0 e 1.
5. 1 e 2.
6. 0 e 2.
7. 0, 1 e 2.

Gruppo 4

E' composto da 44 job. Le machine class compatibili con tali job sono due (0 e 1) e hanno 2 machine ciascuna. Di seguito si riporta il pseudocodice per la generazione del gruppo.

```
1  job 1←new job(start, durata=3, machineClassCompatibili
       ={1})
2  ++start
3  for (i←0;i<10;++i)
4        job i*4+2←new job(start, durata=3,
             machineClassCompatibili={1})
5        ++start
6        job i*4+3←new job(start, durata=7,
             machineClassCompatibili={0, 1})
7        ++start
8        job i*4+4←new job(start, durata=3,
             machineClassCompatibili={0})
9        ++start
10       job i*4+5←new job(start, durata=7,
             machineClassCompatibili={0, 1})
11       start+=(i<9?5:4)
12 job 42←new job(start, durata=5,
       machineClassCompatibili={0})
13 ++start
14 job 43←new job(start, durata=4,
       machineClassCompatibili={1})
```

```
15 ++start
16 job 44←new job(start, durata=3,
       machineClassCompatibili={1})
```

Il gruppo presentato appartiene alla categoria 2 di insiemi di job per istanze feasible descritta a pagina 29. Infatti il pattern ripetuto 10 volte all'interno del ciclo *for* potrebbe essere ripetuto un numero arbitrario e non nullo di volte causando comunque infeasibility.

7.2 Risultati

I tempi di esecuzione mediamente calano del 30 - 40% rispetto a instanze feasible compatibili per numero di job, numero di machine class, numero di machine per machine class e compatibilità job - machine class. La dimunizione del tempo si ha sia per il riconoscimento dell'infeasibility che per la ricerca della soluzione ottima del rilassamento. Tra le due situazioni non si può stabilire a priori quella che verrà eseguita con minor tempo, infatti dai test si nota che ogni istanza è una situazione a sè stante. L'entità della diminuzione è uguale sia se ci si riferisce a modelli che non fanno uso di essential partition che a modelli che ne fanno uso.

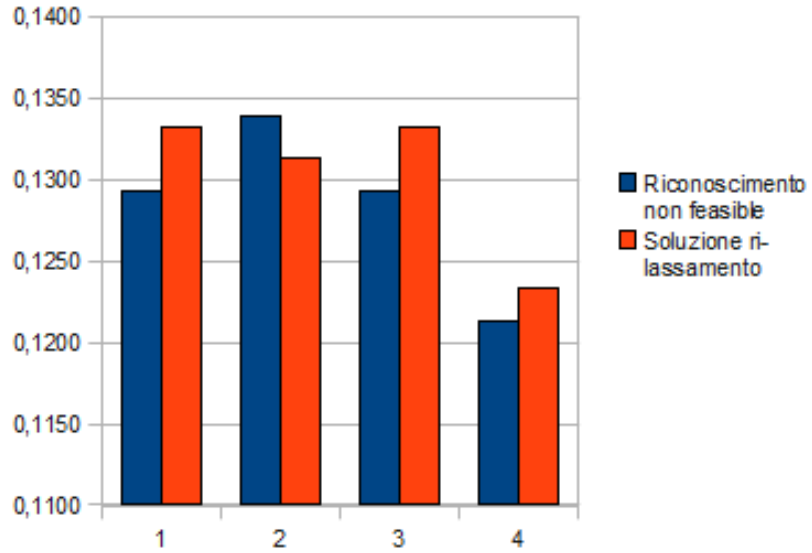

Figura 7.1: Tempo di esecuzione per istanze non feasible con modelli che non utilizzano essential partition.

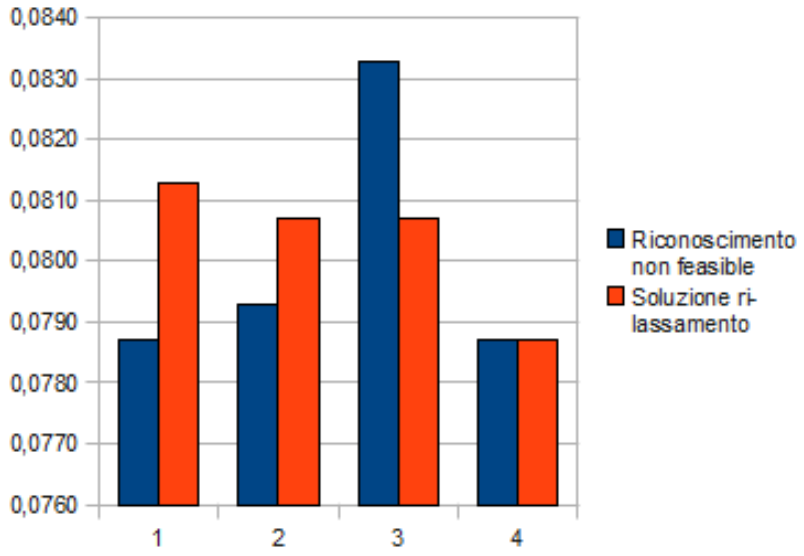

Figura 7.2: Tempo di esecuzione per istanze non feasible con modelli che utilizzano essential partition.

Si conferma invece l'utilità degli essential partition. Infatti consentono di risparmiare tra il 35 e il 40 % del tempo sui corrispondenti modelli che non ne fanno uso.

		senza EP	con EP	% diminuzione tempo di esecuzione data da EP
Istanza feasible		0,1967	0,1300	33,91
Riconoscimento non feasible	1	0,1293	0,0787	39,13
	2	0,1340	0,0793	40,82
	3	0,1293	0,0833	35,58
	4	0,1213	0,0787	35,12
Soluzione rilassamento	1	0,1333	0,0813	39,01
	2	0,1313	0,0807	38,54
	3	0,1333	0,0807	39,46
	4	0,1233	0,0787	36,17

Tabella 7.1: Diminuzione del tempo di esecuzione dato dall'utilizzo degli essential partition per istanze infeasible.

Capitolo 8

Conclusioni

8.1 Commento dei risultati

Nei test si è verificata l'esistenza di relazioni tra tempo di esecuzione e caratteristiche delle matrici dei coefficienti. I risultati ottenuti escludono particolari condizionamenti al tempo di esecuzione dati dal numero di zeri nella tabella. Ad influenzare in modo decisamente maggiore è invece la dimensione della matrice. Infatti l'utilizzo degli essential partition consente di diminuire il numero di vincoli e quindi il tempo di esecuzione. A supporto della relazione vi è il caso analizzato per *rilassamento vincolo non riassegnabilità job in differenti essential partition*. In questo caso la dimensione della matrice cresceva notevolmente, così come il tempo di esecuzione.

Risultati soddisfacenti sono stati dati da *rilassamento vincolo 3.2*. Infatti la qualità delle soluzioni è stata notevole. Nella maggior parte dei casi coincideva con la soluzione ottimale del problema originario. Quando invece vi erano delle differenze queste erano sostanzialmente contenute. Il fatto che la qualità sia buona è anche dovuto alla tipologia di istanze. Infatti vi è un numero elevato di machine per ogni machine class che generalmente garantisce molte scelte per ogni job. I tempi di esecuzione sono rimasti pressochè invariati rispetto al problema originario.

In generale conviene utilizzare i metodi euristici nel caso in cui si debba modificare un piano di schedulazioni già esistente. Specialmente nel caso in cui le modifiche al piano siano limitate, con un numero di iterazioni relativamente basso si può arrivare al nuovo ottimo.

Il *rilassamento vincolo 3.2* si è dimostrato efficace anche nelle situazioni di istanze infeasible. Infatti i tempi di esecuzione sono minori rispetto alla ricerca della soluzione ottima per istanze di dimensione comparabile. Inoltre la percentuale di job schedulati rimane comunque alta, escludendo solo quei

job a basso guadagno che comportavano la infeasibility dell'instanza.

8.2 Argomenti di studio e miglioramento futuri

Visti gli effetti sul tempo di esecuzione dati dalla dimensione della matrice del modello è conveniente lavorare su questa. Infatti incide sui tempi molto di più di quanto possano i singoli valori. Nel corso delle attività della tesi è stato ipotizzato di migliorare i vincoli 3.3 ponendo al posto di p_j il valore del numero massimo di job che potrebbero richiedere una machine di classe j nell'intervallo di interessse. I risultati sperimentali hanno fatto notare che il tempo di esecuzione non subiva variazioni visibili. Pertanto studi più approfonditi non sono stati compiuti.

Nei test si è affrontata una tipologia di istanze in cui ogni job è stato trattato in modo separato dagli altri. Infatti non sono state previste classi di job con determinate caratteristiche. Caso interessante è quando i job di una particolare classe possano utilizzare solamente determinate classi di machine. Questo limita il numero di scelte per ogni job più pesantemente di quanto visto in questa tesi.

Le compatibilità interessavano coppie aereo - tipologia di piazzola. Estensioni future possono interessarsi a compatibilità dove siano coinvolte più piazzole, anche di tipologia differente. Dovrà però essere meglio definito il concetto di guadagno in tali situazioni.

Nei test come algoritmo euristico di approssimazione si è ricorsi al simulated annealing. Vi sono comunque alternative generalmente più performanti in quanto a qualità delle soluzioni fornite. Studi successivi possono utilizzare altri algoritmi, specialmente quelli genetici. Inoltre può essere interessante analizzare tipologie di istanze in cui per ogni job ci sia meno libertà di scelta.

Bibliografia

[1] G.B. Dantzig and D.R. Fulkerson, *Minimizing the number of tankers to meet a fixed schedule*, Nav Res Logist Q 1, 1954.

[2] L.R. Ford Jr. and D.R. Fulkerson, *Flows in networks*, Princeton University Press, Princeton, New Jersey, 1962.

[3] A.W.J. Kolen, J.K. Lenstra et al., *Interval Scheduling: A Survey*, Wiley interScience, 2007.

[4] E.M. Arkin and E.B. Silverberg, *Scheduling with fixed start and end times*, Discrete Appl Math 18, 1987.

[5] R.J. Lipton and A. Tomkins, *Online interval scheduling*, Proceedings of the fifth annual ACM-SIAM Symposium on Discrete Algorithms, Arlington, Virginia, 1994.

[6] M. Fischetti, S. Martello, and P. Toth, *The fixed job schedule problem with working-time constraints*, Oper Res 37, 1989.

[7] A.W.J. Kolen and L.G. Kroon, *On the computational complexity of (maximum) class scheduling*, European Journal of Operational Research, 1991.

[8] L.G. Kroon, M. Salomon and L.N. Van Wassenhove, *Exact and approsimation algorithms for the tactical fixed interval scheduling problem*, Oper Res 45, 1997.

[9] A.W.J. Kolen and L.G. Kroon, *License Class Design: Complexity and Algorithms*, Eur. J. Opnl. Res. 63, 1992.

[10] L.G. Kroon, M. Salomon and L.N. Van Wassenhove, *Exact and approsimation algorithms for the operational fixed interval scheduling problem*, European Journal of Operational Research 82, 1995.

[11] P.J.M. van Laarhoven, and E.H.L. Aaris, *Simulated annealing: theory and applications*, D.Reidel, Dordrecht, 1987.

[12] S. Kirkpatrick, C.D. Gelatt Jr. and M.P. Vecchi, *Optimization by Simulated Annealing*, IBM Research Report RC 9355, 1982.

[13] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller and e. Teller, *Equation of State Calculations by Fast Computing Machines*, Journal of Chemical Physics, 21(1953) 1087-1092.

[14] C. Blum and A. Roli, *Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison*, ACM, New York, 2003.

[15] W. Feller, *An Introduction to Probability Theory and Applications*, vol. 1, Whiley, New York, 1950.

[16] D.S. Johnson, C.R. Aragon, L.A. McGeoch and C. Schevon, *Optimization by Simulated Annealing: an Experimental Evaluation*, List of Abstracts, Workshop on Statistical Physics in Engineering and Biology, Yorktown Heights, April 1984, revised version, 1986.

Ringraziamenti

Ringrazio i Proff. Giorgio Romanin Jacur e Matteo Fischetti per la disponibilità, il materiale fornito e il supporto.

Intendo ringraziare Nicola Zago per i suggerimenti e le correzioni alla relazione.

Grazie a tutti i parenti e gli amici che in questo periodo mi hanno aiutato e incoraggiato.

In particolar modo ringrazio i miei genitori Franco e Anna Maria, mia sorella Marta e i miei nonni Maria, Pietro e Maria.

Zoggia Luca