UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Tecnica e Gestione Industriale

Corso di laurea triennale in Ingegneria Gestionale

OTTIMIZZAZIONE DEI TURNI DI LAVORO CON MANSIONI MULTIPLE

Relatore:

Ch.mo Prof. Giorgio Romanin Jacur

Laureanda:

Ilaria Maci

1008541

Anno Accademico 2014-2015

INDICE

SOMMARIO

Il tema centrale della tesi è un algoritmo trifase, efficiente ed efficace, redatto per la soluzione del problema riguardante la pianificazione delle attività, per ottenere una minimizzazione dei turni del personale. Tale argomento è stato discusso nell'articolo pubblicato online il 24 gennaio 2014 dalla rivista European Journal of Operational Research, *Minimizing shifts for personnel task scheduling problems: A three-phase Algorithm.*

I risultati sperimentali mostrano che l'algoritmo proposto supera gli algoritmi esistenti in termini di soluzioni ottimali, in quanto migliora la maggior parte delle soluzioni più conosciute e rileva soluzioni fattibili di alta qualità per alcuni casi irrisolti della letteratura.

Per illustrare l'efficacia, l'algoritmo presentato viene applicato ad un problema di verifica mediante lo sviluppo di un programma.

INTRODUZIONE

Il problema della pianificazione del personale o *personnel scheduling problem* (di seguito indicato con l'acronimo PSP) è stato oggetto di interesse per ricercatori e professionisti già dagli studi pionieristici di Edie e di Dantzig nel 1954

Il PSP consiste in svariate decisioni che devono essere prese su come assegnare una sequenza di mansioni e/o di turni.

Per quanto riguarda una pianificazione orizzontale, gli attuali PSP possono essere classificati in tre grandi gruppi: *shift scheduling* (pianificazione dei turni), *days off scheduling* (pianificazione dei giorni di riposo) e *tour scheduling* (pianificazione delle rotazioni). La prima tipologia di problema ha come obiettivo la soddisfazione delle richieste di personale in ogni turno. Nella seconda tipologia, con la pianificazione dei giorni di riposo si ottiene una lista rispettando il vincolo che la lunghezza della settimana lavorativa non corrisponda alla lunghezza della settimana operativa di un dipendente. Mentre, nell'ultima categoria di problemi vengono combinati i primi due tipi, sia per quanto riguarda la suddivisione delle mansioni nelle varie ore della giornata, sia per la programmazione dei giorni lavorativi durante la settimana.

Questi tre tipi di PSP sono comuni in molti campi di applicazione, come i sistemi di trasporti, i sistemi manifatturieri, i call center, gli alberghi, gli ospedali e i sistemi di servizi sanitari. I ricercatori hanno sempre mostrato un continuo interesse nei confronti di tale questione proprio perché la richiesta di soluzioni a tali problemi deriva da vari ambiti.

In questo elaborato viene presentato e testato mediante una versione semplificata un algoritmo trifase definito in letteratura negli ultimi anni, che combina i vantaggi dell'algoritmo euristico costruttivo, del meta-euristico e del metodo esatto per poter risolvere problemi di tipo *shift minimization personnel task scheduling* (di seguito indicato con l'acronimo SMPTSP).

Il resto di questa trattazione è così organizzato:

- Capitolo 1: viene presentato il problema e le varie soluzioni proposte in letteratura nel corso degli anni, con una breve presentazione di quelli che sono i vantaggi e gli svantaggi di ogni singolo algoritmo a cui viene fatto cenno;
- Capitolo 2: viene definito il problema SMPTSP e il rispettivo modello risolutivo;

- Capitolo 3: viene descritto nel dettaglio l’algoritmo trifase;
- Capitolo 4: viene presentato un modello semplificativo con l’ausilio di un programma da me codificato, compilato ed eseguito con la finalità di dimostrare la fattibilità e le caratteristiche delle varie soluzioni ottenute eseguendo sequenzialmente le singole fasi dell’algoritmo;
- Capitolo 5: viene proposto il confronto tra l’algoritmo trifase e quello esistente, il VAWA, tratto dalla letteratura.

CAPITOLO 1

PSP e soluzioni presenti in letteratura

I problemi di scheduling consistono nella schedulazione del personale o dei turni del personale, cioè nel processo mirato alla definizione dei turni di lavoro degli operatori, in modo da soddisfare le richieste di intervento previste in un determinato periodo temporale. Tale processo può essere suddiviso in diverse parti. La prima parte mira a definire il numero di persone, con particolari competenze, necessarie a soddisfare una determinata richiesta di intervento. Nella seconda parte singole risorse o membri della squadra devono essere assegnati a turni in modo da soddisfare il livello di competenza richiesto alla squadra per eseguire un intervento e, infine, le singole mansioni sono assegnate agli individui per ogni turno.

È estremamente difficile trovare delle buone soluzioni a questi problemi altamente vincolati e complessi, ed è anche difficile determinare le soluzioni ottime che minimizzano i costi, strettamente dipendenti dal numero di risorse impiegate.

Esiste una vasta letteratura fatta di soluzioni a tali problemi, vari metodi risolutivi. Sostanzialmente si può fare una suddivisione in due grandi gruppi: metodi esatti e metodi euristici.

1.1.Metodi esatti

Tra i metodi esatti vi sono:

- la *programmazione lineare* (Fowler, Wirojanagud et al., 2008, pp.724-740; Hochbaum & Levin, 2006, pp.327-340; Hojati & Patil, 2011, pp.37-50): un problema di programmazione lineare è un problema di ottimizzazione in cui la funzione obiettivo e tutti i vincoli che devono essere rispettati dalla soluzione sono scritti come funzione lineare delle variabili;
- la *programmazione a vincoli* o *constraint programming* (Laporte & Pesant, 2004, pp.1208-1271; Qu & He, 2009, pp.221-224): tale programmazione è un paradigma dove le relazioni fra variabili possono essere dichiarate in forma di vincoli, i quali differiscono dalle primitive, normalmente definite dagli altri linguaggi di programmazione, per il fatto che non specificano azioni singole da eseguire step by

step, ma si limitano a specificare le proprietà di cui deve essere dotata la soluzione da trovare;

- il *metodo goal programming* (Azaiez & Al Sharif, 2005, pp.491-507; Lin, Chen et al., 2012, pp.483-493; Topaloglu & Ozkarahan, 2004, pp.135-158): è un metodo di analisi multi-obiettivo che si basa sulla minimizzazione della funzione di distanza da traguardi predefiniti;
- la *programmazione intera (Eiselt & Marianov, 2008*, pp.513-524*; Eitzen, Pantoet al., 2004, pp.359-372; Seckiner, Gokcen et al., 2007, pp.694-699)*: un problema di programmazione lineare intera è un problema di programmazione lineare nel quale le variabili sono vincolate ad assumere valori interi;
- la *programmazione intera mista* (Firat & Hurkens, 2012, pp.363-380; Hertz, Lahrichi, et al., 2010, pp.860-873; Yilmaz, 2012, pp.491-496): un problema di programmazione lineare intera mista è un problema in cui solo alcune variabili sono vincolate ad essere intere;
- il *metodo della generazione di colonna o column generation (Al-Yakoob & Sherali, 2008, pp.34-43; He & Qu, 2012, pp.3331-3343; Restrepo, Lozano et al., 2012, pp.466-472):* tale metodo viene usato principalmente in presenza di un elevato numero di variabili e di conseguenza non tutte le relative colonne possono essere riportate all'interno del tableau (da ciò deriva il nome di column generation);
- la *programmazione dinamica* (Beliën & Demeulemeester, 2007, pp.143-166; Elshafei & Alfares, 2008, pp.85-93): tecnica originariamente introdotta per risolvere alcuni problemi decisionali a più fasi e problemi di controllo ottimo, in cui si deve prendere una decisione per ogni istante di tempo, con tale tecnica il problema complessivo si separa in una successione di problemi a una singola variabile;
- il *metodo dei moltiplicatori di Lagrange* (Bard & Purnomo, 2007, pp.5-23; Pot, Bhulai et al., 2008, pp.421-428): se il vincolo non è lineare rispetto alle variabili, si utilizza il metodo dei moltiplicatori di Lagrange, che trasforma un problema di massimo o minimo vincolato in un problema di massimo o minimo libero.

Tutti i metodi fin qui esposti, applicati a problemi di PSP di moderate dimensioni, hanno permesso di ottenere soluzioni ottimali anche in tempi rapidi, mentre per PSP di più elevate dimensioni non è stato possibile ottenere soluzioni fattibili in tempi moderati.

Da tali difficoltà, relative a problemi più ampi, sono stati sviluppati una serie di algoritmi euristici che hanno permesso il riscontro di soluzioni fattibili in tempi di calcolo ritenuti ragionevoli.

1.2.Metodi euristici

Come da definizione, le procedure euristiche sono da ritenersi strade obbligate per poter ricavare una soluzione approssimativamente molto vicina a quella ottima.

Gli algoritmi euristici presenti nella letteratura come soluzione di problemi di PSP possono essere suddivisi in due tipologie: *euristici costruttivi (constructive heuristics) ed euristici migliorativi (improvement heuristics).*

1.2.1.Algoritmo euristico costruttivo

Si tratta di un procedimento da seguire per "costruire" una soluzione ammissibile. Le fasi fondamentali di un algoritmo costruttivo sono:

- ✓ Inizializzazione: si sceglie un elemento di partenza per la costruzione della soluzione parziale S.
- ✓ Selezione di un nuovo elemento da aggiungere alla soluzione parziale: si individua il criterio di selezione di un nuovo elemento da aggiungere alla soluzione parziale S.
- ✓ Criterio di arresto: se S è completa e, quindi, ammissibile, la procedura si interrompe altrimenti si ritorna alla fase precedente.

Nel caso del problema in esame, questo metodo attribuisce un indice di priorità ad ogni lavoratore mediante una dettagliata regola, e specifica, ad ogni step, un turno o una mansione. In questo modo, una volta che ogni turno di un dipendente viene determinato, esso risulta essere fisso e non può essere cambiato.

Negli ultimi decenni algoritmi euristici costruttivi per la soluzione del PSP sono stati proposti per un gran numero di applicazioni come la pianificazione infermieristica (Brucker, Burke et al., 2010, pp.559-573; Wright, Bretthauer et al. 39-70, 2006), la programmazione dell'equipaggio (Elizondo, Parada et al., 2010, pp.575-591; Lin et al., 2012, pp.483-493) e la pianificazione dei turni di lavoro dei camionisti (Goel, Archetti et al., 2012, pp.1122-1132).

Il vantaggio di queste tipologie di procedure è rappresentato dalla velocità con cui vengono prodotte le liste di configurazione di turni e operatori, però spesso tali che la qualità dei turni ottenuti non è eccellente, soprattutto per problemi su larga scala.

1.2.2.Algoritmo euristico migliorativo

Il metodo euristico migliorativo, al contrario di quello costruttivo, parte da una lista implementata e da quest'ultima genera un elenco iterativamente migliorato.

In tale algoritmo trovano particolare applicazione le tecniche meta-euristiche, mediante le quali si parte da una soluzione euristica iniziale e si estende la ricerca di possibili miglioramenti in regioni diverse nello spazio delle soluzioni (fase di diversificazione) e poi, una volta individuata una regione si mette in evidenza la soluzione migliore in tale regione (fase di intensificazione), estendendo così la ricerca locale.

Le tre tecniche meta-euristiche maggiormente adottate per la soluzione dei PSP sono: simulated annealing (Akbari, Zandieh et al., 2013, pp.1017-1027; Cordeau, Laporte et al., 2010, pp.393-409; Thompson & Goodale, 2006, pp.376-390), tabu search (Di Gaspero et al., 2007, pp.79-105; Elizondo et al., 2010, pp.575-591; Lucic & Teodorovic, 2007, pp.311-338) e algoritmi genetici (Asensio-Cuesta, Diego-Mas et al., 2012, pp.1161-1174; Beddoe & Petrovic, 2006, pp.649-671; Moz & Pato, 2007, pp.667-691). Oltre a queste tre classi generali sono state aggiunte altre tecniche meta-euristiche popolari come scatter search (Burke, Curtois et al., 2010, pp.1667-1679; Laguna, Casado et al., 2005, pp.649-658; Maenhout & Vanhoucke, 2010, pp.155-197), ricerca locale iterata (Bellanti, Carello et al., 2004, pp.28-40; Burke, Curtois et al., 2011, pp.360-367), variable neighborhood search (Burke, Curtois et al., 2008, pp.330-341; Burke, Li et al., 2010, pp.484-493), particle swarm optimization (Akjiratikarl, Yenradee et al., 2007, pp.559-583; Gunther & Nissen, 2010, pp.451-461; Nissen & Gunther, 2009, pp.228-239), mimetic algorithm (Ozcan, 2005, pp.482-492), electromagnetic meta-heuristic (Maenhout & Vanhoucke, 2007, pp.359-385), neural network (Hao, Lai et al., 2004, pp.65-90), ant colony optimization (Gutjahr & Rauner, 2007, pp.642-666), greedy random adaptive search procedure (Goodman, Dowsland et al., 2009, pp.351-379), hill-climbing heuristic (Cipriano, Di Gaspero et al., 2006, pp.110-123) e hyperheuristic algorithm (Chakhlevitch & Cowling, 2005, pp.23-33).

Il vantaggio di queste tecniche sicuramente è rappresentato dalla possibilità di ottenere soluzioni fattibili anche in tempi di calcolo ragionevoli. Ma anche in questo caso, come per gli algoritmi euristici costruttivi, non è possibile dimostrare che si ottengono delle soluzioni ottimali e neanche dei restringimenti del range in cui vanno ricercate tali soluzioni.

CAPITOLO 2

SMPTSP e modello risolutivo proposto

Il problema della programmazione del personale è diventato un problema di minimizzazione proprio perché è atto a soddisfare l'esigenza pratica di avere giornalmente una lista dei turni, con l'obiettivo di minimizzare il numero dei turni e i costi che ad essi sono necessariamente connessi. Da tutto ciò deriva il nome di *shift minimization personnel task scheduling.*

2.1.Caratteristiche dello SMPTSP

Un problema, in cui devono essere minimizzati i turni, è tipico di un contesto tale che:

- l'unico costo sostenuto è relativo al numero di lavoratori impiegati e dunque di turni necessari per eseguire l'insieme di attività indicato;
- ogni singolo lavoratore possiede le qualifiche per poter svolgere un dato sottoinsieme di mansioni tra tutte quelle che sono richieste;
- ciascuna attività deve essere svolta da uno specifico lavoratore, in un determinato turno indicato, in un intervallo temporale specificato da un istante di inizio attività ed uno di fine;
- l'obiettivo principale è quello di minimizzare il numero di lavoratori impiegati e con esso anche il numero dei turni richiesti.

Un problema in cui compaiono queste caratteristiche si può considerare come una variante di un PSP utile per la minimizzazione del numero degli addetti richiesti, che può essere usato come strumento di gestione operativa giornaliera, in quanto relativamente ad ogni singolo giorno, indica il numero di operatori necessari per poter svolgere tutte le mansioni di quel particolare giorno.

2.1.1.Origini dello SMPTSP

Lo SMPTSP introdotto da Krishnamoorthy, Ernst et al. (2012, pp.34-48) deriva da un problema di turni in un grande aeroporto internazionale, che coinvolge circa 500 dipendenti su un orizzonte di pianificazione mensile. In questo studio, i ricercatori hanno messo in evidenza una formulazione matematica e hanno proposto un nuovo algoritmo euristico basato su *volume algortihm* (VA) e *Wedelin's algorithm* (WA), algoritmi che

insieme vengono indicati con l'acronimo VAWA. Entrambi gli algoritmi si basano sulla definizione di vincoli più difficili da rilassare. È per questo che il VAWA è definito più efficiente e può essere usato per risolvere problemi su vasta scala, come quelli tipici della realtà.

Negli anni passati, nella letteratura sono stati fortemente sviluppati algoritmi meta-euristici e metodi esatti, solo nell'ultimo periodo hanno preso maggior piede tecniche ibride e approcci multi-fase.

2.2.Definizione del problema

Il problema di minimizzazione dei turni può essere formalmente definito come segue.

Un insieme di task T=$\{t_1, \dots, t_n\}$ necessita di essere attribuito ad un insieme eterogeneo di lavoratori[1] W=$\{w_1, \dots, w_m\}$ mediante un orizzonte di pianificazione specificato. L'intervallo di tempo in cui ogni singolo task deve essere eseguito è definito da una dettagliata tabella dei tempi, in cui compaiono per ogni singolo task un tempo di inizio attività s_{t_i} e un tempo di fine attività f_{t_i}. Ciascun lavoratore, inoltre, possiede un certo numero di abilità o di qualifiche che gli permettono di eseguire un sottoinsieme limitato di compiti specifici.

L'obiettivo è quello, già più volte specificato, di ottenere un elenco ottimizzato dei turni, e quindi conseguentemente dei lavoratori, necessari per svolgere tutte le mansioni relative ad uno specifico arco temporale.

Nello SMPTSP in esame vengono inoltre presentate le seguenti ipotesi:

- Il numero di attività, così come i loro orari di inizio e fine attività, sono fissi e noti in anticipo.
- Ogni attività è considerata un'entità indivisibile, anche se si pensa che possa essere composta da singole unità, definite micro task.
- Non è consentito attribuire arbitrariamente delle preferenze tra le varie mansioni.
- Non esistono vincoli di precedenza tra i compiti.

[1] Il termine lavoratore, in tale contesto, può anche essere riferito ad altre risorse come macchinari o processori.

- Le attività non possono essere svolte da lavoratori che non possiedono le competenze specifiche richieste.
- Ogni attività viene eseguita solo una volta da un lavoratore qualificato.
- Ogni lavoratore può eseguire, al massimo, un compito alla volta.
- Ogni lavoratore opera indipendentemente dagli altri lavoratori.
- Una volta che un lavoratore inizia a svolgere un compito, egli deve essere disponibile fino al completamento del lavoro, e non devono essere presenti interruzioni né dovute a pause né ad altre cause.
- Una volta che il compito è stato iniziato da un lavoratore, non può essere trasferito ad un altro lavoratore, quindi l'intera operazione deve essere completata dall'operatore iniziale stesso.

2.3.Modello risolutivo

Il problema descritto sopra può essere formulato mediante il modello BIP, che è simile al modello presentato in Krishnamoorthy, Ernst et al. (2012, pp.34-48):

Minimizzare $z = \sum_{j=1}^{m} y_{w_j}$

Sotto i vincoli $\sum_{w_j \in W_{t_i}} x_{t_i w_j} = 1, \forall i$ (1)

$\sum_{t_i \in c_k^W} x_{t_i w_j} \leq y_{w_j}, \forall j,\ \mathrm{c_k^W} \in C^W$ (2)

$y_{w_j}{}^{2} \in \{0,1\}, \forall j$ (3)

$x_{t_i w_j} \in \{0,1\}, \forall i,j$ (4)

dove y_{w_j} (j = 1, 2, . . ., m) è una variabile decisionale binaria che indica se il lavoratore w_j è impegnato (y_{w_j} =1) oppure no (y_{w_j} =0); W_{t_i} rappresenta l'insieme dei lavoratori che possono eseguire compiti t_i; $x_{t_i w_j}$ è una variabile decisionale binaria che indica se il task

[2] Krishnamoorthy et al. (2012) hanno messo in evidenza che considerando y_{w_j} come una variabile continua ($0 \geq y_{w_j} \geq 1$) si ottengono risultati con tempi di calcolo nettamente inferiori.

t_i è assegnato al lavoratore w_j ($x_{t_i w_j}$ =1) oppure no ($x_{t_i w_j}$ =0); c_k^W specifica un insieme ristretto massimo di task che possono essere eseguiti in uno specifico intervallo temporale senza sovrapposizioni, appartenente all'insieme $C^W = \{ c_1^W, \dots, c_l^W \}$, con *k=1,…, l*; tale insieme ristretto può essere identificato mediante un algoritmo in tempo polinomiale[3], proposto da Krishnamoorthy e Ernst (2001, pp.343-368).

Nel modello BIP sopraindicato vengono riportati: il vincolo che garantisce l'unicità di attribuzione di ciascuna mansione ad un solo lavoratore (1) e il vincolo per prevenire la sovrapposizione tra due mansioni attribuite ad uno stesso lavoratore e per garantire l'esecuzione da parte di un lavoratore dei soli task che egli può eseguire in accordo con le sue qualifiche e le sue capacità (2). I vincoli (3) e (4) indicano rispettivamente che $x_{t_i w_j}$ e y_{w_j} sono variabili decisionali binarie.

[3]Un algoritmo è detto in tempo polinomiale se il suo tempo di esecuzione è limitato superiormente da un'espressione polinomiale nella dimensione dell'input per l'algoritmo, cioè, T(n) = O(n^k), per una qualche costante k.

CAPITOLO 3

Algoritmo trifase

L'approccio proposto si compone di tre fasi. La prima fase è costituita da un euristico costruttivo che permette di ottenere rapidamente una soluzione iniziale dello SMPTSP. La seconda fase è composta da un algoritmo greedy iterativo (di seguito indicato con l'acronimo IG), che permette di perfezionare la soluzione iniziale cercando di ottenere una pianificazione quasi ottimale. Infine, la pianificazione quasi ottimale viene posta come soluzione iniziale in modo tale da rappresentare un limite superiore per la funzione obiettivo nel modello BIP, se tale soluzione risulta essere fattibile. Da qui mediante Gurobi, un solutore di programmazione matematica state-of-the-art, si può ottenere una soluzione ottima.

3.1.Fase uno: utilizzo di un euristico costruttivo per ottenere una soluzione iniziale

Questa procedura euristica costruttiva, di seguito esposta, permette di ottenere rapidamente una soluzione iniziale ξ_0 dello SMPTSP, non necessariamente fattibile. Essa è composta dai seguenti passaggi:

a) Costruire la *matrice dei conflitti* in cui compaiono tutte le possibili coppie di task tali che la coppia composta dai task t_i e $t_{i'}$ è in conflitto se i tempi di elaborazione dei due task si sovrappongono, *Conflict* $(t_i, t_{i'})=1$, oppure non lo è, *Conflict* $(t_i, t_{i'})=0$.
b) Costruire la *matrice delle assegnazioni* per individuare tutte le possibili assegnazioni tra tutte le coppie composte dai lavoratori w_j e dai task t_i. *Assignment* $(t_i, w_j)=1$ se il task t_i può essere assegnato all'operatore w_j e quindi se il lavoratore w_j possiede le qualità richieste per l'esecuzione del task t_i, altrimenti *Assignment* $(t_i, w_j)=0$.
c) Impostare le variabili di inizializzazione: $U(\xi_0)=\{t_1,\ldots,t_n\}$, dove U è l'insieme dei turni non ancora assegnati e $TCP_{wj}=0$, $\forall j$, dove TCP è il tempo cumulativo di ogni lavoratore w_j, ottenuto sommando tutti i tempi di elaborazione di tutti i task che gli sono stati attribuiti.

d) Ordinare i task non assegnati seguendo l'ordine ascendente dei tempi di inizio; nell'eventualità in cui due task inizino nello stesso istante, si segue l'ordine ascendente dei pedici dei task.

e) In accordo con la lista ordinata così ottenuta, cercare di attribuire i task non ancora assegnati, uno dopo l'altro, ai lavoratori disponibili, fintanto che tutti i task non siano stati considerati.

   Quando si cerca di eliminare il task t_i dall'insieme $U(\xi_0)$ bisogna ricercare il lavoratore w_j tale per cui:

   - ✓ *Assignment* (t_i , w_j)=1
   - ✓ *Conflict* (t_i , $t_{i'}$)=0, $\forall t_{i'} \in \left\{t_i | x_{t_i w_j} = 1, \forall i \; e \; i \neq i'\right\}$
   - ✓ w_j deve essere l'operatore con il più vasto tempo cumulativo tra tutti i lavori disponibili che già soddisfano le due precedenti condizioni

   Se nessuno degli operatori soddisfa queste tre condizioni, si deve inserire il task t_i di nuovo nell'insieme $U(\xi_0)$.

   A seguito di ogni attribuzione andata a buon fine si deve eliminare il task in oggetto dall'insieme $U(\xi_0)$ e aggiornare il TCP dell'operatore in questione.

f) Se l'insieme $U(\xi_0)$ non è ancora vuoto, passare ad una procedura di riassegnazione, applicata più volte fino a quando non vengono assegnati tutti i task o fino a quando non si verifica una situazione di ciclo infinito; tale procedura è composta dai seguenti step:
   i) Organizzare i task t_i in ordine ascendente rispetto agli istanti di inizio, in caso di task che hanno avvio contemporaneamente seguire l'ordine degli indici dei task.
   ii) Cercare di riassegnare i compiti t_i, uno ad uno, all'operatore w_j disponibile fino a quando tutti i task non sono stati assegnati. Il lavoratore w_j, al quale assegnare il task t_i, viene così individuato: si deve calcolare il tempo totale dei compiti in contrasto con t_i per ogni lavoratore w_j che presenta le qualifiche per poter eseguire il suddetto task t_i, quindi si deve evidenziare il lavoratore con il tempo totale massimo di conflitti e attribuire a questo il task t_i. Una volta fatta tale attribuzione si deve eliminare dall'elenco dei task assegnati a w_j quelli in contrasto con t_i.
   iii) Aggiungere tutte le attività rimosse in $U(\xi_0)$ e così procedere con quanto indicato negli step d) ed e). Se la condizione di terminazione non viene ancora

soddisfatta ripartire con la procedura di riassegnazione e quindi con un nuovo ciclo.

g) Concludere e valutare la soluzione iniziale ξ_0 e indicare il numero di lavoratori impiegati mediante il valore di $|S(\xi_0)|$ e l'elenco dei task non ancora assegnati mediante gli elementi dell'insieme $U(\xi_0)$.

3.2.Fase due: utilizzo dell'algoritmo di greedy iterato per ottenere un miglioramento della soluzione iniziale

L'euristico IG appartiene alla classe di meta-euristiche neighborhood search, denominata VLNS, cioè alla classe di strategie di ricerca locale atte ad individuare miglioramenti per le soluzioni ottenute nel primo livello. Essendo un algoritmo di ricerca locale si basa su vicinati sempre più ampi, tanto da individuare ad ogni passaggio una soluzione migliore rispetto a quella candidata e ripartire da essa per riprendere l'esplorazione in un vicinato più piccolo. Questo algoritmo meta-euristico, definito efficace ed efficiente, è stato proposto da Jacobs e Brusco (1995, pp.1129-1140).
La procedura proposta da questo algoritmo è simile al R&R, un metodo basato principalmente su due componenti, una procedura di distruzione e mutazione (Ruin) e una di ricostruzione nonché miglioramento (Recreate). La differenza principale tra questi due approcci è che il metodo R&R ottiene una nuova possibile soluzione mediante la distruzione di una considerevole parte della soluzione in carica, mentre l'algoritmo IG la ottiene mediante la distruzione di solo una piccola parte della soluzione in carica.
Ruiz e Stützle (2007, pp.2033-2049) hanno sviluppato un euristico IG che ha risolto con successo il flusso di permutazioni relativo ai problemi di pianificazione e che ha messo in crisi l'euristico IG tradizionale; tale risultato negli ultimi anni è stato applicato per ottenere un vasto numero di soluzioni all'avanguardia relative a problemi di scheduling.
In recenti studi, Ruiz e Stützle (2008, pp.1143-1159), Ying (2008, pp.348-354), Ying (2009, pp.810-817), Ying et al. (2009, pp.7087-7092) hanno portato all'integrazione di tecniche di simulated annealing.

> La Simulated Annealing è una tecnica meta-euristica che si basa sull'analogia tra il processo fisico di tempra dei solidi e la risoluzione di problemi di ottimizzazione combinatoria. La tempra consiste in un processo fisico attraverso il quale un solido, immerso in un bagno caldo, viene progressivamente riscaldato aumentando la temperatura del bagno fino al punto di fusione, in corrispondenza del quale le particelle del sistema si distribuiscono in maniera casuale. Successivamente si provvede al

raffreddamento attraverso il lento abbassamento della temperatura del bagno. In questo modo si raggiunge uno stato di equilibrio termico caratterizzato da un valore dell'energia interna minima. Così, la Simulated Annealing deriva dall'algoritmo di Metropolis, sviluppato per simulare il processo fisico di tempra dei solidi.

A partire dallo stato corrente S, alla temperatura *T* e con energia E, definito dalle posizioni delle molecole che lo costituiscono, viene applicata una perturbazione consistente nello spostamento di una molecola in una nuova posizione scelta casualmente. In tal modo il sistema raggiunge un nuovo stato, *S'*, con un diverso valore di energia E'. Il nuovo stato viene accettato come stato corrente.

In analogia, per la risoluzione di problemi di ottimizzazione combinatoria, fissato un valore iniziale di *T* ed individuata una soluzione iniziale come soluzione corrente, l'algoritmo individua, attraverso l'applicazione casuale di una mossa, una soluzione nell'intorno della soluzione corrente. Se la soluzione esaminata è migliore di quella corrente è sicuramente accettata, altrimenti, se peggiore, viene accettata con una probabilità definita da una funzione che dipende dal parametro di controllo.

L'operazione descritta viene eseguita un certo numero di volte al termine di ognuna delle quali il valore *T* viene abbassato ed il processo riprende allo stesso modo. L'algoritmo termina quando il parametro *T* raggiunge un valore minimo prestabilito.

3.2.1.Procedura dell'euristico IG proposto

I dettagli della procedura relativa all'euristico di greedy iterato possono essere descritti come segue:

3.2.1.1.Generazione di una soluzione iniziale: si considera una soluzione iniziale ξ_0 coincidente con quella ottenuta in chiusura della prima fase.

Settare: $\xi_{best} = \xi_0$

$\xi^* = \xi_0$

dove ξ_{best} è la soluzione migliore, ξ^* è la soluzione in carica.

3.2.1.2.Fase di distruzione e costruzione:

a) Evidenziare α task in maniera random tra quelli presenti in ξ^*, eliminarli da ξ^* e inserirli nell'elenco che contiene tutti i task rimossi ξ_R nell'ordine in cui sono stati evidenziati, nel frattempo impostare come soluzione parziale ξ_D l'insieme ξ^* ottenuto dopo la rimozione degli α task.

b) Successivamente riassegnare i task dalla lista ξ_R alla lista ξ_D finché non si ottiene una nuova soluzione in carica ξ^*_{new}. Per riassegnare un task t_i da ξ_R ad un lavoratore di ξ_D bisogna cercare il lavoratore w_j tale per cui:
   - ✓ *Assignment (t_i , w_j)=1*;
   - ✓ *Conflict (t_i, $t_{i'}$)=0*, $\forall t_{i'} \in \left\{t_i | x_{t_i w_j} = 1, \forall i \; e \; i \neq i'\right\}$;
   - ✓ w_j deve essere l'operatore con il più vasto tempo cumulativo tra tutti i lavoratori che già soddisfino le due precedenti condizioni.

c) Porre:

|U(ξ^*)|= numero dei task non assegnati della soluzione ξ^*;

|U(ξ^*_{new})|= numero dei task non assegnati della soluzione in carica ξ^*_{new};

|U(ξ_{best})|= numero dei task non assegnati della soluzione migliore ξ_{best};

|S(ξ^*)|= numero dei task assegnati della soluzione ξ^*;

|S(ξ^*_{new})|= numero dei task assegnati della soluzione in carica ξ^*_{new};

|S (ξ_{best})|= numero dei task assegnati della soluzione migliore ξ_{best}.

Ora:

se |U(ξ^*_{new})|=0 e |U(ξ^*)|≠0, allora

poni ξ^* come ξ^*_{new}

altrimenti se |S(ξ^*_{new})|≤ |S(ξ^*)| e |U(ξ^*_{new})|≤|U(ξ^*)|, allora

poni ξ^* come ξ^*_{new}

altrimenti se |S(ξ^*_{new})|= |S(ξ^*)| e |U(ξ^*_{new})|>|U(ξ^*)|, allora

calcola ΔE =|U(ξ^*_{new})|-|U(ξ^*)|; genera un numero random r compreso tra 0 e 1 (estremi esclusi);

se r<$e^{(-\frac{\Delta E}{T})}$ allora poni ξ^* come ξ^*_{new}

altrimenti se |S(ξ^*_{new})|>|S(ξ^*)|, allora

calcola ΔE =|S(ξ^*_{new})|-|S(ξ^*)|; genera un numero random r compreso tra 0 e 1 (estremi esclusi);

se r<$e^{\left(-\frac{\Delta E}{T}\right)}$ allora poni ξ^* come ξ^*_{new}

altrimenti scarta la soluzione generata ξ^*_{new}

d) se |U(ξ^*)|=0 e |S(ξ^*_{new})|>|S(ξ^*)|, allora poni ξ_{best} come ξ^*

altrimenti se |U(ξ^*)|=0 e |U(ξ_{best})|≠0, allora ξ_{best} come ξ^*

3.2.1.3.Criteri di terminazione

L'euristico IG proposto trae vantaggi dalla tecnica della riduzione della temperatura nella fase di simulated annealing. La temperatura corrente T viene ridotta rispetto alla temperatura iniziale T_0 e ad ognuna delle I_{iter} iterazioni, dove I_{iter} è il totale di iterazioni prefissato, essa si riduce di un valore prefissato, secondo la relazione:

$$T \leftarrow \lambda T \text{ dove } 0 < \lambda < 1$$

L'algoritmo itera con le fasi di distruzione e costruzione espresse al punto 3.2.1.2 fintanto che il numero di riduzioni di temperatura non è maggiore del numero massimo di riduzioni prefissato ($N_{non\text{-}improving}$).

Riassumendo, nello step 3.2.1.1 si genera una soluzione iniziale ξ_0 che è frutto della procedura descritta nella fase uno per l'algoritmo euristico costruttivo. Negli step a) e b) del paragrafo 3.2.1.2 sono state descritte le procedure per applicare la tecnica di costruzione e distruzione della soluzione, che insieme possono essere viste come un meccanismo di perturbazione. Lo step c) dà in uscita una nuova soluzione ξ^*_{new}, mediante l'individuazione di cinque possibili situazioni:

1) Se ξ^*_{new} fattibile e ξ^* non fattibile, quindi se |U(ξ^*_{new})|=0 e |U(ξ^*)|≠0, allora ξ^* sarà sostituita da ξ^*_{new};
2) Se il numero di task assegnati e dei task non assegnati di ξ^*_{new} è lo stesso o inferiore rispetto a quello di ξ^*, ad esempio se |S(ξ^*_{new})|≤|S(ξ^*)| e |U(ξ^*_{new})|≤|U(ξ^*)|, allora ξ^* sarà sostituita da ξ^*_{new}, indipendentemente se le soluzioni sono fattibili o non fattibili;

3) Se il numero di task assegnati da ξ^*_{new} è uguale a quello relativo la soluzione ξ^* e il numero dei task non assegnati da ξ^*_{new} è maggiore di quello relativo la soluzione ξ^*, ad esempio se $|S(\xi^*_{new})| = |S(\xi^*)|$ e $|U(\xi^*_{new})| > |U(\xi^*)|$, si applica la funzione di Boltzmann per far sì che la soluzione in carica sfugga dai minimi locali indipendentemente dal fatto se ξ^*_{new} e ξ^* siano fattibili o non fattibili. Ciò si ottiene mediante la generazione di un numero random $r \in [0; 1]$ e la sostituzione di ξ^* con ξ^*_{new} se $r < e^{(\frac{\Delta E}{T})}$, dove $\Delta E = |U(\xi^*_{new})| - |U(\xi^*)|$ e T è la temperatura corrente;
4) Se il numero di task assegnati dalla soluzione ξ^*_{new} è maggiore di quello relativo la soluzione ξ^*, ad esempio se $|S(\xi^*_{new})| > |S(\xi^*)|$, sarà generato un numero random $r \in [0; 1]$ e ξ^* sarà sostituita con ξ^*_{new} se $r < e^{(\frac{\Delta E}{T})}$, dove $\Delta E = |S(\xi^*_{new})| - |S(\xi^*)|$;
5) Nei restanti casi la soluzione generata ξ^*_{new} sarà scartata.

Nello step finale d) se ξ^* è fattibile e il numero di task assegnati dalla soluzione ξ^* è minore del numero di task assegnati da ξ^*_{new}, ξ_{best} sostituirà ξ^*.

3.3.Fase tre: risoluzione della formulazione BIP per ottenere una soluzione ottima

Per abbreviare il tempo di elaborazione richiesto per la soluzione del BIP, si pone la soluzione ottenuta mediante l'algoritmo euristico IG come soluzione iniziale e poi utilizzando Gurobi (versione 5.02) si ottiene l'ottimo. La soluzione ottenuta dall'euristico IG agisce come limite superiore della funzione obiettivo, di conseguenza si avrà un netto ridimensionamento dell'insieme delle soluzioni ed il problema SMPTSP può essere, così, risolto in maniera efficiente. Questo risulta essere un vantaggio significativo, soprattutto per i problemi su vasta scala.
Ma vi è anche la possibilità che le soluzioni generate dalla fase due o dalla fase uno non siano fattibili, in questo caso Gurobi rigetta la soluzione iniziale e cerca di generare esso stesso una soluzione fattibile durante la procedura di ricerca.

CAPITOLO 4

Esempio di applicazione dell'algoritmo trifase

Per testare la fattibilità dell'algoritmo, soprattutto per quanto concerne le prime due fasi, è stato codificato un programma che implementa una versione semplificata del problema. Per rendere l'intera elaborazione più semplice si è ipotizzato che i lavoratori fossero privi di qualifiche e di conseguenza fossero in grado di svolgere qualunque mansione, quindi nella procedura di assegnazione è venuta meno la fase di costruzione della matrice delle assegnazioni.
Inoltre, il problema viene testato con un numero ridotto di attività, dieci al massimo, e un numero ridotto di lavoratori, anch'essi dieci al massimo.

La codifica è stata realizzata mediante il software di codifica DEV C++[4] 5.7.1 che utilizza il compilatore TDM-GCC 4.8.1 e il linguaggio in cui è scritto il programma è C puro.

4.1.Codifica della procedura di riassegnazione

La procedura di riassegnazione viene applicata a conclusione della fase uno nel caso in cui l'insieme dei task non assegnati U(ξ_0) non risulti essere completamente svuotato.

```
q=0;
for(i=0;i<natt;i++) if(attivita[i].lid<0) ch=1;
while(ch){
    ch=0;
    if(c==nlav) break;
    else{
        for(i=0;i<natt;i++){
        if(attivita[i].lid==(-1)){
            for(j=0;j<nlav;j++) totconf[j]=0;
            for(k=0;k<nlav;k++){
            nt=lavoratore[k].n;
            for(j=0;j<nt;j++){
            if(conflitti[attivita[i].id][lavoratore[k].idatt[j]]==1){
                if(attivita[i].inizio>attivita[lavoratore[k].idatt[j]].inizio)
                totconf[k]+=attivita[i].inizio-attivita[lavoratore[k].idatt[j]].inizio;
                else totconf[k]+=attivita[i].inizio-attivita[lavoratore[k].idatt[j]].inizio;
                if(attivita[i].fine>attivita[lavoratore[k].idatt[j]].fine)
                totconf[k]+=attivita[i].fine-attivita[lavoratore[k].idatt[j]].fine;
                else totconf[k]+=attivita[i].fine-attivita[lavoratore[k].idatt[j]].fine;
              }
             }
            }
```

[4] DEV-C++ è un IDE (Integrated Development Environment), cioè un programma che mette a disposizione un editor, per scrivere l'intero programma sorgente in C++, un compilatore per compilarli linguaggio macchina e un debug che permette di rilevare e correggere gli errori del programma sorgente.

```
            }
            }
            k=max(totconf,nlav); //lavoratore a cui togliere attività
            nt=lavoratore[k].n;
            for(j=0;j<nt;j++){
            if(conflitti[attivita[i].id][lavoratore[k].idatt[j]]==1){ //rimuovo le attivita' in conflitto
              attivita[lavoratore[k].idatt[j]].lid=(-1);
              lavoratore[k].tcp-=(attivita[lavoratore[k].idatt[j]].fine-attivita[lavoratore[k].idatt[j]].inizio);
              lavoratore[k].idatt[j]=(-1);
              }
            }
            for(j=0;j<nt;j++){
              if(lavoratore[k].idatt[j]==(-1)){
                  lavoratore[k].idatt[j]=attivita[i].id;
                  attivita[i].lid=lavoratore[k].id;
                }
            }
          }
      }
      }for(j=0;j<natt;j++){if(attivita[j].lid<0) c++; ch=1;}
   }
}  //FINE RIASSEGNAZIONE
```

4.2.Codifica della fase di distruzione

Con la procedura di distruzione si eliminano in maniera random un numero di task pari ad α che è stato calcolato mediante l'espressione $\alpha = \left\lfloor \beta * \frac{numero\ di\ task}{numero\ di\ turni} \right\rfloor$.

```
t_lavoratore pckl[nlav];
struct t_lavoratore ka[natt];
printf("\nInserisci n (non migliorabilita''): ");
scanf("%d",&n);

for(x=0;x<n;x++){

printf("\nInserisci beta: ");
scanf("%d",&beta);
alfa=beta*(int)natt/nlav;
int casuali[alfa];

for (i=0; i<alfa; i++){
 ok = 0;
     while(!ok){
casuali[i] = rand()%natt;
ok = 1;
for(j=0; j<i; j++){
 if(casuali[i] == casuali[j]) ok = 0;
}
}
}
```

```
for(i=0;i<alfa;i++){

        k=attivita[casuali[i]].lid;
        while(s==1){
            for(j=0;j<lavoratore[k].n;j++)
            if(lavoratore[k].idatt[j]==casuali[i]){
                lavoratore[k].idatt[j]=(-1);
                s=0;
            }
        }
        s=1;

        lavoratore[k].tcp-=(attivita[casuali[i]].fine-attivita[casuali[i]].inizio);
        attivita[casuali[i]].lid=(-2);
        nt++;

}

printf("\nProcessi rimossi : %d\n",alfa);
for (i=0;i<natt;i++){
    if(attivita[i].lid==(-2)) printf("ID Attivita\' %2d inizio %3d fine %3d\n",attivita[i].id,attivita[i].inizio,attivita[i].fine);
}
```

4.3.Codifica della procedura di confronto tra il risultato della fase uno e il risultato della fase due

Si deve mettere in evidenza un'ulteriore semplificazione introdotta in questa fase: la variabile T viene inserita da tastiera.

4.3.1.Caso uno: ξ^*_{new} fattibile e ξ^* non fattibile

```
if(((nonassegn(bcka,natt)==0)&&(nonassegn(attivita,natt)!=0))){
    for(j=0;j<natt;j++) attivita[j]=bcka[j];
    for(j=0;j<nlav;j++) lavoratore[j]=bckl[j];
```

4.3.2.Caso due: il numero di task assegnati e non assegnati di ξ^*_{new} è lo stesso o inferiore rispetto a quello di ξ^*

```
}else if(((assegn(bcka,natt))<=(assegn(attivita,natt)))&&((nonassegn(bcka,natt))<=(nonassegn(attivita,natt)))){
    for(j=0;j<natt;j++) attivita[j]=bcka[j];
    for(j=0;j<nlav;j++) lavoratore[j]=bckl[j];
```

4.3.3. Caso tre: il numero di task assegnati da ξ^*_{new} è uguale a quello di ξ^* mentre il numero dei task non assegnati è maggiore

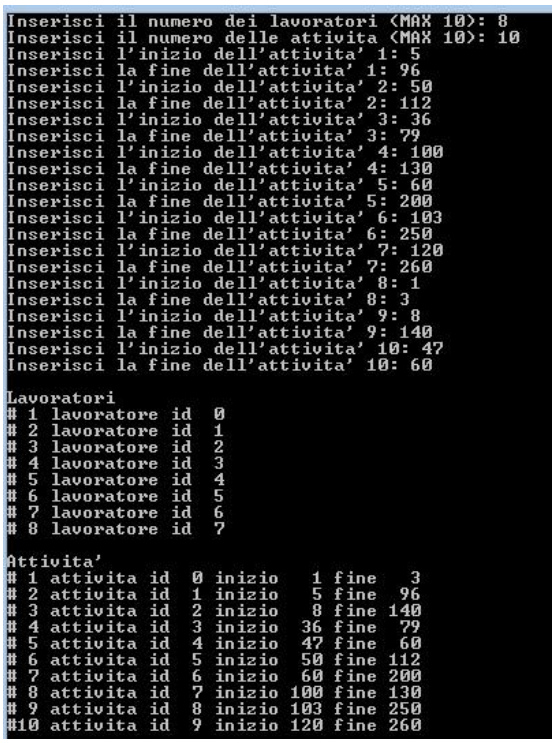
```
}else if(((assegn(bcka,natt))==(assegn(attivita,natt)))&&((nonassegn(bcka,natt))>(nonassegn(attivita,natt)))){
    printf("\nInserisci T: ");
    scanf("%d",&t);
    r=rand()%2;
    e=nonassegn(bcka,natt)-nonassegn(attivita,natt);
    esponente=(-e/t);
    esp=pow(2.71828,esponente);
    if(r<esp){
    for(j=0;j<natt;j++) attivita[j]=bcka[j];
    for(j=0;j<nlav;j++) lavoratore[j]=bckl[j];
    }
```

4.3.4.Caso quattro: il numero di task assegnati dalla soluzione ξ^*_{new} è maggiore di quello di ξ^*

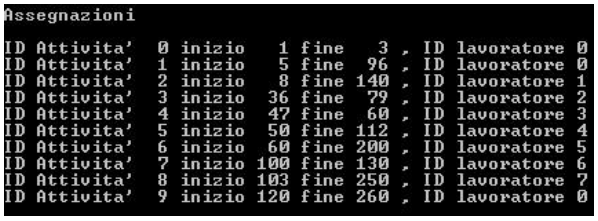
```
}else if((assegn(bcka,natt))>(assegn(attivita,natt)))
{
    printf("\nInserisci T: ");
    scanf("%d",&t);
    r=rand()%2;
    e=assegn(bcka,natt)-assegn(attivita,natt);
    esponente=(-e/t);
    esp=pow(2.71828,esponente);
    if(r<esp){
    for(j=0;j<natt;j++) attivita[j]=bcka[j];
    for(j=0;j<nlav;j++) lavoratore[j]=bckl[j];
    }
```

4.4.Analisi dei risultati

Partendo dall'inserimento dei seguenti dati:

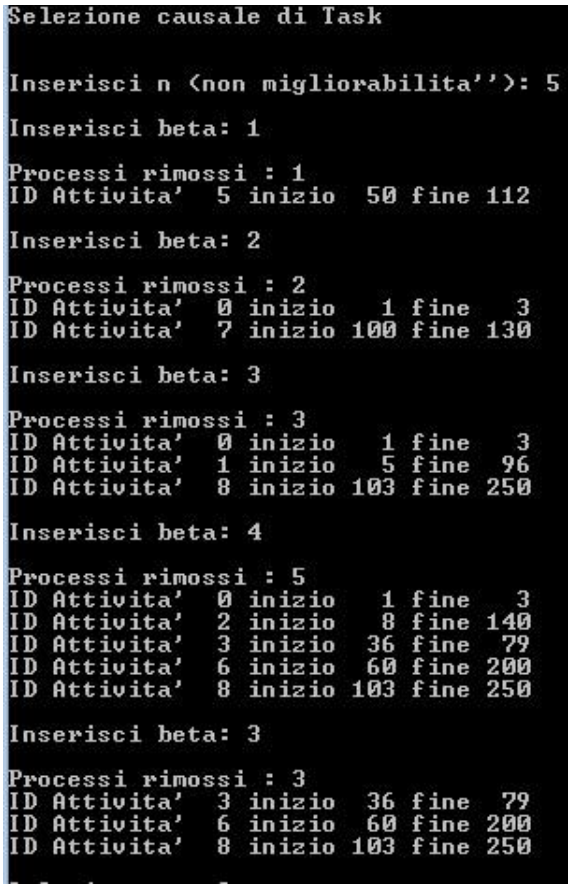


Si ottiene una prima soluzione parziale:

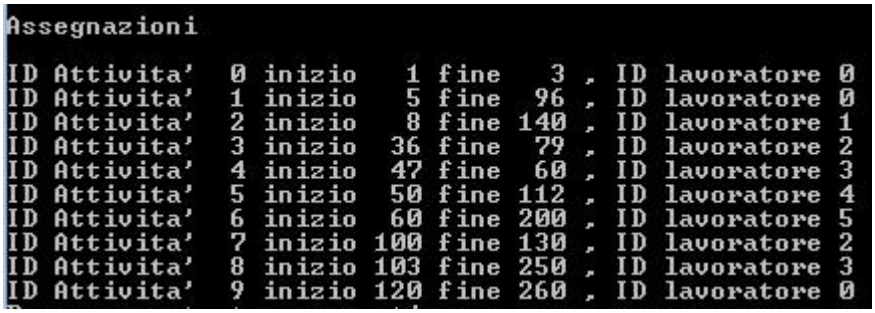


Nella soluzione parziale sono stati utilizzati tutti i lavoratori disponibili, come si evince dall'immagine sopra riportata.

Successivamente, viene richiesto l'inserimento da tastiera dei valori relativi i parametri $N_{non\text{-}improving}$ e β, a seguito del quale si ha la rimozione di α task in maniera random.



Ed infine si ottiene la soluzione finale, a seguito dell'esecuzione della seconda fase dell'algoritmo:



Dai risultati emerge un netto miglioramento, se pure le dimensioni del problema sono piccole, tra la soluzione ottenuta dall'esecuzione della fase uno, che prevede l'impiego degli otto operatori a disposizione e quella successivamente ottenuta dall'esecuzione della fase due, che invece, prevede l'impiego di soli sei lavoratori.

CAPITOLO 5

Confronto tra l'algoritmo trifase e il VAWA

Per valutare l'efficienza e l'efficacia dell'approccio proposto sono state eseguite delle comparazioni e delle simulazioni computazionali mediante l'esecuzione di un problema settato con 137 istanze, presentate da Krishnamoorthy et al. (2012, pp.34-48). Tali istanze sono state generate sulla base della valutazione di cinque aspetti: il numero di turni, il numero di mansioni, la lunghezza delle attività, il livello di polivalenza e quello di tenuta degli operatori.

Il numero di turni è stato valutato nel seguente range (22; 420), mentre il numero di attività nel range (40; 2105). La durata delle attività è risultata essere di tre tipi: lunga, media e breve e di conseguenza poteva essere generata, rispettivamente per i tre casi, da una distribuzione triangolare[5] del tipo: (200, 300, 400), (50, 200, 250) e (50, 100, 200). Nello specifico, i problemi che presentano task dalla lunga durata sono semplici da risolvere in quanto il numero di turni necessari sarà inferiore rispetto al caso di task con durata più breve, di conseguenza sono stati solo valutati task con livelli di durata più contenuti, per raggiungere un certo livello di generalità. I livelli di polivalenza usati sono stati due: gli operatori possono essere di primo livello e quindi essere in grado di svolgere i 2/3 delle attività oppure di secondo e dunque svolgere solo 1/3 delle attività.

L'algoritmo trifase è stato codificato in C++ su un calcolatore con sistema operativo Windows 7 (64 bits) ed eseguito su un processore Intel_ Core™ i5-3317U @1.70 gigahertz, 4 cores e 4 gigabytes di RAM.

I parametri che sono stati utilizzati nelle prime sperimentazioni hanno assunto i seguenti valori:

- $\lambda = 0.93, 0.95, 0.97, 0.99$
- I_{iter}=10000, 20000, 30000, 40000
- $N_{non\text{-}improving}$=30, 50, 70, 100
- T_0=0.4, 0.6, 0.8, 1.0, 1.2, 1.5

[5] La distribuzione triangolare è una distribuzione di probabilità continua in cui la funzione di densità di probabilità descrive un triangolo.

- $\alpha = \left\lfloor \beta * \frac{numero\ di\ task}{numero\ di\ turni} \right\rfloor$ con β = 1.0, 1.5, 2.0, 2.5, 3.0

Gli esperimenti pilota hanno dimostrato che la selezione dei parametri ha influenzato la qualità delle soluzioni dell'algoritmo trifase. Ad esempio, il numero di turni o di lavoratori utilizzati e quindi il valore della funzione obiettivo diminuisce quando il numero totale di iterazioni aumenta, fino ad un valore di iterazioni massimo oltre il quale non è possibile ottenere ulteriori miglioramenti della funzione obiettivo.

La comparazione è stata effettuata tra le soluzioni ottenute dall'applicazione dell'algoritmo trifase, nell'ambiente sopra descritto e con i parametri sopra evidenziati, e le soluzioni ottenute con l'algoritmo VAWA, implementato su un processore Xeon da 2.93 gigahertz, con 4 MB di cache, 64 gigabytes di RAM, 16 cores e limite massimo di tempo di calcolo impostato a 1800 secondi.

Ogni istanza di prova è stata risolta una volta per la fase uno, una per la fase due, una per la fase tre, un'altra per le fase uno e tre (quindi la fase tre considerava come soluzione iniziale il risultato della fase uno) e infine un'ultima per l'intero algoritmo trifase proposto. I confronti sono stati effettuati:

a) nel caso in cui si ottengono soluzioni ottime mediante l'esecuzione dell'intero algoritmo e dunque delle tre fasi combinate;
b) nel caso in cui si ottenga, sempre con l'esecuzione dell'intero algoritmo, una soluzione fattibile e un LB[6] (lower bound) della soluzione ottima;
c) nel caso in cui si ottenga una soluzione fattibile, ma non un LB della soluzione ottima.

Le valutazioni di seguito presentate si basano sui dati contenuti all'interno delle tabelle presenti in appendice, in cui si mostrano i risultati computazionali dell'esecuzione delle varie fasi nei tre casi[7].

5.1.Valutazione del confronto nel caso di soluzioni ottime

Mediante l'algoritmo trifase si ottengono 101 soluzioni ottimali su 137 prove, mentre con la procedure VAWA il risultato è pari a 94 su 137. È evidente che questi risultati

[6] Il lower bound si ottiene utilizzando Gurobi MIP; esso è un valido limite inferiore della soluzione del problema. La differenza tra la soluzione fattibile e il lower bound è costituito da un gap che tende a zero una volta raggiunto l'ottimo.
[7] I file delle istanze la cui esecuzione da dato i risultati sopraindicati sono consultabili nella libreria presente al seguente link: http://people.brunel.ac.uk/~mastjjb/jeb/orlib/ptaskinfo.html, mentre le tabelle sono consultabili in Shih-Wei Lin & Kuo-Ching Ying (2014), pp.331-333.

dimostrano che l'algoritmo trifase riesce ad ottenere più soluzioni ottimali rispetto al VAWA, nello stesso intervallo temporale di calcolo. Inoltre, dal confronto fra i vari risultati ottenuti con l'esecuzione delle fasi che partono da una soluzione inziale e di quelle senza soluzione inziale si è messo in evidenza un aspetto della formulazione BIP dello SMPTSP: la risoluzione può essere efficientemente migliorata se si utilizza una soluzione iniziale generata da un euristico efficace.

5.2.Valutazione del confronto nel caso di soluzioni fattibili e LB della soluzione ottima

L'algoritmo trifase ha prodotto soluzioni fattibili di alta qualità molto vicini al LB in 29 casi su 137; ma anche mediante l'esecuzione della sola fase uno, della sola fase due, della sola fase tre o delle fasi uno e tre combinate si sono ottenute soluzioni più fattibili rispetto a quelle ottenute con l'implementazione dell'euristico VAWA.

5.3.Valutazione del confronto nel caso di soluzioni fattibili ma senza LB della soluzione ottima

In 7 casi su 137 non si è potuto calcolare LB entro l'intervallo di tempo prestabilito di 1800 secondi. In ogni caso le soluzioni fattibili ottenute nella fase tre entro il tempo sopraindicato sono state nettamente migliori di quelle ottenute nella fase due, così come quelle ottenute dall'applicazione dell'algoritmo VAWA sono state peggiori anche rispetto alle soluzioni delle singole fasi e delle fasi uno e tre combinate. La principale differenza tra la soluzione ottenuta dall'esecuzione della sola fase uno e la soluzione ottima, calcolata in numero di turni o lavoratori impiegati, è pari a tre, mentre la differenza principale tra la soluzione ottenuta dall'esecuzione della sola fase due e la soluzione ottima è pari a zero.

5.4.Comparazione generale tra l'euristico IG e l'algoritmo VAWA

Le soluzioni iniziali prodotte dall'algoritmo euristico costruttivo sono state non fattibili in solo 2 casi su 137 e le soluzioni ottenute dall'euristico costruttivo sono, soprattutto su larga scala, migliori di quelle ottenute dall'algoritmo VAWA. Inoltre su 137 test la fase due ha ottenuto 65 soluzioni migliori, 58 uguali e solo 15 peggiori rispetto a quelle ottenute con la procedura del VAWA. Quindi solo 36 soluzioni sono state migliorate con

l'esecuzione dell'intero algoritmo trifase, come si può notare dai dati precedentemente riportati. Per concludere, l'euristico proposto IG può essere preso in considerazione nei casi in cui si vogliono ottenere dei buoni risultati, anche se non ottimi, in tempi più brevi (siamo di fronte a tempi di elaborazione di 0,02 secondi per l'esecuzione dell'euristico IG per problemi di larga scala). Anche i tempi di elaborazione dell'approccio euristico sono nettamente inferiori rispetto a quelli del VAWA.

Per comparare ulteriormente le prestazioni dell'euristico IG con quelle del VAWA si valuta il *tasso di errore (RER)* della soluzione ottenuta rispetto il lower bound per ogni singola istanza (eccetto per le 7 istanze per cui non è stato possibile calcolare tale valore).

Esso si calcola come segue:

$$RER = \frac{(W^h - LB)}{LB} \times 100\%$$

dove W^h denota il numero di turni/operatori utilizzati dall'algoritmo h e LB è il lower bound ottenuto dall'esecuzione dell'algoritmo per intero.

La media di RER sui 130 casi in cui è stato possibile calcolare LB è pari a 1,23% per l'euristico IG e 4,08% per l'algoritmo VAWA. Il massimo RER per l'algoritmo euristico è pari al 10,0% e per il VAWA è pari al 22,86%. Questi risultati hanno confermato che le prestazioni dell'euristico sono nettamente superiori a quelle dell'algoritmo già presente in letteratura e giustificano il carattere di robustezza attribuito a tale euristico.

CONCLUSIONI

Con questo lavoro si è voluto approfondire ogni singolo aspetto delle varie fasi dell'algoritmo, partendo da un'analisi accurata di ogni caratteristica e di ogni vincolo del problema denominato SMPTSP. Mediante la codifica di una versione semplificata, rispetto a pochi vincoli del tutto secondari per un esempio di applicazione, sono state testate principalmente l'efficacia e l'efficienza della fase uno e della fase due, riguardanti, rispettivamente, la procedura dell'euristico costruttivo e l'algoritmo di greedy iterato. Dai risultati ottenuti si evince quello che da tempo è stato enunciato nella letteratura, cioè l'algoritmo trifase è un algoritmo efficace ed efficiente, che già con l'esecuzione delle prime due fasi permette di ottenere dei miglioramenti di soluzioni fattibili del problema. Inoltre, è stato approfondito anche l'aspetto riguardante la comparazione di tale algoritmo con un altro esistente in letteratura con la finalità di mettere in evidenza che tale algoritmo risulta avere delle prestazioni nettamente superiori rispetto agli algoritmi precedentemente proposti.

BIBLIOGRAFIA

Jacobs L. W. & Brusco M. J., 1995, A local-search heuristic for large set-covering problems. *Naval Research Logistics*, vol.42, pp.1129–1140.

Krishnamoorthy M., Ernst A. T. & Baatar D., 2012, Algorithms for large scale shift minimisation personnel task scheduling problems. *European Journal of Operational Research*, vol.219, pp.34–48.

Krishnamoorthy M. & Ernst A. T., 2001, The personnel task scheduling problem.

In X. Yang K. L. Teo, & L. Caccetta (Eds.), Optimisation methods and applications, pp. 343–368.

Ruiz R. & Stützle T., 2007, A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, vol.177, pp.2033–2049.

Ruiz R. & Stützle T., 2008, An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research*, vol.187, pp.1143–1159.

Shih-Wei Lin & Kuo-Ching Ying, 2014, Minimizing shifts for personnel task scheduling problems: A three-phase algorithm. *European Journal of Operational Research*, vol.237, pp.323-334.

Ying K. C., 2008, Solving non-permutation flowshop scheduling problems by an effective iterated greedy heuristic. *International Journal of Advanced Manufacturing Technology*, vol.38, pp.348–354.

Ying K. C., 2009, An iterated greedy heuristic for multistage hybrid flowshop scheduling problems with multiprocessor tasks. *Journal of the Operational Research Society*, vol.60, pp.810–817.

Ying K. C., Lin S. W. & Huang, C. Y., 2009, Sequencing single-machine tardiness problems with sequence dependent setup times using an iterated greedy heuristic. *Expert Systems with Applications*, vol.36, pp.7087–7092.

APPENDICE

Tabella A.1 Risultati computazionali nel caso di soluzioni ottime

Instance name	VAWA		Phase I		Phase II		Phase III		Phases I + III		Phases I + II + III	
	W	CPU	W	CPU	W	CPU	W	CPU	W	CPU	W	CPU
Data_1_23_40_66	20	0.08	20	<0.01	20	1.42	20	0.03	20	0.03	20	1.45
Data_2_24_40_33	20	0.02	21	<0.01	20	1.26	20	0.02	20	0.03	20	1.28
Data_3_25_40_66	20	0.16	20	<0.01	20	1.44	20	0.03	20	0.05	20	1.45
Data_4_23_59_33	20	0.09	–	<0.01	20	2.00	20	0.05	20	0.06	20	2.03
Data_5_25_60_33	20	0.06	24	<0.01	20	1.85	20	0.06	20	0.06	20	1.86
Data_6_48_80_66	40	0.41	41	<0.01	40	2.51	40	0.13	40	0.16	40	2.62
Data_7_51_80_66	40	0.40	41	<0.01	40	2.53	40	0.14	40	0.16	40	2.65
Data_8_48_85_33	40	0.61	44	<0.01	40	2.23	40	0.06	40	0.08	40	2.29
Data_9_49_104_33	41	2.36	46	<0.01	40	3.90	40	0.30	40	0.33	40	4.06
Data_10_51_111_66	40	2.04	42	<0.01	40	3.81	40	0.45	40	0.52	40	4.20
Data_11_24_119_33	21	0.99	23	<0.01	21	5.34	20	0.17	20	0.23	20	5.49
Data_12_49_119_33	40	0.88	42	<0.01	40	3.26	40	0.19	40	0.23	40	3.40
Data_13_25_120_33	20	0.59	–	0.02	21	5.54	20	1.11	20	1.39	20	6.54
Data_14_75_124_33	60	2.61	62	<0.01	60	3.21	60	0.14	60	0.20	60	3.37
Data_15_72_126_33	60	1.93	65	<0.01	60	3.23	60	0.14	60	0.19	60	3.37
Data_16_75_131_66	60	6.55	62	<0.01	60	3.65	60	0.30	60	0.37	60	3.95
Data_17_23_139_66	20	2.84	22	<0.01	22	6.29	20	0.86	20	0.81	20	7.08
Data_18_48_160_66	40	1.17	42	<0.01	40	6.15	40	0.62	40	0.64	40	6.63
Data_19_97_160_33	80	1.04	81	<0.01	80	4.06	80	0.22	80	0.22	80	4.27
Data_20_99_163_33	80	3.07	84	<0.01	80	4.21	80	0.25	80	0.25	80	4.45
Data_21_93_175_33	80	15.71	85	<0.01	81	4.10	80	0.36	80	0.36	80	4.45
Data_22_47_180_66	40	6.43	44	<0.01	41	11.84	40	6.27	40	8.07	40	21.03
Data_23_74_180_66	60	2.18	61	<0.01	60	5.26	60	0.51	60	0.51	60	5.65
Data_24_110_200_33	100	10.27	102	<0.01	100	4.63	100	0.28	100	0.28	100	4.93
Data_25_120_200_33	100	2.75	103	<0.01	100	4.99	100	0.25	100	0.27	100	5.24
Data_26_116_203_66	100	64.14	101	<0.01	100	5.54	100	0.52	100	0.55	100	6.07
Data_27_49_204_66	40	6.39	42	<0.01	41	10.17	40	11.99	40	12.04	40	21.62
Data_28_75_208_66	60	9.71	61	<0.01	60	10.17	60	4.57	60	4.52	60	12.60
Data_29_22_219_66	20	5.44	22	<0.01	22	10.27	20	5.04	20	4.90	20	15.15
Data_30_25_219_66	20	2.94	23	<0.01	22	11.54	20	9.41	20	11.17	20	21.06
Data_31_90_230_66	80	11.30	81	<0.01	80	7.46	80	2.37	80	2.37	80	8.86
Data_32_70_236_66	60	34.82	60	<0.01	60	7.66	60	2.82	60	1.95	60	9.61
Data_33_76_240_66	60	4.37	63	<0.01	60	12.34	60	2.73	60	2.70	60	13.56
Data_34_152_240_33	120	19.66	120	<0.01	120	6.16	120	0.33	120	0.34	120	6.51
Data_35_171_280_33	140	25.80	140	<0.01	140	6.93	140	0.50	140	0.55	140	7.43
Data_36_175_280_33	140	30.10	142	<0.01	140	6.99	140	0.60	140	0.64	140	7.61
Data_37_145_321_33	121	51.00	124	<0.01	120	12.03	120	7.82	120	7.86	120	16.52
Data_38_147_347_66	120	213.93	120	<0.01	120	10.00	120	7.60	120	4.87	120	15.04
Data_39_45_351_66	41	49.48	42	0.02	41	24.77	40	46.89	40	38.92	40	94.15
Data_40_138_360_33	120	29.41	121	<0.01	120	11.09	120	1.79	120	1.78	120	12.01
Data_41_144_360_66	120	159.02	121	<0.01	120	12.28	120	2.20	120	2.31	120	14.57
Data_42_101_380_66	80	73.64	81	<0.01	80	12.87	80	46.08	80	45.96	80	30.90
Data_43_156_387_66	140	305.81	141	<0.01	140	10.39	140	14.52	140	14.66	140	19.33
Data_44_121_400_33	100	26.35	101	<0.01	100	15.71	100	4.13	100	4.13	100	18.21
Data_45_67_420_33	67	64.34	66	<0.01	64	13.93	60	90.01	60	89.70	60	97.78
Data_46_147_423_33	121	70.94	126	<0.01	122	15.69	120	60.29	120	59.77	120	75.64
Data_47_150_430_33	121	77.96	126	<0.01	121	17.80	120	72.43	120	71.84	120	111.82
Data_48_120_434_66	100	133.25	102	<0.01	101	19.64	100	78.48	100	77.28	100	97.45
Data_49_211_446_66	182	692.23	183	<0.01	182	13.00	180	130.20	180	129.56	180	141.74
Data_50_187_447_66	160	530.77	161	<0.01	160	12.23	160	36.21	160	36.64	160	31.68
Data_51_196_480_33	160	66.09	163	<0.01	160	13.34	160	2.81	160	2.87	160	16.21
Data_52_205_480_66	160	343.87	161	<0.01	160	13.88	160	14.24	160	14.82	160	21.59
Data_53_127_487_66	101	446.45	100	<0.01	100	16.04	100	119.84	100	44.56	100	60.70
Data_54_175_492_66	140	526.00	142	<0.01	140	20.62	140	115.92	140	113.74	140	51.96
Data_55_85_493_66	72	175.90	73	<0.01	71	28.35	70	289.68	70	350.57	70	394.39
Data_56_163_500_66	140	484.61	141	0.02	140	26.72	140	91.93	140	91.98	140	65.02
Data_57_88_508_66	72	301.70	73	0.02	71	19.30	70	386.46	70	385.61	70	402.32
Data_58_158_517_66	140	496.12	140	<0.01	140	16.58	140	68.25	140	38.45	140	54.71
Data_59_70_525_33	68	63.01	66	<0.01	64	22.46	59	1250.95	59	1335.50	59	258.79
Data_60_181_549_66	139	410.60	140	<0.01	139	19.87	139	158.42	139	158.24	139	78.39
Data_61_121_557_66	100	495.99	100	<0.01	100	20.25	100	454.49	100	69.89	100	90.35
Data_62_101_571_33	90	117.06	87	<0.01	82	37.41	80	561.82	80	565.45	80	470.72
Data_63_97_577_66	82	328.96	81	<0.01	80	24.66	80	525.02	80	902.45	80	90.86
Data_64_176_595_66	160	683.07	161	<0.01	160	18.94	160	136.17	160	135.71	160	75.91
Data_65_179_596_66	159	722.91	159	<0.01	159	19.25	159	82.54	159	53.84	159	73.12
Data_66_348_600_33	300	206.12	302	<0.01	300	16.07	300	4.40	300	4.57	300	20.61
Data_67_371_600_66	300	1308.60	300	<0.01	300	18.94	300	12.90	300	13.67	300	32.56
Data_68_359_613_66	300	1714.62	302	<0.01	300	22.73	300	21.64	300	23.29	300	46.04
Data_69_148_614_33	125	136.08	124	<0.01	121	27.49	120	270.45	120	272.20	120	295.92
Data_70_192_623_66	160	738.22	160	<0.01	160	19.98	160	94.72	160	57.43	160	76.35
Data_71_197_624_33	158	202.66	163	<0.01	159	23.84	158	189.93	158	190.73	158	212.57
Data_72_205_624_66	160	781.35	160	<0.01	160	20.55	160	146.66	160	58.72	160	77.83

Instance name	VAWA		Phase I		Phase II		Phase III		Phases I + III		Phases I + II + III	
	W	CPU	W	CPU	W	CPU	W	CPU	W	CPU	W	CPU
Data_73_155_661_66	123	762.24	122	<0.01	121	22.54	120	1114.62	120	1103.90	120	1127.04
Data_74_209_664_33	180	155.29	182	<0.01	181	17.35	180	118.62	180	118.77	180	135.74
Data_75_72_665_33	71	68.15	69	<0.01	66	22.32	61	1807.36	62	1800.16	60	498.20
Data_76_162_683_66	140	763.90	141	<0.01	140	45.29	140	353.34	140	352.16	140	165.50
Data_77_180_688_33	162	265.75	169	<0.01	163	24.09	160	495.47	160	487.09	160	511.76
Data_78_199_688_66	160	1128.88	162	<0.01	161	22.11	160	778.60	160	787.43	160	1113.70
Data_80_112_691_33	107	157.46	106	<0.01	102	31.00	99	1072.05	99	741.89	99	1334.86
Data_81_97_692_66	83	379.43	82	<0.01	81	28.13	80	1091.52	80	1091.18	80	1132.06
Data_83_222_700_66	180	809.02	180	<0.01	180	22.93	180	252.51	180	94.46	180	115.21
Data_84_136_718_66	120	497.86	121	<0.01	121	26.24	120	783.39	120	797.74	120	823.32
Data_85_217_720_66	180	752.00	181	<0.01	181	22.90	180	88.84	180	90.94	180	112.20
Data_86_178_721_33	146	231.99	145	<0.01	142	23.82	140	677.62	140	683.34	140	700.80
Data_87_203_735_33	174	347.47	176	<0.01	173	35.71	170	841.04	170	842.91	170	869.81
Data_88_137_777_66	123	681.48	120	<0.01	120	29.56	121	1801.43	120	176.92	120	205.19
Data_89_88_788_33	86	63.87	76	<0.01	73	48.72	70	732.75	70	641.74	70	500.00
Data_90_157_791_66	140	1339.59	140	<0.01	139	30.67	140	1802.30	140	1801.96	139	312.78
Data_91_147_851_66	124	994.49	119	<0.01	118	40.67	121	1802.29	119	1857.28	118	440.26
Data_95_204_882_33	177	426.44	173	<0.01	170	27.36	170	1032.27	170	1024.18	170	183.41
Data_96_98_886_66	83	590.10	80	<0.01	80	36.43	80	1623.42	80	185.66	80	219.94
Data_97_383_895_33	300	548.26	302	<0.01	300	30.58	300	43.18	300	44.44	300	62.60
Data_99_176_956_66	160	1447.34	161	<0.01	160	44.16	160	1285.65	161	1800.38	160	368.25
Data_100_194_956_66	160	1349.33	161	<0.01	160	34.99	160	730.05	160	720.17	160	272.81
Data_101_166_997_66	144	1800.02	141	<0.01	140	40.05	141	1800.10	141	1800.09	140	523.04
Data_103_348_1024_33	303	973.30	305	0.02	301	31.73	300	571.79	300	568.79	300	600.52
Data_113_141_1221_66	114	1452.83	111	<0.01	110	52.00	123	1800.38	111	1802.60	110	740.05
Data_116_205_1262_66	190	1800.04	177	<0.01	176	52.25	194	1800.62	177	1805.06	176	1497.99
Data_122_422_1358_66	348	1800.18	348	0.02	348	48.50	348	999.02	348	979.37	348	1096.15
Data_127_192_1472_66	185	1800.06	170	<0.01	170	64.48	183	1800.62	170	1057.81	170	1158.68
Data_128_207_1542_66	205	1800.10	178	0.02	178	63.16	194	1800.80	178	1643.71	178	1717.33

Tabella A.2 Risultati computazionali nel caso di soluzioni fattibili e LB della soluzione ottima

Instance name	VAWA			Phase I		Phase II		Phase III		Phases I + III		Phases I + II + III		
	W	CPU	LB_{VAWA}	W	CPU	W	CPU	W	CPU	W	CPU	W	CPU	LB
Data_79_94_689_33	93	88.24	78.43	86	<0.01	84	25.8	81	1800.06	80	1183.72	81	1800.03	80
Data_82_89_697_66	82	436.07	78.95	82	<0.01	81	28.78	80	472.10	80	775.99	81	1800.35	80
Data_92_126_856_66	106	719.08	96.86	99	<0.01	99	32.45	101	1803.10	99	1801.42	99	1801.54	98
Data_93_141_856_66	125	1065.65	117.62	120	<0.01	120	34.10	122	1801.93	120	1801.85	120	1801.93	119
Data_94_93_881_33	91	200.62	78.43	88	<0.01	84	42.73	81	1800.07	80	1643.76	82	1800.07	80
Data_98_91_896_33	90	184.02	78.12	86	0.02	84	38.05	81	1800.06	82	1800.18	82	1800.26	80
Data_102_179_997_66	147	1800.02	136.46	139	<0.01	139	38.49	151	1800.45	139	1800.37	139	1800.37	138
Data_104_181_1057_33	165	403.26	143.53	151	<0.01	150	30.12	151	1800.06	151	1801.24	150	1800.09	146
Data_105_173_1075_66	156	1800.02	147.56	151	<0.01	151	43.04	165	1800.43	151	1800.61	151	1800.38	150
Data_106_121_1096_33	113	335.04	98.14	106	<0.01	104	34.79	107	1800.06	106	1800.09	104	1800.03	100
Data_107_114_1112_33	112	633.83	98.39	106	<0.01	103	36.25	107	1800.04	103	1800.15	103	1800.12	100
Data_108_162_1115_33	145	486.52	125.38	137	<0.01	132	52.67	133	1800.13	133	1801.45	132	1800.13	128
Data_109_205_1115_33	176	553.69	154.30	163	<0.01	159	38.99	162	1800.09	162	1800.13	159	1800.09	157
Data_110_183_1143_66	167	1800.05	153.35	158	0.02	157	46.57	176	1800.59	158	1803.77	157	1800.57	155
Data_111_155_1211_33	155	646.94	138.82	145	<0.01	142	61.09	144	1829.09	144	1801.58	142	1800.99	139
Data_112_200_1213_33	194	602.51	167.06	174	<0.01	171	53.37	174	1800.07	174	1801.57	171	1800.10	169
Data_114_157_1227_33	157	915.58	136.42	143	<0.01	141	57.05	144	1800.06	143	1801.81	141	1800.04	138
Data_115_228_1257_33	199	922.25	175.09	183	0.02	178	54.85	206	1800.40	183	1801.85	178	1800.38	177
Data_117_192_1285_33	170	829.22	144.65	154	<0.01	150	45.94	174	1800.34	154	1800.33	150	1800.29	149
Data_118_180_1302_33	165	767.91	144.65	153	0.02	150	62.56	172	1800.51	153	1800.83	150	1800.41	147
Data_119_236_1335_33	208	1194.02	185.24	193	<0.01	189	58.53	214	1800.41	193	1805.39	189	1800.37	188
Data_120_228_1341_33	208	996.82	184.29	193	<0.01	190	50.97	214	1800.45	193	1800.51	190	1800.42	187
Data_121_147_1345_33	140	422.62	118.16	127	<0.01	123	55.51	131	1800.13	127	1801.16	123	1800.13	120
Data_123_187_1376_33	178	919.58	156.01	164	<0.01	160	64.71	186	1800.33	164	1800.30	160	1800.35	159
Data_124_198_1383_33	182	704.23	156.08	165	0.02	161	52.20	185	1800.45	165	1800.45	161	1800.46	158
Data_125_157_1448_33	152	594.30	127.61	135	0.02	132	65.85	155	1800.32	135	1800.32	132	1800.34	130
Data_126_193_1462_33	191	1529.38	164.78	172	0.02	170	54.27	193	1800.52	172	1800.49	170	1800.56	167
Data_129_233_1546_33	206	821.01	175.84	181	<0.01	180	50.98	205	1800.54	181	1800.55	180	1800.52	178
Data_133_211_1647_33	211	1800.02	181.66	191	0.02	189	55.61	NA	1800.71	191	1800.76	189	1800.71	185

Tabella A.3 Risultati computazionali nel caso di soluzioni fattibili ma senza LB della soluzione ottima

Instance name	VAWA			Phase I		Phase II		Phase III		Phases I + III		Phases I + II + III	
	W	CPU	LB_{VAWA}	W	CPU	W	CPU	W	CPU	W	CPU	W	CPU
Data_130_176_1562_66	145	1800.05	138.35	141	0.00	140	76.50	154	1800.13	141	1804.46	140	1800.13
Data_131_415_1610_33	359	1800.05	344.07	351	0.00	348	46.80	377	1800.67	351	1800.93	348	1800.21
Data_132_216_1645_33	211	1507.90	183.09	192	0.00	189	72.01	216	1800.29	192	1800.07	189	1800.29
Data_134_184_1776_66	179	1800.08	157.56	161	0.02	160	79.08	174	1800.17	161	1802.98	160	1800.23
Data_135_213_1988_33	206	1590.80	177.20	182	0.02	180	81.62	208	1800.10	182	1800.07	180	1800.10
Data_136_216_2000_66	216	1800.12	173.81	179	0.00	179	89.90	195	1800.24	179	1800.31	179	1800.26
Data_137_245_2105_33	223	1407.54	187.55	192	0.02	191	88.23	219	1800.12	192	1800.09	191	1800.12