UNIVERSITÀ DEGLI STUDI DI MILANO
Facoltà di Scienze Matematiche, Fisiche e Naturali
Dipartimento di Tecnologie dell'Informazione
Corso di Laurea in Informatica

UN ALGORITMO DI TABU SEARCH PER IL MAXIMUM DIVERSITY PROBLEM

Relatore: Prof. Roberto Aringhieri
Correlatore: Prof. Roberto Cordone

Tesi di Laurea di:
Yari Melzani
Matricola 634009

Anno Accademico 2004/2005

Indice

Capitolo 1

Introduzione

1.1 Definizione del problema

Il Maximum Diversity Problem (MDP) consiste nella scelta di m elementi da una popolazione di partenza N in modo da massimizzare la loro reciproca diversità. Il problema può trovare applicazione in una vasta gamma di ambiti, come la definizione di trattamenti medici, calendari d'esame, giurie, nonchè alla selezione di investimenti e al VLSI design.

Il Maximum Diversity Problem è NP-difficile [3]. Di conseguenza, a meno che $P = NP$, non è possibile trovare un algoritmo polinomiale con garanzia di ottimalità, diviene quindi necessario ricorrere ad algoritmi di tipo approssimato. Gli algoritmi di tipo esatto, permettono di risolvere istanze di dimensioni molto inferiori a quelle richieste dalle applicazioni reali.

1.2 Applicazioni

Il Maximum Diversity Problem permette di risolvere molte applicazioni reali quali il mantenimento dell'equilibrio biologico, lo sviluppo di linee difensive in grado di combattere potenziali malattie, la creazione di nuove varietà vegetali, il corretto ridimensionamento di un'azienda e la corretta composizione di una giuria.

1.2.1 Salvaguardia della Biodiversità

La biodiversità [9] indica una misura della varietà di specie animali e vegetali nella *biosfera* (parte del nostro pianeta nella quale si riscontrano le condizioni indispensabili alla vita animale e vegetale); essa è il risultato di lunghi processi evolutivi. L'evoluzione è il meccanismo che da oltre tre miliardi di anni permette alla vita di adattarsi al variare delle condizioni sulla terra e che deve continuare a operare perchè questa possa ancora ospitare forme di vita in futuro. La diversità della vita

sulla terra è costituita dall'insieme degli esseri viventi che popolano il Pianeta. Essa prende il nome di **biodiversità**, dall'inglese *biodiversity*, tale termine può essere tradotto "varietà della vita".

La biodiversità è intesa non solo come il risultato dei processi evolutivi, ma anche come il serbatoio da cui attinge l'evoluzione per attuare tutte le modificazioni genetiche e morfologiche che originano nuove specie viventi. La biodiversità si può considerare almeno in tre livelli diversi:

- a livello di geni (cromosomi) in una specie (o popolazione)
- a livello specie
- a livello di ecosistemi [1]

Le *caratteristiche morfologiche*, ovvero tutte le caratteristiche visibili degli esseri viventi come ad esempio il colore degli occhi e dei capelli dell'uomo, il colore del pelo dei gatti, sono esempi della varietà che esiste a livello di geni all'interno di ogni singola specie. La varietà di specie di farfalle che frequentano il nostro giardino, l'incredibile numero di fiori diversi che possono essere trovati in un campo sono esempi della biodiversità a livello di specie. Infine, la varietà di ambienti in una determinata area naturale è l'espressione della biodiversità a livello di *ecosistemi*.

La terra è popolata da numerosi esseri viventi, animali e vegetali che non conosciamo: oggi sono state classificate appena un milione di specie, mentre le stime elaborate dai biologi vanno dai 5 ai 10 milioni. Diventa, quindi, ancora più urgente e importante occuparsi della conservazione di specie e ambienti che rischiano di sparire per sempre a causa dell'uomo, ancora prima di essere scoperti. E' noto che alcuni *biomi*[2] risultano più importanti rispetto ad altri in termini di ricchezza di specie: le barriere coralline, gli estuari dei fiumi e le foreste tropicali che accolgono oltre la metà degli esseri viventi, pur ricoprendo il 6% della superficie terrestre, sono i più importanti. Affinché la diversità nell'ambito di una comunità biologica possa essere considerata una risorsa deve essere caratterizzata da un adeguato numero di specie, da un'alta valenza ecologica e da un legame con le condizioni ambientali. Inoltre è necessaria un'uniforme e approfondita conoscenza dei dati di base e la disponibilità di dati recenti.

La biodiversità è l'assicurazione sulla vita del nostro pianeta. Quindi la conservazione della biodiversità deve essere perseguita senza limiti poichè essa costituisce un patrimonio universale, che può offrire vantaggi immediati per l'uomo.

Supponiamo di dover selezionare un certo numero di specie (animali o vegetali) da una popolazione di esseri viventi al fine di costituire un bioma nel rispetto delle esigenze comuni dei membri che ne faranno parte (ad esempio esigenze climatiche

[1] L'ecosistema è costituito dall'insieme di tutti gli esseri viventi di un determinato ambiente fisico e delle relazioni che intercorrono sia tra loro che tra loro e l'ambiente fisico.

[2] Un bioma è un complesso di comunità animali e vegetali che, in una data area geografica, hanno raggiunto una relativa stabilità mantenuta dalle condizioni ambientali. I biomi si distinguono in biomi terrestri e biomi dell'idrosfera.

o nutrizionali). Non potendo mettere tutte le specie potenzialmente compatibili, bisogna selezionare quelle che hanno la maggiore diversità al fine di costituire un sistema equilibrato. Il problema può essere risolto definendo N come l'insieme della popolazione degli esseri viventi totale ed m come la dimensione ottimale di un bioma. Selezionare quindi il bioma con maggiore diversità consiste nel trovare una soluzione al problema MDP con parametri N e m .

1.2.2 Trattamenti medici

Per pianificare al meglio un trattamento contro le malattie è necessario differenziare il più possibile le linee difensive sia in termini di prevenzione che in termini di cura. In questo modo si riesce a proteggersi contro un largo spettro di agenti patogeni che ogni giorno attaccano il nostro organismo. Bisogna definire la diversità tra un trattamento ed un altro al fine di determinare, fra tutti i possibili trattamenti, qual'è il sottoinsieme massimamente efficace nella lotta contro le malatie. Posto N l'insieme di tutti i trattamenti possibili, la selezione del sottoinsieme di cardinalità m più efficace è un'applicazione del MDP.

1.2.3 La varietà nell'agricoltura

Nella selezione delle strategie per la selezione di nuove varietà vegetali, è di fondamentale importanza avvalersi di strategie in grado sfruttare criteri di qualità basati sulla diversità degli elementi appartenenti ai sistemi biologici. La qualità della selezione ottenuta è legata alla diversità fra gli elementi che la costituiscono. Al fine di massimizzare la qualità del risultato è necessario poter selezionare dei sottoinsiemi composti da elementi aventi la massima diversità in termini genetici. Se pensiamo N come l'insieme di tutte le possibili varietà e m la cardinalità del sottoinsieme contenente quelle che presentano fra loro la massima diversità, è evidente una possibile applicazione del MDP.

1.2.4 Corretto ridimensionamento di un'impresa

Il ridimensionamento di un'impresa comporta la ridefinizione sia del numero delle risorse umane che dei ruoli ad esse associate, nel rispetto delle necessità legate al tipo di impresa. Nel processo di riduzione delle risorse umane vanno classificate le figure professionali; va inoltre definita un'importanza e una reciproca diversità. L'idea consiste nell'eliminare ruoli simili fra loro mentre si cerca di privilegiare la presenza di figure professionali diversificate. Una volta fissato il nuovo numero m di figure necessarie per il corretto svolgimento dell'impresa, si tratta di selezionare un sottoinsieme M di cardinalità m dall'insieme delle figure, in modo da massimizzare la diversità dei ruoli scelti.

1.2.5 Composizione di una giuria

Quando si vuole costituire una giuria, in grado di esprimersi in merito ad una particolare questione, si cerca di raggruppare un certo numero di persone in grado di produrre un parere il più giusto ed equo possibile. Il problema della scelta dei componenti è difficile, se si vuole che la giuria rappresenti al meglio la popolazione si devono estrarre i giurati da tutte le sfere della società, per favorire la presenza di numerosi punti di vista molto diversi fra loro. Così facendo si mira ad ottenere un giudizio secondo giustizia e secondo equità. Una volta definito un modo per poter stabilire la diversità fra le possibili categorie di giurati, si tratta di estrarne m dalla popolazione totale in modo da massimizzare la loro reciproca diversità.

1.2.6 Diversity Data Mining

I DataBase Management Systems (DBMS), che gestiscono la memorizzazione e la manipolazione di grosse quantità di dati si, sono diffusi enormemente negli ultimi anni. Uno dei fattori trainanti è certamente la rivoluzione tecnologica, la quale ha sottolineato la crescente importanza delle attività legate alle basi di dati. Ormai non esiste più settore dell'indistria che non raccolga e conservi grosse quantità di dati. Queste attività, facilitate dai moderni sistemi informativi, hanno lo scopo di catturare dati che costituiranno utili informazioni al momento di una loro selezione e analisi. La diffusione delle **data warehouse** è stata fortemente influenzata dallo sviluppo di tecniche di **data mining**. Questi strumenti permettono in modo efficiente di fare statistiche e previsioni sulla base della grossa mole di dati memorizzati nelle data warehouse. Il processo di analisi prevede il riconoscimento di **pattern** e relazioni che altrimenti passerebbero inosservati. Il punto di forza delle tecniche di **data mining** è la possibilità di raffinare la ricerca di informazioni applicando semplici interrogazioni o brevi descrizioni sulla base di dati. Ad esempio, dato il risultato di un'interrogazione, si potrebbe richiedere quali sono gli m elemeneti, che la compongono, di massima diversità, dove la diversità per ogni coppia di elementi è definita rispetto al valore di un particolare attributo. Le dimensioni delle moderne basi di basi mettono chiaramente in evidenza l'impossibilità pratica di risolvere il problema all'ottimo. Un algoritmo che risolve il MDP potrebbe essere aggiunto come strumento di **data mining** allo scopo di identificare sottoinsiemi di cardinalità fissata della popolazione, aventi diversità massima. Tale strumento prenderebbe il nome di **Diversity Data Mining** (DDM) [10]. Chiamare la procedura DDM ad operare sul risultato di un'interrogazione può voler dire dover risolvere un MDP su una popolazione di dimensioni gigantesche. Ancora una volta viene messa in evidenza la necessità di trovare una soluzione approssimata, che permetta di trovare una soluzione con un tempo di calcolo ragionevole.

1.3 Descrizione ed obiettivi del lavoro

Dopo una completa analisi della letteratura, viene presentato un algoritmo di Tabu Search, con alcune varianti, per risolvere MDP. Le varianti prese in considerazione includono meccanismi per modellare una memoria sia a breve che a lungo termine per guidare l'esplorazione l'esplorazione dello spazio delle soluzioni. Le prestazioni degli algoritmi sono misurate valutando i singoli algoritmi, e confrontando i risultati tra loro e e con quelli in letteratura. Il confronto è stato effettuato su differenti categorie di istanze prese dalla letteratura e riguarda la qualità delle soluzioni determinate e l'efficienza computazionale.

L'obiettivo principale è quello di fornire un algoritmo capace di migliorare i risultati in letteratura sia in termini di qualità delle soluzioni trovate che in tempo di calcolo.

Capitolo 2

Maximum Diversity Problem

2.1 Formulazioni Matematiche

Dato un insieme $N = \{1, .., n\}$ di elementi e_i e una funzione **diversità** $d : (N \times N) \rightarrow \Re^+$ ed un numero naturale $m < n$, MDP consiste nel selezionare un sottoinsieme $M \subseteq N$, di cardinalità m, tale da massimizzare la somma delle diversità tra gli elementi scelti, misurata come $z(M) = \sum_{i,j \in M,\ i<j} d_{ij}$, dove d_{ij} è la distanza tra l'elemento e_i ed e_j. Per convenzione consideriamo nulla la diversità di un elemento da se stesso, $d_{ij} = 0$ se $i = j$. La diversità è simmetrica $(d_{ij} = d_{ji})$, mentre non si richiede che valga la disuguaglianza triangolare $(d_{ij} + d_{jk} \geq d_{ik})$.

Una prima formulazione non lineare per MDP, può essere la seguente: introduciamo n variabili binarie x_i, tali che $x_i = 1$ se $e_i \in M$, $x_i = 0$ altrimenti. Lo scopo del Maximum Diversity Problem è quello di massimizzare la somma z delle diversità definite tra gli elementi che compongono la soluzione M, ovvero

$$z = \max \sum_{i,j \in N,\ i<j} d_{ij} x_i x_j$$

con il vincolo che la cardinalità dell'insieme soluzione sia m ossia

$$\sum_{i \in N} x_i = m$$

Questa formulazione ha il vantaggio di essere molto semplice, tuttavia, non è utilizzabile per stendere un modello di programmazione lineare intera in quanto la funzione obiettivo è quadratica.

2.1.1 Una formulazione lineare intera

In questa sezione proponiamo una possibile formulazione lineare intera del Maximum Diversity Problem derivandola dalla precedente.

Introduciamo n^2 ulteriori variabili binarie y_{ij} che assumono il valore 1 se entrambi gli elementi i e j appartengono alla soluzione, mentre $y_{ij} = 0$ altrimenti. Ovvero $y_{ij} = 1 \Leftrightarrow x_i = x_j = 1 \Leftrightarrow x_i \cdot x_j = 1$. La funzione obiettivo diventa quindi $z = \max \sum_{i,j \in N,\ i<j} d_{ij} y_{ij}$. Il numero di elementi che appartengono alla soluzione è pari a m. Per eliminare il vincolo quadratico $y_{ij} = x_i x_j$, introduciamo due nuovi vincoli di attivazione $y_{ij} \leq x_i$ e $y_{ij} \leq x_j$, rispettivamente per la variabile decisionale i e j, che vincolano $y_{ij} = 1$ alla condizione $x_i = x_j = 1$.

Una formulazione lineare intera del MDP può essere la seguente:

$$z = \max \sum_{i,j \in N,\ i<j} d_{ij} y_{ij} \tag{2.1}$$

s.t.

$$\sum_{i \in N} x_i = m \tag{2.2}$$

$$y_{ij} \leq x_i \qquad \forall i, j \in N \tag{2.3}$$

$$y_{ij} \leq x_j \qquad \forall i, j \in N \tag{2.4}$$

$$x_i \in \{0, 1\} \qquad \forall i \in N \tag{2.5}$$

$$y_{ij} \in \{0, 1\} \qquad \forall i, j \in N \tag{2.6}$$

L'equazione 2.1 rappresenta la funzione obiettivo; l'equazione 2.2 modella il vincolo di cardinalità dell'insieme M; le equazioni 2.3 e 2.4 modellano il vincolo di attivazione di e_i ed e_j, rispettivamente. La formulazione presentata può essere rinforzata introducendo il seguente vincolo di taglio,

$$\sum_{j \in N} y_{ij} = (m-1) x_i \qquad \forall i \in N \tag{2.7}$$

il quale impone che ogni elemento $e_j \in M$ deve essere collegato ai rimanenti $(m-1)$ elementi della soluzione.

E' possibile risolvere il modello con proposto con uno strumento *general purpose*, ad esempio un risolutore lineare intero quali GLPK e CPLEX come illustrato nell'appendice A. L'impossibilità di risolvere istanze di grosse dimensioni in tempo ragionevole, rappresenta il limite di tale formulazione. Per questo motivo, in passato, ci sono stati tentativi di formulazioni alternative, ad esempio, ricordiamo il modello quadratico proposto in [5] e poi risolto con il codice per la programmazione quadratica binaria di P.M. Pardalos e G.P. Rodgers [12].

2.2 Algoritmi euristici

Weitz e Lakshminarayanan [17] hanno sviluppato cinque euristiche per determinare gruppi di studenti con le caratteristiche più diverse possibili, ad esempio nazionalità, età, livello di istruzione ecc. Gli algoritmi sono stati testati su istanze basate su dati reali e confrontate con le soluzioni un algoritmo esatto.

Glover [7] ha presentato delle euristiche di tipo distruttivo e costruttivo per risolvere delle istanze da lui create, aventi dimensione variabile (il massimo numero di elementi della popolazione è 30). Glover ha mostrato che le euristiche proposte trovano risultati con un gap del 2% rispetto a quelli trovati utilizzando metodi esatti, ma con un tempo di calcolo molto più limitato.

Ghosh [5] ha proposto un'euristica di tipo GRASP (Greedy Randomized Adaptive Search Procedure) in grado di ottenere ottimi risultati sulle piccole istanze del problema. Successivamente, Andrade ha proposto un nuovo tipo di GRASP, testandolo su un insieme di istanze proprie generate randomicamente con una popolazione massima di 500 individui. Questo nuovo algoritmo è stato in grado di trovare alcune soluzioni migliori rispetto a quelle trovate dall'algoritmo di Ghosh.

Infine, Silva [16] ha proposto un nuovo GRASP testandolo con una popolazione massima di 500 elementi. Sono stati così confrontati i nuovi algoritmi proposti con quelli già presenti in letteratura evidenziando nuovi sensibili miglioramenti. Tale confronto ha mostrato che il GRASP risulta il miglior algoritmo proposto per la soluzione di MDP.

2.3 Greedy Randomized Adaptive Search Procedure

L'algoritmo Greedy Randomized Adaptive Search Procedure (GRASP) appartiene alla classe delle metaeuristiche di ricerca locale. Esso è composto di due fasi distinte: la fase di **costruzione** di una soluzione e la fase di miglioramento della stessa attraverso uno schema di **ricerca locale**. Di conseguenza, la bontà della soluzione finale dipende in modo sinergico da entrambe le fasi. Le due fasi sono illustrate rispettivamente nel paragrafo 2.4 e 2.5 con riferimento alla letteratura sul problema MDP.

Una caratteristica interessante di tale approccio è la possibilità di scegliere indipendentemente le due fasi. Un esempio di pseudocodice della procedura per il GRASP puro è riportato nella tabella 2.1.

I parametri necessari all'esecuzione della procedura sono: la dimensione della popolazione n, il numero di elementi della soluzione m, la matrice delle diversità D e il numero massimo di iterazioni da eseguire. La matrice $D = [d_{ij}]$ contiene la diversità d_{ij} definita per ogni coppia di elementi $e_i, e_j)$ della popolazione N . Le righe 2 e 9 rappresentano il ciclo della procedura GRASP: la funzione $Build\_Random\_Greedy\_Solution$ (riga 3) costruisce una soluzione ammissibile

```
procedure pure_GRASP(n, m, d, I_max)
1.  z_Best ← −∞;
2.  for i = 1, ..., I_max do
3.  sol ← Build_Random_Greedy_Solution(n, m, d);
4.  sol ← Local_Search(Sol);
5.  if z(Sol) < z_best do
6.  z_best ← z(Sol);
7.  Best_Sol ← Sol;
8.  end if;
9.  end for;
10. return Best_Sol
```

Tabella 2.1: Procedura $pure\_GRASP$

dalla quale partire con la ricerca locale (riga 5). Se la soluzione trovata è migliore della $Best\_Sol$ si procede all'aggiornamento. La procedura termina restituendo $Best\_Sol$. Nelle prossime sezioni verranno illustrate più nel dettaglio la fase costruttiva e quella di ricerca locale.

2.4 Fase costruttiva

Per definizione, la fase costruttiva del GRASP è un algoritmo di tipo Greedy. Esso parte da una soluzione vuota $M = \emptyset$, ad ogni iterazione, gli elementi $e_i \in N \setminus M$ vengono valutati attraverso una funzione $g : N \setminus M \rightarrow \Re^+$ che considera quale sarebbe il guadagno in termini di funzione obiettivo, se l'elemento $i \in N \setminus M$ venisse incluso nella soluzione M. Gli algoritmi che presentiamo nei paragrafi seguenti fanno uso di una **Restricted Candidate List** (RCL). Il processo termina dopo m passi, quando si è raggiunta una soluzione ammissibile.

Restricted Candidate List

Gli elementi dell'insieme N vengono ordinati per valori decrescenti della funzione g. La costruzione della soluzione M avviene estraendo dalla Restricted Candidate List (RCL) di volta in volta l'elemento che ha il massimo valore di g. La fase di costruzione termina dopo m passi. La dimensione della RCL è limitata da un parametro α: troveremo nella RCL solamente quegli elementi $i \in N \setminus M$ ai quali è associato un valore della funzione g compreso nell'intervallo chiuso $[(1 - \alpha)g_{max},\ g_{max}]$.

Prais e Ribero ha proposto una nuova procedura chiamata Reactive GRASP [13], nella quale il parametro α, utilizzato nella fase costruttiva, subisce una variazione ad ogni iterazione. Ecco brevemente il funzionamento. Nella prima iterazione si sceglie casualmente un α da un insieme $A = \{\alpha_1, ..., \alpha_m\}$. Ad ogni elemento α_i è associata una probabilità p_i. Inizialmente la densità è uniforme, ossia

$p_i = \frac{1}{m},\ i = 1, ..., m$. Periodicamente la distribuzione di probabilità viene aggiornata utilizzando informazioni raccolte durante la fase di costruzione della soluzione. L'obiettivo è quello di associare maggiore probabilità a valori di α che portano a migliori soluzioni e minore probabilità a valori di α che conducono a soluzioni peggiori.

2.4.1 K Larger Distances

L'algoritmo K Larger Distances (**KLD**) [16] costruisce una soluzione iniziale selezionando casualmente, ad ogni iterazione, un elemento da una RCL di lunghezza K. Ad ogni elemento $i \in N$ è associata come funzione g la somma delle distanze da tutti gli altri elementi $j \in N \setminus \{i\}$. Gli elementi i vengono ordinati per di g decrescenti e i primi K costituiscono la RCL. Nel primo blocco di iterazioni pari a $B_1 = 0.4m$ vengono valutati quattro differenti valori di $K \in \{K_1, K_2, K_3, K_4\}$; la valutazione è fatta dividendo il blocco in uguali intervalli $c_i,\ i = 1, ..., 4$. Si utilizza il valore K_i per tutte le iterazioni che appartengono all'intervallo x_j,$i = j$. I valori di K_i sono mostrati nella tabella 2.2. Dopo l'ultima iterazione del blocco B_1 si valuta la qualità della migliore soluzione ottenuta per ogni K_i,e il valore medio della diversità, definito

$$\mu_i(z) = \Sigma_{1 \leq q \leq 0.1m} z(sol_{iq})$$

per le soluzioni $sol_{iq},\ i = 1, ..., 4;\ q = 1, ..., 0.1m$ ottenute utilizzando ogni K_i.

i	c_i	K
1	$[1, ..., 0.1m]$	$m + \mu - 0.2\mu$
2	$(0.1m, ..., 0.2m]$	$m + \mu - 0.1\mu$
3	$(0.2m, ..., 0.3m]$	$m + \mu + 0.1\mu$
4	$(0.3m, ..., 0.4m]$	$m + \mu + 0.2\mu$
$\mu = \frac{(n-m)}{2}$		

Tabella 2.2: Valori di K per il blocco B_1

I valori di K_i sono memorizzati in una lista L_k ordinata rispetto ai valori di $\mu_i(z)$. L'algoritmo prosegue considerando il secondo blocco di iterazioni $B_2 = 0.6m$. Tale blocco viene diviso in quattro intervalli y_i ognuno dei quali è formato da un numero diverso di iterazioni e valori di K_i come mostrato in tabella 2.3. In questo modo, i valori K_i che forniscono soluzioni migliori vengono utilizzati in un maggior numero di iterazioni.

Ad ogni iterazione del GRASP che contiene KLD si applica una tecnica di filtro: si costruiscono 400 soluzioni e si selezionana la migliore sulla quale proseguire con la ricerca locale. In tabella 2.4 è riportato lo pseudocodice della procedura KLD [16].

i	y_i	K
1	$[0.4m, .., 0.64m]$	L_{k1}
2	$(0.64m, ..., 0.82m]$	L_{k2}
3	$(0.82m, ..., 0.94m]$	L_{k3}
4	$(0.94m, ..., m]$	L_{k4}

Tabella 2.3: Valori di K per il blocco B_2

procedure $constr\_KLD(it\_GRASP, m, numsol, n, I_{max})$
1. $best\_cost\_sol \leftarrow 0$;
2. $K \leftarrow det\_K(it\_GRASP, m, LK, i)$;
3. $num\_sol[i] \leftarrow num\_sol[i] + 1$;
4. $RCL \leftarrow Build\_RCL(K)$;
5. **for** $j = 1, ..., max\_sol\_filter$ **do**
6. $sol \leftarrow \{\}$;
7. **for** $k = 1, ..., I_{max}$ **do**
8. Selezione di e^* dalla RCL;
9. $sol \leftarrow sol \cup \{e^*\}$;
10. $RCL \leftarrow RCL - \{e^*\}$;
11. **end for**;
12. **if** $(z(sol) > best\_cost\_sol)$ **then do**
13. $sol\_constr \leftarrow sol$;
14. $best\_cost\_sol \leftarrow z(sol)$;
15. **end if**
16. **end for**;
17. $sol\_eval[i, num\_sol[i]] \leftarrow z(sol\_constr)$;
18. **if** $((it\_GRASP == 0.4m)$ **then do**
19. $L_k \leftarrow Build\_L_k(sol\_eval)$;
20. **end if**;
21. **return** sol_constr;

Tabella 2.4: Procedura $constr\_KLD$

Nella riga 1 viene inizializzato il valore della migliore soluzione trovata nell'esecuzione delle iterazioni di max_sol_filter. Alla riga 2, il valore di K serve per costruire la Restricted Candidate List (RCL), la procedura $det\_K(it\_GRASP, m, LK, i)$ definisce il valore di K per l'implementazione del GRASP reattivo. La riga 3 viene aggiornato il numero di soluzioni trovate per un K specifico. Alla riga 4 viene costruita la RCL. Dalla linea 5 alla linea 16, viene eseguita max_sol_filter volte la procedura di costruzione e solo la migliore è restituita per poter essere usata come soluzione iniziale dalla procedura di ricerca locale. Dalla linea 7 alla 11, viene costruita una soluzione selezionando un elemento casualmente dalla RCL. Alle linee 12-15, la migliore soluzione trovata dalla procedura di costruzione viene aggiornata e alla linea 17 viene memorizzato il costo della soluzione trovata utilizzando il K selezionato. Al termine del blocco di iterazioni, i valori K_i vengono valutati e inseriti in ordine decrescente nella lista L_k alla riga 19.

Versione adattiva

Sia M_c una soluzione parziale contenente c elementi $0 \leq c < m$ e $i \in N \setminus M_c$ un elemento candidato ad essere inserito nella prossima soluzione parziale M_{c+1}. Per ogni i si selezionano i $(K-1-c)$ elementi $j \in N \setminus (M_c \cup \{i\})$ che hanno d_{ij} maggiore e si calcola la loro somma. La RCL viene costruita sulla base di una funzione greedy $g(i) = \sum_{j \in M_c} d_{ij} + s_i$, dove il primo termine corrisponde alla somma delle distanze del candidato i con da gli elementi $j \in M_c$; il secondo termine rappresenta la somma delle distanze dell'elemento i dagli $(K-1-c)$ elementi non appartenenti alla soluzione parziale M_c che presentano la maggiore distanza da i. La RCL viene quindi formata prendendo i K elementi con $g(i)$ massima. La procedura è adattiva perchè i valori $g(i)$ cambiano ad ogni estrazione, dato che l'insieme M_c cresce mentre l'insieme $N \setminus (M_c \cup \{i\})$ decresce. Il GRASP reattivo e il filtro nella fase costruttiva sono implementati in modo analogo al KLD. Questo algoritmo è stato proposto come fase costruttiva in una versione GRASP in [16].

2.4.2 Most Distant Insertion

L'euristica Most Distance Insertion (**MDI**) è stata proposta in [16].

Sia M_c una soluzione parziale formata da c, $(1 \leq c \leq m)$ elementi; per $c = 1$, la soluzione M_1 è ottenuta selezionando casualmente un elemento e_i dall'insieme N di partenza; per $c = 2$, $M_2 = M_1 \cup \{e_i\}$, dove e_i è l'elemento che presenta la maggiore distanza d_{ij}, $i \in M_1$, $j \in N \setminus M_1$. Per ottenere M_c $(c \geq 3)$ da M_{c-1}, l'elemento da inserire in soluzione è casualmente scelto dalla RCL. In questo algoritmo, la RCL viene costruita sulla base della funzione

$$dsum(j) = \sum_{1 \leq y \leq c-2} \sum_{y+1 \leq w \leq c-1} d_{yw} + \sum_{1 \leq v \leq c-1} d_{vj} \tag{2.8}$$

Il primo termine corrisponde alla somma delle distanze tra tutti gli elementi $i \in M_{c-1}$; il secondo termine rappresenta la somma tra tutti gli elementi $i \in M_{c-1}$ con un candidato j che non appartiene alla soluzione parziale M_{c-1}.

MDI prevede la creazione di una lista iniziale dei candidati (ICL), formata dagli elementi $j \in N \setminus M_{c-1}$, disposti in ordine decrescente rispetto ai valori della funzione $dsum(j)$. I primi $\alpha \cdot n$ elementi della ICL sono estratti dalla RCL.

L'implementazione è molto simile a quella in KLD. Il primo blocco di dimensione $B_1 = 0.4m$ dviene diviso in quattro intervalli di uguale ampiezza, vengono inoltre valutati quattro valori $\alpha \in \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$. La tabella che segue mostra il valore di α impiegato in ogni intervallo. I valori $\alpha_i, i = 1, ..., 4$ sono ottenuti calcolando la diversità media così ottenuta:

$$\mu_i(z) = \sum_{1 \leq q \leq 0.1m} z(sol_{iq}) \tag{2.9}$$

per ogni soluzione $sol_{iq},\ i = 1, ..., 4;\ q = 1, ..., 0.1m$. Gli α_i vengono memorizzati in una lista L_α ordinata per valori decrescenti di $\mu_i(z)$

i	c_i	α
1	$[1, ..., 0.1m]$	0.03
2	$(0.1m, ..., 0.2m]$	0.05
3	$(0.2m, ..., 0.3m]$	0.07
4	$(0.3m, ..., 0.4m]$	0.1

Tabella 2.5: Valori di α per il blocco B_2

Il secondo blocco $B_2 = 0.6m$ iterazioni, analogamente al blocco precedente, viene diviso in quattro intervalli y_i ognuno dei quali è formato da un numero variabile di iterazioni. Anche in questo caso si associa un valore di α ad ogni elemento come mostrato dalla tabella 2.6 .

i	y_i	α
1	$[0.4m, ..., 0.64m]$	$l\alpha_1$
2	$(0.64m, ..., 0.82m]$	$l\alpha_2$
3	$(0.82m, ..., 0.94m]$	$l\alpha_3$
4	$(0.94m, ..., m]$	$l\alpha_4$

Tabella 2.6: Valori di α per il blocco B_1

Riportiamo lo pseudocodice per MDI nella tabella 2.7. Nella prima riga, viene inizializzato il valore della migliore soluzione. Il valore di α da utilizzare per costruire la RCL è calcolato dalla procedura det_α alla linea 2. Questa procedura seleziona α basandosi sulla tecnica "reactive GRASP" discusso in precedenza. Alla riga 3, vengono aggiornate le soluzioni trovare relative ad uno specifico valore di α e, alla linea 4 avviene l'inizializzazione dell'insieme che dovrà contenere gli elementi candidati ad entrare a far parte della soluzione. Dalla riga 5 alla 24, viene eseguita max_sol_filter volte la procedura di costruzione, che porterà a selezionare la migliore soluzione sulla quale partire con la fase di ricerca locale. Dalla riga 7 alla 19,

procedure $constr\_MDI(it\_GRASP, m, numsol, n, I_{max})$
1. $best\_cost\_sol \leftarrow 0$;
2. $\alpha \leftarrow det\_\alpha(it\_GRASP, m, L_\alpha, i)$;
3. $num\_sol[i] \leftarrow num\_sol[i] + 1$;
4. $N\_RCL \leftarrow N$;
5. **for** $j = 1, ..., max\_sol\_filter$ **do**
6. $sol \leftarrow \{\}$;
7. Selezione casuale di un elemento m_i da N; $sol \leftarrow sol \cup \{m_1\}$;
8. **for** $j \in N \setminus M_1$ **do**
9. Calcolo di d_{m1j}
10. $m_2 \leftarrow l, |d_{m_1 l} = max(d_{m1j}), j \in N \setminus M_1$;
11. $sol \leftarrow sol \cup \{m_2\}$;
12. **end di tutti**;
13. $N\_RCL \leftarrow N - M_2$;
14. **for** $k = 3, ..., I_{max}$ **do**
15. $RCL \leftarrow Build\_RCL\_\alpha(N\_RCL, \alpha)$;
16. Selezione casuale di un elemento e^* dalla RCL;
17. $sol \leftarrow sol \cup \{e^*\}$;
18. $N\_RCL \leftarrow N\_RCL - \{e^*\}$;
19. **end for**;
20. **if** $(x(sol) > best\_cost\_sol)$ **then do**
21. $sol\_constr \leftarrow sol$;
22. $best\_cost\_sol \leftarrow z(sol)$;
23. **end if**
24. **end for**;
25. $sol\_eval[i, num\_sol[i]] \leftarrow z(sol\_constr)$;
26. **if** $(it\_GRASP == 0.4m)$ **then do**
27. $L_\alpha \leftarrow Build\_L\alpha(sol\_eval)$;
28. **end if**;
29. **return** sol_constr;

Tabella 2.7: Procedura $constr\_MDI$

vengono individuati gli elementi da inserire in soluzione. Ad ogni iterazione, viene creata una RCL, selezionato un elemento casualmente e aggiornata la lista dei candidati (linea 18). Si procede quindi con l'aggiornamento della migliore soluzione trovata. Il valore relativo all'α selezionato viene memorizzato (linea 25). Terminato il numero di iterazioni previste per il blocco B_1, si procede con la valutazione dei valori α_i e con il loro posizionamento nella lista L_α (riga 27).

2.4.3 Euristica di Ghosh

Questa euristica prende il nome dall'autore che l'ha proposta [5]. Sia M_{k-1} una soluzione parziale con $k-1$ elementi, con $(1 \leqslant k \leqslant m)$. Per ogni $e_i \in N \setminus M_{k-1}$, sia $\Delta z(i)$ il contributo marginale che e_i fornisce a $z(M_m)$. Siccome la soluzione finale M_m non è ancora determinata, tale contributo è incognito ma è possibile valutare una stima per difetto $\Delta z_L(i)$ e una per eccesso $\Delta z_U(i)$.

Sia $d_i^r(Q_{ir})$ l'r-esima distanza maggiore nell'insieme $\{d_{ij} : j \in Q_{ik}\}$ dove Q_{ik} è dato da $Q_{ik} = N \setminus M_{k-1} \setminus \{i\}$. Poniamo allora,

$$\Delta z_L(i) = \sum_{j \in M_{k-1}} d_{ij} + \sum_{n-m+1 \leq r \leq n-k} d_i^r(Q_{ik}) \tag{2.10}$$

$$\Delta z_U(i) = \sum_{j \in M_{k-1}} d_{ij} + \sum_{1 \leq r \leq n-k} d_i^r(Q_{ik}) \tag{2.11}$$

Il passo successivo prevede l'estrazione di un numero casuale u da una distribuzione $U(0,1)$ uniforme tra 0 e 1. Tale numero ci serve per calcolare la quantità

$$\Delta z'(i) = (1-u)\Delta z_L(i) + u\Delta z_U(i) \tag{2.12}$$

Il valore di $\Delta z'(i)$ è il criterio per scegliere scegliere il nuovo elemento e_i^* da aggiungere a M, ovvero quello tale che $\Delta z'(i^*) = max_{i \in N \setminus M_{k-1}} \Delta z'(i)$. Questo processo iterativo termina quando la cardinalità dell'insieme M è pari a m.

2.4.4 Euristica di Andrade

L'euristica di Andrade è stata proposta in [1] e differisce da quelle di Ghosh [5] per i seguenti punti.

Inizialmente, vengono calcolati i valori $SD(i) = \sum_{j \in N} d_{ij},\ i = 1, .., n$ e $MD(i) = \frac{1}{n} \sum_{j \in N} d_{ij},\ i = 1, .., n$, dove $SD(i)$ rappresenta la somma delle distanze tra e_i e gli altri elementi $e_j \in N$ mentre $MD(i)$ rappresenta la media di tali distanze. Il primo elemento di M_1 è selezionato casualmente fra gli m elementi che hanno il valore

procedure $Buil\_list(N, M_{k-1})$

1. **for all** $i \in N - M_{k-1}$ **do**
2. $SDS(i) \leftarrow \frac{\sum_{j \in M_{k-1}} d_{ij}}{k-1}$;
3. **if** $((SDS(i) > SD(i))$ and $(k > \frac{m}{2}))$ **then do**
4. $\Delta z(i) \leftarrow SDS(i)$;
5. **else**
6. $\Delta z(i) \leftarrow \frac{SDS(i)+MD(i)}{2}$;
7. **end if**;
8. **end for**;
9. Sort $i \in N - M_{k-1}$ by Δz;
10. **if** $m > \frac{n}{2}$ **then do**
11. $lim\_list \leftarrow n - m$;
12. **else**;
13. $lim\_list \leftarrow m$;
14. **end if**;
15. $D \leftarrow 0$;
16. **for** $(i = 0, ..., lim\_list - 1)$ **do**
17. $D \leftarrow D + (\Delta z(i) - \Delta z(i+1))$;
18. **end for**;
19. $D \leftarrow \frac{D}{lim\_list}$;
20. $Q \leftarrow 0$;
21. $Restricted\_List \leftarrow \{\varnothing\}$;
22. **for** $(i = 0, ..., lim\_list - 1)$ **do**
23. **if** $((\Delta z(i) - \Delta z(i+1)) < D)$ **then do**
24. $Restricted\_List \cup \{i\};\ Q \leftarrow Q + 1$;
25. **else**
26. **break**;
27. **end for**;
28. **return** $Restricted\_List$;

Tabella 2.8: Procedura $Build\_list$

maggiore di MD. Ad ogni iterazione, si crea una RCL e si seleziona casualmente un elemento, come mostrato nel pseudocodice riportato nella tabella 2.8.

Alla riga 2, viene calcolata la distanza media $SDS(i)$ di ogni elemento $i \in N \backslash M_{k-1}$ dagli elementi che compongono già la soluzione. Dalla riga 3 alla 7, si calcolano i $\Delta z(i)$. Nei primi $\frac{m}{2}$ passi Δz è la media fra MD e SDS, in modo da favorire gli elementi che hanno una forte distanza sia dall'intera popolazione, sia dalla soluzione parziale M_{k-1} . Nei successivi $\frac{m}{2}$ passi, si considera solo la distanza dalla soluzione parziale. Alla riga 9, vengono ordinati per valore decrescente di Δz quegli elementi che non appartengono alla soluzione parziale; dalla riga 10 alla 14, viene deciso un valore iniziale lim_list per la lunghezza della Initial Restricted List. Nelle righe successive (15-19), si calcola la distanza media D fra i Δz degli elementi della lista. Dalla linea 20 alla 27, viene creata la lista ristretta finale, selezionando elementi dalla lista iniziale che presentano una reciproca differenza Δz minore di D. Questo ultimo accorgimento, assicura la presenza in soluzione di quegli elementi che presentano Δz vicina ad un elemento già in soluzione. D'altro canto, si cerca di impedire la selezione degli elementi che hanno una Δz reciproca troppo alta.

2.5 Fase di ricerca locale

La fase che segue la costruzione della soluzione ammissibile è di solito una ricerca locale con l'obiettivo di migliorare la soluzione, costruita nella fase precedente. Per ogni iterazione di GRASP, l'algoritmo di costruzione viene eseguito n volte, generando ogni volta una soluzione. Solo la migliore viene presa come partenza per la ricerca locale. Nel seguito descriveremo le fasi di ricerca locale per gli algoritmi descritti della sezione precedente.

2.5.1 Scambio di un elemento

Per ogni $e_i \in M$ e $e_j \in N \setminus M$, il miglioramento dovuto allo scambio dell'elemento e_i con l'elemento e_j, è ottenuto calcolando la quantità

$$\Delta z(i,j) = \sum_{u \in M \setminus \{i\}} (d_{ju} - d_{iu}) \tag{2.13}$$

Se per ogni e_i, e_j si verifica $\Delta z(i,j) < 0$, ovvero, non si riescono più a trovare scambi vantaggiosi in grado di migliorare la funzione obiettivo, la ricerca locale termina. Altrimenti si determina la coppia (e_i^*, e_j^*) che corrisponde al miglioramento massimo della funzione obiettivo, e si ottiene una nuova soluzione scambiando gli elementi individuati, dando luogo ad una nuova soluzione M , e quindi un nuovo intorno sul quale proseguire la ricerca locale.

Questo tipo di intorno è stato adottato nella ricerca locale degli algoritmi in [5, 1].

2.5.2 Scambio di coppie di elementi

E' possibile definire un intorno più vasto, definito dall'insieme di tutte le soluzioni ottenute rimpiazzando una coppia di elementi della soluzione, con una coppia non appartenente. Per ogni $(e_i, e_j) \in M$ e $(e_v, e_w) \in N \setminus M$, il miglioramaneto dovuto allo scambio degli elementi (e_i, e_j) con (e_v, e_w) è

$$\Delta z((i,j),(v,w)) = \sum_{u \in M \setminus \{i,j\}} (d_{vu} + d_{wu} - (d_{iu} + d_{ju})) \tag{2.14}$$

Se per tutte le coppie (e_i, e_j), (e_v, e_w), si verifica $\Delta z((i,j),(v,w)) < 0$, siccome non è possibile trovare uno scambio vantaggioso, la ricerca locale termina. Altrimenti, le coppie (e_i^*, e_j^*), (e_v^*, e_w^*) che forniscono la massima $\Delta z((i,j),(v,w))$ vengono scambiate creando una nuova soluzione e, quindi, un nuovo intorno da esplorare con un'ulteriore iterazione di ricerca locale.

Questo tipo di intorno è stato adottato nell'algoritmo SOM (Silva, Ochi e Martins) in [16].

La composizione delle fasi costruttive e di ricerca locale viste in precedenza determinano gli algoritmi GRASP proposti sia da Andrade [1] che da Silva [16]: analizzando i risultati riportati, si osserva che il GRASP proposto in [16] è attualmente il migliore algoritmo per la soluzione di MDP in termini di qualità della soluzione.

2.5.3 GRASP con Path-Relinking

Nelle versioni base il GRASP è un algoritmo che non prevede alcuna funzione di memoria. Infatti l'unico scambio di informazione tra due soccessive iterazioni è la soluzione corrente. Le iterazioni risultano tra di loro indipendenti. Con la tecnica di Path-Relinking introduciamo un concetto di memoria nello schema del GRASP con l'obiettivo di pilotare la ricerca verso migliori soluzioni [2].

La strategia della tecnica di Path-Telinking consiste nell'intensificare la ricerca nelle aree più promettenti attraverso la combinazione di soluzioni. Le soluzioni promettenti trovate nelle precedenti iterazionivengono memorizzate in un *insieme di élite* E. Quando l'insieme E ha raggiunto la dimensione massima prevista, per ogni soluzione s trovata in un'iterazione di GRASP, si seleziona una delle soluzioni $s_e \in E$ e si applica la tecnica di Path-Relinking sulla coppia (s, s_e).

Ci sono due modi per scegliere l'elemento e : mediante estrazione casuale o estraendo quello di maggior valore. La realizzazione del Path-Relinking avviene partendo da una soluzione iniziale s_i e incorporando gradualmente attributi da una soluzione guida s_g fino a che s_i diviene uguale a s_g. La letteratura presenta molteplici modi per selezionare s_i e s_g [14]. La strategia adottata da Andrade [2] è la seguente:

procedure $GRASP\_PR(n, m, D, I_{max}, EliteSel, RelinkType)$
1. $z\_best \leftarrow \infty$;
2. $E \leftarrow \{\}$;
3. $Filter \leftarrow \{\}$;
4. **for** $i = 1, ..., I_{max}$ **do**
5. $Sol \leftarrow Build\_Random\_Greedy\_Solution(n, m, D)$;
6. $Search\_Sol\_Filter(Filter, Sol, SolNotFound)$;
7. **if** $SolNotFound$ **then do**
8. $Sol \leftarrow Local\_Search(Sol)$;
9. **end if**;
10. **if** E **is** $full$ **then do**
11. $SolElite \leftarrow Select\_Sol\_Elite(E, EliteSel)$;
12. $Sol \leftarrow PathRelinking(Sol, SolElite, RelinkType)$;
13. **end if**;
14. $Insert\_Elite(E, Sol)$;
15. **if** $z(Sol) > z\_best$ **do**
16. $z\_best \leftarrow z(Sol)$;
17. $Best\_Sol \leftarrow Sol$;
18. **end if**;
19. **end for**;
20. **return** $Best\_Sol$;

Tabella 2.9: Procedura $GRASP\_PR$

- *Backward relinking*: la migliore soluzione è assegnata a s_i mentre la peggiore a s_g
- *Forward relinking*: la peggiore soluzione è assegnata a s_i mentre la migliore a s_g
- *Mixed relinking*: esplorazione simultanea di entrambe le traiettorie; ad ogni iterazione di Path-relinking, s_i e s_g si invertono i ruoli.
- Un algoritmo di tipo GRASP con Path-Relinking è stato proposto da Andrade in [2] .

Lo pseudocodice è riportato in tabella 2.9.

I parametri della procedura sono: la dimensione della popolazione n, il numero m di elementi della soluzione M, la matrice delle distanze D, il numero massimo di iterazioni I_{max}, il parametro che permette di costruire l'insieme di élite E e la politica di relinking da adottare ($RelinkType$). In successione vengono inizializzati gli insiemi filtro e di elite. Il ciclo fra le righe 5 e 20 esegue le iterazioni del GRASP, per ognuna si costruisce una soluzione e alla riga 6, si ricerca nell'insieme filtro se la soluzione è già stata trovata; in caso negativo, viene eseguita una ricerca locale (riga 8). Se l'insieme di elite E è pieno, parte la procedura di Path-Relinking secondo la

procedure $Path\_Relinking(s, e, RelinkType)$
1. $BetterSolRC \leftarrow \{\}$;
2. $Find\_Better\_Worst(s, e, BS, WS)$;
3. $BetterSolRC \leftarrow BS$;
4. **if** $RelinkType$ **uguale** a $T1$ or $T3$ **then do**
5. $s_i \leftarrow BS$;
6. $s_g \leftarrow WS$;
7. **else do**
8. $s_i \leftarrow WS$;
9. $s_g \leftarrow BS$;
10. **end if**;
11. **while** $s_i \neq s_g$ **do**
12. $DifEl \leftarrow Obtain\_Different\_Elements(s_i, s_g)$;
13. $InitSol \leftarrow Obtain\_Int\_Sol(s_i, DifEl)$;
14. **if** $z(InitSol) > z(BetterSolRC)$ **then do**
15. $BetterSolRC \leftarrow IntSol$;
16. **endi if**;
17. **if** ($RelinkType$ = $T1$ **or** $RelinkType = T2$) **then do**
18. $Initial \leftarrow IntSol$;
19. **else do**
20. $Initial \leftarrow Guide$;
21. $Guide \leftarrow IntSol$;
22. **end if**;
23. **end while**;
24. **return** $BetterSolRC$;

Tabella 2.10: Procedura $Path\_Relinking$

strategia definita dal parametro $RelinkType$. Alla riga 14, sia le soluzioni generate dal GRASP che quelle generate dal path-relinking vengono valutate per stabilire se inserirle nell'insieme di élite E. Nel caso in cui E sia pieno, la soluzione candidata viene effettivamente inserita solo se è migliore della peggiore soluzione presente in E. Le istruzioni successive provvedono all'aggiornamento e restituzione della migliore soluzione trovata. La tabella 2.10 riporta lo pseudocodice relativo alla procedura $Path\_Relinking$.

Alla riga 2, la soluzione s generata dal GRASP e la soluzione s_e di élite vengono confrontate per determinare quale delle due ha maggiore costo (BS). Dalla riga 4 alla riga 10, vengono inizializzate le soluzioni s_i e s_g. Nell'ipotesi che si sia scelta una strategia di tipo $backward\ relinking$ $(T1)$ o di tipo $mixed\ relinking$ $(T3)$ viene scelta come soluzione iniziale la migliore fra s ed s_e, la peggiore. Dalla riga 11 alla riga 23,si esegue il path-relinking fino a che la soluzione iniziale non raggiunge quella di guida (s_g): alla riga 12, vengono trovati quegli elementi che sono nella soluzione guida ma non in quella iniziale.

Le soluzioni intermedie si ottengono rimpiazzando un elemento appartenente alla

soluzione iniziale, con uno che non appartiene alla soluzione guida e ha tutti gli altri elementi appartenenti alla soluzione guida e non alla soluzione iniziale. Come risultato del ciclo di path-relinking, si prende la migliore soluzione tra quelle esplorate (riga 15). Dalla linea 17 alla 22, rispettando la strategia adottata vengono selezionate le nuove soluzioni, s_i e s_g che verranno usate nella seconda iterazione.

2.6 Un primo test computazionale

Si è svolta una campagna computazionale sulle istanze più piccole proposte da Andrade [2], al fine di mostrare l'impossibilità pratica di risolvere all'ottimo con risolutori general purpose le istanze di dimensioni richieste dalle applicazioni reali. Infatti all'aumentare della dimensione del problema, diviene molto difficile trovare la soluzione ottima in un tempo di calcolo accettabile.

Per calcolare le soluzioni ottime sono stati usati due risolutori general purpose: **GLPK** (Versione 4.8) e **CPLEX** (Versione 8.1). Il pacchetto GLPK (GNU Linear Programming Kit) permette di risolvere problemi di programmazione lineare (LP), programmazione lineare intera mista (MIP) e altri problemi correlati. CPLEX è conosciuto come il migliore strumento commerciale, attualmente sul mercato per risolvere una vasta gamma di problemi di ottimizzazione, fra i quali LP e MIP. Grazie al supporto di GLPK con il linguaggio di programmazione matematica AMPL, è stato possibile scrivere il modello del Maximum Diversity Problem per risolverlo sia con GLPK che con CPLEX. Il modello di programmazione scritto in linguaggio AMPL è riportato nell'appendice A e fa riferimento al modello PLI presentato nella sezione 2.1.1.

I test sono stati condotti su un sottoinsieme di istanze [2] di piccole dimensioni ($n = 50$) utilizzando un PC AMD Xp2000+ con 512Mb di RAM in ambiente Linux Slackware 10.0. Per i dettagli relativi alle classi di istanze si rimanda alla sezione 4.1. I parametri utilizzati sono quelli standard per entrambi i risolutori.

I risultati della sperimentazione sono riportati nella tabella 2.11: la colonna "*Istanza*" indica il tipo di istanza risolta dove la prima lettera si riferisce alla classe (a,b,c,d), il primo gruppo di cifre dice quale è la dimensione della popolazione N e il secondo la dimensione della soluzione M; z^* è il valore della soluzione ottima, trovato sia da GLPK[1] che da CPLEX. Per ogni pacchetto sono state inoltre riportate le seguenti informazioni: *CPU time* indica il tempo necessario per determinare z^*; la colonna L_b (*Lower Bound*) rappresenta la soluzione euristica trovata dal risolutore al nodo radice, per arrotondamento della soluzione frazionaria, del rilassamento continuo della formulazione lineare intera; analogamente la colonna U_b (*Upper Bound*) è stato ottenuto dal risolutore con il rilassamento continuo dello stesso modello; il gap al nodo radice è stato calcolato come $gap = \frac{U_b - L_b}{L_b}$.

Per quanto riguarda CPLEX si è impostato il tempo limite massimo di calcolo a 6 ore. Hanno terminato tutte le istanze tranne 2: **b050m20** e **c050m20**. In questi

[1] Le istanze b050m20 e c050m20 non sono state risolte perchè troppo onerose per i tempi di calcolo. Come conferma il tentativo di risolvere l'istanza d050m20.

		GLPK			
Istanza	z^*	***CPU time***	L_b	U_b	***gap %***
a050m10	492	14m 49s	482	535,98	11,2
b050m10	334976	1h 21m 42s	269103	402039,88	49,4
c050m10	316409	1h 11m 42s	299167	305928,17	22,6
d050m10	381379	1h 43m 31s	351366	423396,03	20,5
a050m20	1932	1h 22m 2s	1880,70	2049,96	9,0
b050m20	-	-	-	-	-
c050m20	-	-	-	-	-
d050m20	1501904	*16 h*$^{(a)}$	1442649	1663374,30	15,3
		CPLEX			
Istanza	z^*	***CPU time***	L_b	U_b	***gap %***
a050m10	492	2m 53s	373,63	536,20	43,51
b050m10	334976	15m 46s	212325	402084,00	89,37
c050m10	316409	1m 19s	114400	366735,24	220,57
d050m10	381379	29m 46s	234291	423491,93	80,75
a050m20	1932	46m 9s	1797,55	2049,52	14,02
b050m20	*1148174*$^{(b)}$	**6h**	972605	1497846,78	54,00
c050m20	1094343	1m 33s	1009852	1180187,20	16,87
d050m20	*1478180*$^{(c)}$	**6h**	1381684	1662840,37	20,35

Tabella 2.11: Risoluzione all'ottimo di piccole istanze con CPLEX e GLPK

$^{(a)}$ *Nel caso dell'istanza d050m20, il calcolo è stato interrotto dopo 979 minuti (più di 16 ore), con un gap residuo del 3,7%*

$^{(b)}$ *E' il valore della funzione obiettivo restituita da CPLEX dopo 6 ore di calcolo, il gap residuo è 11,74%*

$^{(c)}$ *E' il valore della funzione obiettivo restituita da CPLEX dopo 6 ore di calcolo, il gap residuo è 4,08%*

due casi si è riportato in corrispondenza della colonna z^* il L_b trovato al momento dell'interruzione. L'istanza ***b050m20*** è terminata con un gap residuo del 11,74% mentre l'istanza ***d050m20*** è terminata con un gap del 4,08%.

I risultati mostrano che CPLEX è molto più veloce di GLPK. Questo divario è dovuto dalla differenza nei gap al nodo radice, nettamente più basso per GLPK.

I tempi di calcolo indicati nella tabella non sono accettabili in molte applicazioni reali dove la popolazione può raggiungere dimensioni molto maggiori di 50 elementi. Consapevoli di questa forte limitazione nella risoluzione all'ottimo del problema, nel seguito proponiamo un algoritmo di tipo Tabu Search (capitolo 3) in grado di fornire una buona soluzione con un tempo di calcolo ragionevole. Come verrà mostrato in dettaglio durante la discussione dei risultati (capitolo 4), tutte le versioni di Tabu Search proposte raggiungono il valore ottimo trovato da **CLEX** e **GLPK** quando questo è noto impiegando pochi decimi di secondo contro i minuti di GLPK o CPLEX.

Capitolo 3

Tabu Search

L'algoritmo di Tabu Search è una metodologia di calcolo che serve per superare i limiti tipici di una ricerca locale. Infatti, un algoritmo di Ricerca Locale prosegue la sua ricerca sino a quando trova soluzioni miglioranti; una volta individuato un ottimo locale, la ricerca si ferma a prescindere dalla qualità della soluzione trovata. Per garantire una migliore qualità della soluzione trovata occorrebbe esplorare il maggior numero di ottimi locali.

Nei metodi di tipo GRASP, presentati nel precedente capitolo, l'esplorazione avviene diversificando le zone di partenza per la ricerca locale; ad ogni iterazione di GRASP un algoritmo Greedy costruisce una soluzione ammissibile, diversa dalle precedenti, dalla quale partire con la ricerca locale. Al crescere del numero di iterazioni, aumentano le aree esplorate consentendo di raggiungere un maggior numero di ottimi locali.

Invece, negli algoritmi di tipo Tabu Search si rilascia il vincolo di trovare ad ogni iterazione una soluzione migliorante: ad ogni iterazione cerchiamo la migliore soluzione dell'intorno anche se questa è peggiore della precedente così facendo la ricerca in linea di principio, non termina mai e lo spazio delle soluzioni viene esplorato più largamente. Il rischio di un approccio di questo tipo è quello di incorrere in una sequenza di soluzioni che determina un ciclo. Per evitare di incorrere in cicli introduciamo il concetto di *tabu list*: mantenendo una lista delle ultime soluzioni trovate possiamo evitare un ciclo definendo *tabu* una soluzione, dell'intorno corrente, che appartiene a questa lista; quindi le soluzioni tabu vengono scartate per evitate cicli. Una soluzione tabu non viene scartata (**criterio di aspirazione**) se il valore della soluzione è migliore dell'ottimo trovato. Uno schema di algoritmo è riportato in tabella 3.1.

Per maggiori dettagli sulla metodologia di Tabu Search si veda il libro di Glover e Laguna [8].

Nel resto del capitolo presenteremo le componenti dell'algoritmo di base dell'algoritmo di Tabu Search (sezione 3.1). Successivamente mostreremo una sua estensione introducendo il concetto di *Short Term Memory* (sezione 3.2). In conclusione discuteremo un'ulteriore estensione dell'algoritmo introducendo il concetto di *Long*

TabuSearchBase( )
k:=1
s:=GeneraSoluzioneIniziale( )
$s^* := s$
While not condizione di terminazione **do**
Identifica_I(s)
Identifica_T(s,k)
Identifica_A(s,k)
s' :=Scegli_s'_da_I(s,k)=N(s)-T(s,k)+A(s,k)
if $z(s') > z(s^*)$ **then**
$s^* := s'$
end if
$k := k + 1$
end while
return s^*

Tabella 3.1: Algoritmo base di Tabu Search

Term Memory (sezione 3.3). Nel resto del paragrafo introduciamo la notazione che utilizzeremo nel resto del seguito.

Notazione

Denotiamo con s una soluzione di MDP composta di m elementi $e_i \in N$, con $i = 1, ..., m$. Sia M l'insieme degli e_i elementi, $M = \{e_i \in N \ : \ i = 1, ..., m\}$. Data una soluzione s e un elemento $e_i \in N$, definiamo con $D_i^{in} = \sum_{j \in M} d_{ij}$ la diversità dell'elemento e_i rispetto la soluzione s e con $D_i^{out} = \sum_{j \in \overline{M}} d_{ij}$ la diversità con l'insieme degli elementi che non appartengono alla soluzione $\overline{M} = N \setminus M$. Sia $z(s) = \frac{1}{2} \sum_{j \in M} D_j^{in}$ il valore della funzione obiettivo associato alla soluzione s. Si ricorda che $d_{ij} = 0$ se $i = j$.

3.1 Componenti di base dell'algoritmo

Le componenti di base dell'algoritmo sono: la costruzione di una soluzione iniziale, l'intorno di una soluzione e l'implementazione delle liste tabu.

3.1.1 Costruzione di una soluzione iniziale

La costruzione di una soluzione s iniziale avviene utilizzando un algoritmo Greedy. Non essendo il Maximum Diversity Problem un **matroide**, l'approccio greedy rende possibile la costruzione di una soluzione ammissibile, senza garanzia di ottimalità.

La fase di costruzione è un processo iterativo nel quale, ad ogni iterazione, gli elementi $e_i \in \overline{M}$, vengono valutati da una funzione $g : N \to \Re_+$, più in particolare calcola la quantità D_i^{in} per ogni $e_i \notin \overline{M}$. Ad ogni passo diminuisce la distanza dalla soluzione inserendo l'elemento $e_i^* = \max(D_i^{in})$ che ha massima la somma delle distanze con gli elementi e_j che già compongono la soluzione M.

Per maggiore efficienza dell'algoritmo, si utilizza una struttura dati che associa ad ogni elemento $e_i \in N$ il corrispondente valore di D_i^{in} e D_i^{out}. L'aggiornamento della struttura comporta di attualizzare la somma delle distanze D_i^{in} e D_i^{out} ad ogni passo di costruzione della soluzione M: quando l'elemento e_j viene inserito in soluzione, si aggiunge e si toglie il relativo contributo d_{ij} rispettivamente in D_i^{in} ed in D_i^{out}, $\forall e_i, e_j \in N,\ e_i \neq e_j$. L'aggiornamento così fatto ha complessità $O(n)$. Per costruire una soluzione ammissibile sono necessari m passi, la complessità totale è quindi dell'ordine di $O(m\ n)$.

3.1.2 Intorno di una soluzione

Data una soluzione s esiste un intorno polinomiale $I(s)$ costituito da tutte le soluzioni s' ottenute scambiando un elemento $e_j \in M$ con un elemento $e_i \in \overline{M}$. L'esplorazione dell'intorno consiste nel determinare la migliore soluzione s^* tale che $z(s^*) = \max_{s' \in I(s)}\ z(s^{'})$, ovvero occorre valutare quale fra tutti i possibili scambi è quello più vantaggioso.

Il nuovo valore della funzione obiettivo $z(s')$ dopo uno scambio può essere calcolato come segue. Prima dello scambio il valore della funzione obiettivo è

$$z(s) = \frac{1}{2} \sum_{e_h \in M} D_h^{in} = D_j^{in} + (\frac{1}{2} \sum_{e_h \in M \setminus \{e_j\}} D_h^{in})$$

analogamente dopo aver scambiato e_j con e_i ottengo la nuova soluzione M' e il nuovo valore

$$z(s') = \frac{1}{2} \sum_{e_h \in M'} D_h^{in} = D_i^{in} + (\frac{1}{2} \sum_{e_h \in M' \setminus \{e_i\}} D_h^{in})$$

Dalla relazione $\Delta z = z(s') - z(s)$ si deduce il valore della nuova funzione obiettivo $z(s') = z(s) + \Delta z$; poiché $M \setminus \{e_j\} = M' \setminus \{e_i\}$, risulta allora che

$$z(s') = \frac{1}{2} \sum_{e_h \in M} D_h^{in} + (D_i^{in} - D_j^{in})$$

Lo scambio di due elementi qualsiasi come mostrato, permette di esplorare l'intorno definito conservando l'ammissibilità della soluzione.

3.1.3 Liste tabu

Se si rilascia il vincolo di trovare ad ogni iterazione una soluzione migliorante, cercando di volta in volta la migliore dell'intorno, c'è il rischio di cadere in una sequenza di soluzioni che determina un ciclo. Le liste tabu sono lo strumento che caratterizza gli algoritmi di Tabu Search. La loro funzione è duplice: cercare di evitare l'esplorazione ciclica di soluzioni e pilotare la ricerca verso nuovi ottimi locali. L'idea consiste nell'associare ad ogni iterazione dell'algoritmo, una lista di soluzioni tabu e quindi vietate. Una strategia efficace tiene traccia delle ultime soluzioni memorizzando le azioni necessarie per trasformare una soluzione in un'altra [8].

Nel nostro caso, l'azione che comporta la modifica di una soluzione, ovvero lo scambio tra gli elementi $e_j \in M$ e $e_i \in \overline{M}$, si traduce nella dichiarazione di tale mossa tabu per un certo numero di iterazioni successive. Gli elementi marchiati come tabu rimangono bloccati per un numero fissato di iterazioni (*lunghezza della lista tabu*), al termine delle quali riacquistano la possibilità di essere coinvolti in uno scambio.

Per questo problema abbiamo introdotto due liste tabu ℓ_e e ℓ_u di lunghezza diversa con l'obiettivo di controllare rispettivamente l'entrata e l'uscita degli elementi dalla soluzione. La lunghezza delle due liste dipende dalla dimensione dell'intorno delle soluzioni da esplorare.

Dettagli implementativi Per mantenere lo stato degli elementi, è stata utilizzata una struttura nella quale si tiene traccia dell'iterazione che ha comportato l'entrata o l'uscita di un elemento dalla soluzione. Dalla differenza di queste due informazioni si stabilisce lo stato di ogni elemento. Le liste tabu sono implementate con vettori di lunghezza pari al numero di elementi dell'insieme N, ad ogni posizione sono associate le due informazioni per determinare lo stato del nodo. Ogni volta che un elemento entra o esce dalla soluzione si aggiornano i relativi campi della struttura.

Con le componenti sinora presentate possiamo descrivere un algoritmo base di Tabu Search che chiameremo Tabu a 2 Liste (**T2L**). Questa versione è stata utilizzata per configurare al meglio le lunghezze delle due liste come verrà discusso nella sezione 3.2. I parametri derivati da questa sperimentazione sono:

$$\begin{cases} |\ell_e| = 11 \\ |\ell_u| = 5 \end{cases}$$

3.2 Short Term Memory

La Short Term Memory [11] è una tecnica che permette di intensificare la ricerca in zone ritenute promettenti oppure diversificarla in zone che non lo sono. L'individuazione del tipo di zona (promettente o meno) avviene mantenendo una memoria

della qualità delle soluzioni trovate nel breve periodo. Un periodo viene misurato in termini di numero di iterazioni. Una zona sarà ritenuta promettente se per un certo numero consecutivo di iterazioni (fase di miglioramento o *improving phase*) il valore della soluzione trovata è migliore del precedente, viceversa si riterrà non promettente se per un numero consecutivo di iterazioni (fase di peggioramento o *worsening phase*) il valore della soluzione peggiora.

Seguendo le specifiche date in [11], identifichiamo come fase di miglioramento una sequenza $\triangle ip$ di iterazioni consecutive che migliorano il valore della funzione obiettivo. Al contrario, una fase di peggioramento è una sequenza $\triangle wp$ di iterazioni consecutive che peggiorano il valore della funzione obiettivo. Per intensificare la ricerca, l'idea è quella di diminuire la lunghezza delle liste tabu, a fronte di una fase di miglioramento, consentendo quindi una più ampia esplorazione della zona in considerazione. Al contrario si allungano le liste se si verifica una fase di peggioramento; così facendo, alcune delle mosse che erano possibili, divengono tabu e la ricerca viene indirizzata verso altre zone dello spazio delle soluzioni. Per tenere traccia del numero di iterazioni consecutive miglioranti o peggioranti sono stati utilizzati dei contatori. Le lunghezze delle liste possono assumere valori discreti compresi in un intervallo definito; questi valori sono inizialmente impostati a L_{e_tenure} e L_{u_tenure} rispettivamente per le liste ℓ_e ed e variano nell'intervallo definito come

$$Min\_ip\_\ell_e \leq |\ell_e| \leq Max\_wp\_\ell_e \tag{3.1}$$

dove $Min\_ip\_\ell_e$ rappresenta il valore minimo assunto da ℓ_e nella fase di miglioramento $\triangle ip$ e $Max\_wp\_\ell_e$ rappresenta il massimo valore raggiungibile da ℓ_e nella fase di peggioramento $\triangle wp$. Anche ℓ_u segue una simile regola:

$$Min\_ip\_\ell_u \leq |\ell_u| \leq Max\_wp\_\ell_u \tag{3.2}$$

con $Min\_ip\_\ell_u$ e $Max\_wp\_\ell_u$ analogamente definiti per la lista tabu di uscita ℓ_u.

Le sperimentazioni, riportate nella sezione 4.2.2, sono state eseguite con questo set di parametri:

$$\begin{cases} \triangle wp = 5 \\ \triangle ip = 3 \\ L_{e\_tenure} = 11 \\ L_{u\_tenure} = 5 \\ Min\_ip\_\ell_e = 8 \\ Min\_ip\_\ell_u = 3 \\ Max\_wp\_\ell_e = 14 \\ Max\_wp\_\ell_u = 7 \end{cases}$$

La lunghezza delle liste ℓ_e, ℓ_u si muove di due passi verso il centro dell'intervallo (L_{e_tenure}, L_{u_tenure}) se il valore corrente della lunghezza della lista è distante almeno due unità. L'allontanamento, invece, comporta sempre un solo passo. Riassumendo:

se si verifica una ***fase di miglioramento*,**

$$\begin{cases} |l_e| = \max(|\ell_e| - 1,\ Min\_ip\_\ell_e) \\ |l_u| = \max(|\ell_u| - 1,\ Min\_ip\_\ell_u) \end{cases}$$

analogamente se si verifica una ***fase di peggioramento***,

$$\begin{cases} |l_e| = \min(|\ell_e| + 1,\ Max\_ip\_\ell_e) \\ |l_u| = \min(|\ell_u| + 1,\ Max\_ip\_\ell_u) \end{cases}$$

Considerato l'algoritmo Tabu Search di base con l'aggiunta della Short Term Memory, i risultati ottenuti hanno mostrato una maggiore rapidità nell'esplorazione dello spazio delle soluzioni ma un peggioramento, solo per particolari istanze, del valore delle soluzioni ottenute. Si è osservato che privilegiare le posizioni centrali dell'intervallo comporta un miglioramento delle soluzioni trovate. Sulla base di questo fatto, si è deciso di far variare la lunghezza delle liste, utilizzando un passo variabile, dipendente dalla distanza tra lunghezza corrente e valore iniziale delle lunghezza della lista.

Nel dettaglio, abbiamo che l'aggiornamento delle due liste avviene nel modo seguente:

se si verifica una ***fase di miglioramento***,

$$\begin{cases} |l_e| = \max(|\ell_e| - 2,\ Min\_ip\_\ell_e) & se\ \mathbf{abs}(\ |\ell_e| - L_{e\_tenure}) \geq 2 \\ |l_e| = \max(|\ell_e| - 1,\ Min\_ip\_\ell_e) & se\ \mathbf{abs}(\ |\ell_e| - L_{e\_tenure}) = 1 \end{cases}$$

analogamente per la lista ℓ_u

$$\begin{cases} |l_u| = \max(|\ell_u| - 2,\ Min\_ip\_\ell_u) & se\ \mathbf{abs}(\ |\ell_u| - L_{u\_tenure}) \geq 2 \\ |l_u| = \max(|\ell_u| - 1,\ Min\_ip\_\ell_u) & se\ \mathbf{abs}(\ |\ell_u| - L_{u\_tenure}) = 1 \end{cases}$$

mentre, se si verifica una **fase di peggioramento**,

$$\begin{cases} |l_e| = \min(|\ell_e| + 2,\ Max\_wp\_\ell_e) & se\ \mathbf{abs}(\ |\ell_e| - L_{e\_tenure}) \geq 2 \\ |l_e| = \min|\ell_e| + 1,\ Max\_wp\_\ell_e) & se\ \mathbf{abs}(\ |\ell_e| - L_{e\_tenure}) = 1 \end{cases}$$

analogamente per la lista ℓ_u

$$\begin{cases} |l_u| = \min(|\ell_u| + 2,\ Max\_wp\_\ell_u) & se\ \mathbf{abs}(\ |\ell_u| - L_{u\_tenure}) \geq 2 \\ |l_u| = \min(|\ell_u| + 1,\ Max\_wp\_\ell_u) & se\ \mathbf{abs}(\ |\ell_u| - L_{u\_tenure}) = 1 \end{cases}$$

La versione di Tabu Search così ottenuta, prende il nome di Tabu Short Term (**TST**).

3.3 Long Term Memory

Con la versione TST si è cercato di mantenere memoria delle soluzioni recentemente esplorate con l'obiettivo di intensificare la ricerca nelle zone ritenute particolarmente promettenti. Tuttavia questo metodo non è sempre sufficiente a intensificare o diversificare l'esplorazione dello spazio delle soluzioni. Introducendo il concetto di memoria a lungo termine si cerca di ovviare a questo limite.

La memoria a lungo termine si basa sulla seguente osservazione. Durante l'esplorazione dell'intorno, si possono generare più soluzioni di buona qualità: lo schema

di Tabu Search prende la migliore tra queste e trascura le altre. L'idea è quindi di mantenere memoria di queste soluzioni dalle quali eventualmente far ripartire la ricerca.

Un metodo per realizzare la memoria a lungo termine è quello di mantenere un insieme S di *seconde* soluzioni: una seconda soluzione è la seconda migliore soluzione, rispetto la funzione obiettivo, individuata durante l'esplorazione dell'intorno. La cardinalità di S è un parametro dell'algoritmo. Al verificarsi di una certa condizione, la migliore soluzione $s \in S$ viene estratta e da essa viene fatta ripartire la ricerca dell'algoritmo. Questa fase, detta anche *di restart*, effettua una diversificazione perché si cambia radicalmente la zona dello spazio delle soluzioni, ed effettua un'intensificazione perché la ricerca viene fatta ripartire da un punto ritenuto promettente. Una più lunga trattazione dell'idea è data in [11].

La nostra implementazione realizza S come una lista contenente le seconde migliori soluzioni s' trovate durante l'esplorazione di un dato intorno $I(s)$. Per migliorare l'operazione di estrazione, la lista è mantenuta ordinata per valori non decrescenti di $z(s')$. Viceversa, una seconda migliore soluzione s' è aggiunta alla lista se $z(s') > z(s_{\min})$ dove $s_{\min}$ è la soluzione di valore più basso in S, oppure quando la lista non è piena.

Per consentire alla ricerca di ripartire nelle stesse condizioni presenti al momento del salvataggio della seconda soluzione, associamo ad ogni elemento s' della lista lo stato della computazione corrente dato dalla soluzione s, dalle liste tabu ℓ_e ed ℓ_u ed i parametri relativi alla gestione della Short Term Memory.

La condizione che abbiamo imposto per effettuare un restart è la seguente: se per I_{c_1} iterazioni consecutive non è stato migliorato il valore della migliore soluzione trovata si esegue un restart . Denotiamo questa condizione con C_1. Un'ulteriore condizione, detta C_2, è quella che prevede un restart se la lunghezza della lista tabu si mantiene maggiore del valore di tabu tenure per I_{C_2} iterazioni consecutive. La condizione C_1 è una sorta di indicatore *globale* del fatto che siamo in una zona non promettente e che una diversificazione può migliorare l'esplorazione. La condizione C_2 indica invece la difficoltà della ricerca di spostarsi da una zona non promettente solo con gli strumenti della Short Term Memory.

Denotiamo con Tabu Long Term (**TLT**) l'algoritmo ottenuto aggiungendo a TST la tecnica di Long Term Memory con la sola condizione C_1 di restart. La versione estesa, comprensiva della condizione C_2, è stata utilizzata per raffinare la ricerca di ottimi per un particolare sottoinsieme di istanze. Tale analisi è riportata nel capitolo 4.

Capitolo 4

Discussione dei risultati sperimentali

4.1 Strumenti e dati utilizzati

Gli algoritmi sono stati implementati in C standard 2 (1999), compilati con g++ 3.3.6 ed eseguiti su un PC Intel Pentium 4 Mobile 2.8Ghz con 512Mb di RAM, in ambiente Linux OS. Per verificare le prestazioni degli algoritmi sono state utilizzati i benchmark di istanze proposte in letteraura.

Il primo benchmark proposto da Silva[16] è costituito da 20 istanze. Le istanze hanno dimensioni variabili: $n = \{100, 200, 300, 400, 500\}$ con valori di m pari a 10%,20%,30%,40% di n. La matrice delle distanze $D = [d_{ij}]$ è costruita estraendo casualmente valori da una distribuzione uniforme in $[0, 9]$. Il nome delle istanze è formato dal prefisso *matriz*, seguito dal numero di elementi n, quindi la lettera 'm' e il rispettivo valore; ad esempio matrizn100m10 indica $n = 100$ e $m = 10$.

Il secondo bechmark è composto di 40 istanze ed è stato proposto da Andrade [1, 2]. La popolazione ha una popolazione di $n = \{50, 100, 150, 200, 250\}$ m è pari al 20%,40% di n.

Questo set di istanze è diviso in quattro tipi:

- **Tipo A**: I valori delle componenti spaziali degli elementi $\mathbf{e}_i$ sono estratte casualmente da una distribuzione uniforme in $[1, 9]$; la distanza d_{ij} definita per ogni coppia di elementi è calcolata come distanza euclidea.
- **Tipo B**: la distanza d_{ij} viene estratta in modo casuale da una distribuzione uniforme intera in $[1, 9999]$.
- **Tipo C**: il 50% delle distanze d_{ij} sono estratte casualmente da una distribuzione uniforme intera in $[1, 9999]$, il rimanente 50% da una distribuzione uniforme intera in $[1, 4999]$.
- **Tipo D**: il 50% delle distanze d_{ij} sono estratte casualmenete da una distribuzione uniforme intera in $[1, 9999]$, il rimanente 50% da una distribuzione uniforme intera in $[5000, 9999]$.

Le istanze di questo dataset sono identificate nel seguente modo: la prima lettera indica il tipo, il primo gruppo di numeri rappresenta la dimensione della popolazione seguita dalla lettera '*m*' e quindi dal valore di m; ad esempio *a050m10* indica: istanza di tipo A, $n = 50,\ m = 10$.

Su questi due benchmark, i migliori risultati sono stati ottenuti con il GRASP proposto in [16] e in [16, 2]: si osserva infatti che i migliori risultati noti per il benchmark proposto da Andrade sono stati ottenuti con il GRASP di Silva.

Per quanto riguarda i test effettuati in [16], le nove versioni di GRASP proposte sono state implementate in C++, compilate con g++ versione 3.2.2 e testate su un PC AMD Athlon 1.3GHz con 256Mb di RAM. I test pubblicati sono stati eseguiti con la seguente modalità: ogni istanza è risolta con tutte e nove le versioni di GRASP, per tre volte, generando ogni volta un seme (*seed*) diverso. Ogni soluzione è stata ottenuta con 1000 iterazioni di GRASP. Per ogni istanza sono stati pubblicati: il miglior valore di z trovato, la media dei 3 test e il tempo di calcolo.

4.2 Risultati

In questa sezione sono riportati i risultati dei test condotti sui due benchmark di istanze rispettivamente proposte in [16] e [2]. I risultati mettendo in evidenza il confronto di ogni singolo algoritmo con i risultati migliori risultati presi dalla letteratura, confrontando gli uni rispetto agli altri. Nelle prossime sezioni saranno illustrati più nel dettaglio le modalità dei test.

Modalità di confronto dei risultati

Nelle sezioni che seguono sono riportate le tabelle di confronto con i risultati pubblicati in [16] e in [2]. Le tabelle 4.1, 4.3 e 4.5 mostrano i risultati ottenuti con le nostre versioni di Tabu Search, rispettivamente T2L, TST e TLT. Ognuna riporta sia i valori relativi alla nostra versione di algoritmo che i valori di confronto ottenuti da Silva. Per quanto riguarda z^* di Silva, è stato riportato il valore migliore dei 3 test, trovato con una delle versioni GRASP che ha ottenuto il risultato migliore sulla particolare istanza. Per completezza è riportata anche la media dei 3 test. Analogamente per i tempi di calcolo di Silva, sono riportati i tempi associati ai due valori z^* e z *avg*. I campi delle tabelle indicano: il tipo di istanza (*Istanza*), il valore medio della funzione obiettivo dei 3 test condotti sull'istanza (Silva z *avg*), il miglior valore trovato con la migliore versione GRASP (Silva z^*), il valore della soluzione trovata dalla versione del nostro algoritmo ({T2L,TST,TLT} z^*), il gap Δz calcolato come $z^*(\{T2L, TST, TLT\}) - z^*(Silva)$, il gap in percentuale calcolato come $\frac{\Delta z}{z^*(Silva)} \cdot 100$, il tempo di calcolo per trovare Silva z *avg* (Silva: *time (s) z avg*), il tempo di calcolo per trovare Silva z^* (Silva: *time (s)* z^*), il tempo per trovare {T2L,TST,TLT} z^* ({T2L,TST,TLT} *time (s)*), infine, l'ultima iterazione migliorante (U.M. *iteraz.*) e il tempo di calcolo necessario per trovarla *(*U.M. *time (s)*).

Le tabelle 4.2, 4.4 e 4.6 riportano i confronti tra ognuna delle nostre versioni di Tabu Search (T2L, TST, TLT) con i valori *Best Known* riportati da Andrade [2] ma ottenuti con le metaeuristiche di Silva [16]. Per i motivi sopra riportati, è stato possibile mostrare il confronto con i valori z^* ma non con i tempi di calcolo. La modalità di confronto fatto con le istanze di Andrade è analogo a quello fatto con quelle di Silva. I campi delle tabelle citate indicano: il tipo di istanza (*Istanza*), il valore z^* conosciuto ottenuto con una particolare versione di GRASP (Andrade z^*), il valore della migliore soluzione trovata dalla versione del nostro algoritmo ({T2L,TST,TLT} z^*), il gap Δz calcolato come $z^*(\{T2L, TST, TLT\}) - z^*(Andrade)$, il gap in percentuale calcolato come $\frac{\Delta z}{z^*(Andrade)} \cdot 100$, il tempo di calcolo per trovare {T2L,TST,TLT} z^* ({T2L,TST,TLT} *time (s)*), l'ultima iterazione migliorante (U.M. *iteraz.*) e il tempo di calcolo necessario per trovarla *(*U.M. *time (s)*).

4.2.1 Tabu a due liste

Il Tabu a due liste (T2L) è la prima versione presentata, i risultati mostrati sono ottenuti con I_{max} iterazioni massime di Tabu Search. Si è inoltre imposto di terminare il test se per I_c iterazioni consecutive il valore z^* non migliora. La configurazione dei parametri utilizzati è la seguente:

$$\begin{cases} |\ell_e| = 11 \\ |\ell_u| = 5 \\ I_{max} = 2000 \\ I_c = 1000 \end{cases}$$

La tabella 4.1 mostra i risultati ottenuti sulle istanze di Silva [16].

Confrontando i risultati ottenuti con gli algoritmi di Silva si osserva che l'algoritmo T2L ha migliorato il valore di z^* in 6 istanze (in grassetto), determinando i nuovi Best Known, lo ha raggiunto in 11 istanze, mentre in sole 3 istanze l'algoritmo T2L ha trovato una soluzione peggiore di Silva. Quasi tutti i migliori risultati sono stati trovati prima di 1000 iterazioni, solo per 3 istanze di grosse dimensioni (matrizn400m40, matrizn500m50, matrizn500m200) l'algoritmo ha trovato soluzioni miglioranti oltre le 1000 iterazioni. L'algoritmo ha impiegato un tempo di calcolo pari a 0,05 secondi per risolvere l'istanza più piccola (matrizn100m10) mentre sono serviti 10 minuti e 40 secondi per terminare il test dell'istanza più grossa (matrizn500m200). Confrontando i tempi di calcolo si osserva che il divario cresce molto rapidamente all'aumentare della dimensione del problema. Ad esempio, per risolvere l'istanza più grossa, il nostro algoritmo impiega 10 minuti (T2L) contro le 24 ore del migliore test fatto con uno dei 9 GRASP. Se consideriamo il tempo di calcolo necessario per raggiungere l'ultima iterazione di miglioramento, è necessario ancora meno tempo, in media inferiore al 50%.

La tabella 4.2 mostra i risultati ottenuti sulle istanze proposte da Andrade [1]. Il numero di iterazioni necessarie per trovare z^* è molto inferiore rispetto all'altro set di istanze, la media è solamente di 264 iterazioni; ci sono alcune eccezioni: per le istanze a200m80 e a200m80 si è trovata una soluzione migliorante appena

	Silva		**T2L**	**Gap** z^*		**Silva:** ***time*** (s)		**T2L**	**U.M.**	
Istanza	z ***avg***	z^*	z^*	Δz	**%**	z ***avg***	z^*	***time*** **(s)**	***iteraz.***	***time*** **(s)**
matrizn100m10	333	333	333	0	0,00	0,40	0,40	0,05	34	0,00
matrizn100m20	1195	1195	1195	0	0,00	28,20	27,40	0,21	76	0,02
matrizn100m30	2457	2457	2457	0	0,00	57,40	56,90	0,44	119	0,05
matrizn100m40	4142	4142	4142	0	0,00	89,20	88,30	0,70	103	0,06
matrizn200m20	1247	1247	1247	0	0,00	71,70	71,00	0,73	66	0,07
matrizn200m40	4449,3	4450	4450	0	0,00	510,30	507,30	4,35	576	1,57
matrizn200m60	9437	9437	9437	0	0,00	626,60	628,60	6,25	139	0,79
matrizn200m80	16225	16225	16225	0	0,00	1647,00	1638,60	10,37	272	2,18
matrizn300m30	2692	2694	2694	0	0,00	1405,30	1402,30	10,60	590	3,90
matrizn300m60	9682,5	9684	**9689**	**5**	**0,05**	3450,40	3613,17	37,19	830	16,71
matrizn300m90	20727	20734	**20743**	**9**	**0,04**	10514,60	10420,90	51,85	552	18,87
matrizn300m120	35881	35881	35875	-6	-0,02	17517,72	17562,01	79,56	715	33,17
matrizn400m40	4654,7	4658	4658	0	0,00	3371,00	3370,20	44,50	1342	30,22
matrizn400m80	16946,7	16956	16931	-25	-0,15	10048,60	10161,60	94,36	507	32,02
matrizn400m120	36301	36315	**36317**	**2**	**0,01**	19201,30	19220,90	168,47	607	62,26
matrizn400m160	62478	62483	**62487**	**4**	**0,01**	32843,60	32550,50	255,48	747	107,81
matrizn500m50	7124	7130	**7141**	**11**	**0,15**	2934,90	2935,60	91,04	1171	53,78
matrizn500m100	26229	26237	**26258**	**21**	**0,08**	26513,30	26473,10	259,61	884	119,55
matrizn500m150	57055,7	58605	56572	-2.033	-3,47	29051,40	27546,00	239,80	102	21,65
matrizn500m200	97333	97344	97344	0	0,00	86247,00	86513,80	640,29	1292	413,20

Tabella 4.1: Risultati ottenuti con la versione Tabu a 2 liste (**T2L**) sulle istanze di Silva [16]. U.M. indica l'ultimo miglioramento della z^*.

	Andrade	**T2L**	**Gap** z^*		**T2L**	**U.M.**	
Istanza	z^*	z^*	Δz	**%**	***time (s)***	***iteraz.***	***time (s)***
a050m10	492	492	0	0,00	0,04	909	0,02
a050m20	1932	1932	0	0,00	0,13	1982	0,13
a100m20	2007	2007	0	0,00	0,23	158	0,04
a100m40	7730	7730	0	0,00	0,72	168	0,11
a150m30	4552	4552	0	0,00	1,63	890	0,77
a150m60	17482	17482	0	0,00	3,17	0	0,00
a200m40	8132	8132	0	0,00	2,97	4	0,04
a200m80	31049	31049	0	0,00	18,82	1998	18,81
a250m50	12653	12654	**1**	0,01	7,55	0	0,00
a250m100	48384	48384	0	0,00	24,32	115	2,54
b050m10	334976	334976	0	0,00	0,02	0	0,00
b050m20	1171416	1171416	0	0,00	0,07	1	0,00
b100m20	1267277	1267277	0	0,00	0,21	36	0,01
b100m40	4544642	4544642	0	0,00	0,67	67	0,04
b150m30	2758381	2758381	0	0,00	1,11	266	0,25
b150m60	9960461	9960461	0	0,00	3,67	206	0,64
b200m40	4788086	4788086	0	0,00	4,56	495	1,50
b200m80	17544448	17544447	-1	0,00	10,83	103	0,99
b250m50	7388997	7378891	-10.106	-0,14	14,54	836	6,69
b250m100	27162906	27168460	**5.554**	0,02	22,87	27	0,63
c050m10	316409	316409	0	0,00	0,02	7	0,00
c050m20	1094343	1094343	0	0,00	0,06	1	0,00
c100m20	1205722	1207522	**1.800**	0,15	0,18	2	0,00
c100m40	4219476	4219476	0	0,00	0,53	11	0,01
c150m30	2613286	2613286	0	0,00	0,72	16	0,03
c150m60	9374611	9374611	0	0,00	2,00	12	0,03
c200m40	4630545	4630545	0	0,00	2,59	20	0,07
c200m80	16759895	16759895	0	0,00	9,42	219	1,74
c250m50	7178043	7178043	0	0,00	7,30	283	1,63
c250m100	26047022	26047022	0	0,00	20,62	285	4,62
d050m10	381379	381379	0	0,00	0,02	90	0,00
d050m20	1502908	1502908	0	0,00	0,06	7	0,00
d100m20	1570800	1570800	0	0,00	0,24	251	0,05
d100m40	6067776	6067776	0	0,00	0,64	170	0,09
d150m30	3502567	3502567	0	0,00	0,82	59	0,06
d150m60	13611261	13611262	**1**	0,00	2,59	36	0,10
d200m40	6207580	6207580	0	0,00	2,29	46	0,12
d200m80	24133320	24133321	**1**	0,00	8,00	141	1,02
d250m50	9685430	9685430	0	0,00	9,99	637	3,92
d250m100	37753120	37753118	-2	0,00	16,72	17	0,32

Tabella 4.2: Risultati ottenuti con la versione Tabu a 2 liste (**T2L**) sulle istanze di Andrade [2]. U.M. indica l'ultimo miglioramento della z^*.

prima di terminare il test. I tempi necessari per trovare le migliori soluzioni vanno da 2 centesimi di secondo (b050m10) a 24,32 per l'istanza a250m100. Dato che la colonna dell'ultima iterazione migliorante ha una media molto bassa rispetto a I_{max}, lo sono anche i tempi necessari per trovare la migliore soluzione. Il T2L ha trovato 5 nuovi Best Known (in grassetto), ha peggiorato il valore della funzione obiettivo in 3 istanze su 40, di cui 2 molto lievemente, nelle rimanenti 32 è stato determinato lo stesso valore trovato da Andrade. L'algoritmo Greedy ha trovato la migliore soluzione delle istanze a150m60, a250m50 e b050m10 proposte da Andrade.

4.2.2 Tabu con Short Term Memory

La versione di Tabu Search con Short Term Memory è stata testata con il seguente set di parametri:

$$\begin{cases} \triangle wp = 5 \\ \triangle ip = 3 \\ L_{e\_tenure} = 11 \\ L_{u\_tenure} = 5 \\ Min\_ip\_\ell_e = 8 \\ Min\_ip\_\ell_u = 3 \\ Max\_wp\_\ell_e = 14 \\ Max\_wp\_\ell_u = 7 \\ I_{max} = 2000 \\ I_c = 1000 \end{cases}$$

L'algoritmo ha un limite di I_{max} iterazioni oppure termina se dopo I_c iterazioni consecutive non si trovano soluzioni globalmente miglioranti.

La tabella 4.3 mostra i risultati del TST sulle istanze di Silva ([16]). In media il TST ha una maggiore velocità di convergenza (502 iterazioni) rispetto al T2L (536): solo le istanze matrizn400m80 e matrizn500m50 richiedono più di 1000 iterazioni per trovare la migliore soluzione. Il tempo di calcolo è molto simile a quello servito per il T2L.

Confrontando i risultati del TST ottenuti, con le euristiche di Silva si osserva che l'algoritmo ha raggiunto la stessa z^* in 10 istanze; 5 (in grassetto) sono state migliorate determinando i nuovi Best Known e 5 su 20 sono state peggiorate.

La tabella 4.4 mostra i risultati del TST sulle istanze proposte da Andrade in [1]. Il numero di iterazioni necessarie per trovare z^* è molto maggiore rispetto a quello necessario per l'algoritmo T2L, la media è di 441 iterazioni. La colonna che contiene l'ultima iterazione di miglioramento mostra che 5 istanze migliorano appena prima di raggiungere l'iterazione I_{max}. L'algoritmo TST ha trovato 5 nuovi Best Known (in grassetto), ha peggiorato 3 istanze su 40, di cui 2 molto lievemente. Il valore della funzione obiettivo relativo alle rimanenti 32 istanze è stato raggiunto.

	Silva		**TST**	**Gap z^***		**Silva: *time* (s)**		**TST**	**U.M.**	
Istanza	z ***avg***	z^*	z^*	Δz	**%**	z ***avg***	z^*	***time*** (s)	***iteraz.***	***time*** (s)
matrizn100m10	333	333	333	0	0,00	0,40	0,40	0,06	90	0,01
matrizn100m20	1195	1195	1195	0	0,00	28,20	27,40	0,23	203	0,04
matrizn100m30	2457	2457	2457	0	0,00	57,40	56,90	0,41	53	0,02
matrizn100m40	4142	4142	4142	0	0,00	89,20	88,30	0,70	113	0,07
matrizn200m20	1247	1247	1241	-6	-0,48	71,70	71,00	0,61	26	0,04
matrizn200m40	4449,3	4450	4448	-2	-0,04	510,30	507,30	3,08	191	0,49
matrizn200m60	9437	9437	9437	0	0,00	626,60	628,60	6,44	196	1,12
matrizn200m80	16225	16225	16225	0	0,00	1647,00	1638,60	10,39	235	1,97
matrizn300m30	2692	2694	2694	0	0,00	1405,30	1402,30	8,65	216	1,54
matrizn300m60	9682,5	9684	**9689**	**5**	**0,05**	3450,40	3613,17	34,36	629	12,99
matrizn300m90	20727	20734	**20743**	**9**	**0,04**	10514,60	10420,90	44,97	293	10,34
matrizn300m120	35881	35881	35881	0	0,00	17517,72	17562,01	84,62	789	37,45
matrizn400m40	4654,7	4658	4651	-7	-0,15	3371,00	3370,20	35,99	596	13,83
matrizn400m80	16946,7	16956	16956	0	0,00	10048,60	10161,60	127,04	1471	93,27
matrizn400m120	36301	36315	**36317**	**2**	**0,01**	19201,30	19220,90	194,15	868	88,77
matrizn400m160	62478	62483	**62484**	**1**	**0,00**	32843,60	32550,50	247,10	700	100,61
matrizn500m50	7124	7130	7129	-1	-0,01	2934,90	2935,60	93,11	1720	80,25
matrizn500m100	26229	26237	**26258**	**21**	**0,08**	26513,30	26473,10	186,36	361	48,54
matrizn500m150	57055,7	58605	56572	-2.033	-3,47	29051,40	27546,00	294,64	335	72,89
matrizn500m200	97333	97344	97344	0	0,00	86247,00	86513,80	614,86	968	304,41

Tabella 4.3: Risultati ottenuti con la versione Tabu Short Term Memory (**TST**) sulle istanze di Silva [16]. U.M. indica l'ultimo miglioramento della z^*.

	Andrade	**TST**	**Gap** z^*		**TST**	**U.M**	
Istanza	z^*	z^*	Δz	***%***	***time (s)***	***iteraz.***	***time (s)***
a050m10	492	492	0	0,00	0,04	1996	0,04
a050m20	1932	1932	0	0,00	0,13	1998	0,13
a100m20	2007	2007	0	0,00	0,40	1588	0,31
a100m40	7730	7730	0	0,00	1,21	1995	1,21
a150m30	4552	4552	0	0,00	1,50	851	0,70
a150m60	17482	17482	0	0,00	3,16	0	0,00
a200m40	8132	8132	0	0,00	5,23	879	2,38
a200m80	31049	31049	0	0,00	18,81	1998	18,80
a250m50	12653	**12654**	**1**	**0,01**	9,13	0	0,00
a250m100	48384	48384	0	0,00	37,91	624	14,62
b050m10	334976	334976	0	0,00	0,02	0	0,00
b050m20	1171416	1171416	0	0,00	0,06	1	0,00
b100m20	1267277	1267277	0	0,00	0,21	33	0,01
b100m40	4544642	4544642	0	0,00	0,88	404	0,25
b150m30	2758381	2758381	0	0,00	1,41	642	0,56
b150m60	9960461	9959120	-1.341	-0,01	6,11	1186	3,60
b200m40	4788086	4788086	0	0,00	4,37	480	1,37
b200m80	17544448	17544447	-1	0,00	10,47	82	0,79
b250m50	7388997	7379797	-9.200	-0,12	19,57	1164	11,31
b250m100	27162906	**27168460**	**5.554**	**0,02**	26,66	155	3,54
c050m10	316409	316409	0	0,00	0,02	7	0,00
c050m20	1094343	1094343	0	0,00	0,06	1	0,00
c100m20	1205722	**1207522**	**1.800**	**0,15**	0,18	2	0,00
c100m40	4219476	4219476	0	0,00	0,52	11	0,01
c150m30	2613286	2613286	0	0,00	0,69	16	0,02
c150m60	9374611	9374611	0	0,00	2,03	12	0,04
c200m40	4630545	4630545	0	0,00	2,43	20	0,08
c200m80	16759895	16759895	0	0,00	8,47	86	0,69
c250m50	7178043	7178043	0	0,00	6,31	73	0,48
c250m100	26047022	26047022	0	0,00	18,72	71	1,28
d050m10	381379	381379	0	0,00	0,02	43	0,00
d050m20	1502908	1502908	0	0,00	0,06	7	0,00
d100m20	1570800	1570800	0	0,00	0,20	37	0,01
d100m40	6067776	6067776	0	0,00	0,60	42	0,03
d150m30	3502567	3502567	0	0,00	0,80	59	0,05
d150m60	13611261	**13611262**	**1**	**0,00**	2,61	45	0,13
d200m40	6207580	6207580	0	0,00	2,51	84	0,23
d200m80	24133320	**24133321**	**1**	**0,00**	7,28	48	0,36
d250m50	9685430	9685430	0	0,00	12,59	896	6,00
d250m100	37753120	37753118	-2	0,00	17,29	17	0,33

Tabella 4.4: Risultati ottenuti con la versione Tabu Short Term (**TST**) sulle istanze di Andrade [2]. U.M. indica l'ultimo miglioramento della z^*.

4.2.3 Tabu con Long Term Memory

II test sul TLT sono stati condotti con la con la seguente configurazione di parametri:

$$\begin{cases} I_{tot} = 2000 \\ I_c = 300 \\ |S| = 15 \end{cases}$$

i parametri rimanenti sono gli stessi della versione TST. Si eseguono 2000 iterazioni, il restart avviene se per I_c iterazioni consecutive non si trova una soluzione globalmente migliorante. La struttura che contiene le seconde migliori soluzioni ha lunghezza pari a $|S|$.

La tabella 4.5 mostra i risultati sulle istanze di Silva [16]. Per trovare le migliori soluzioni ci sono volute in media 700 iterazioni; in 6 istanze si è trovata una soluzione migliorante dopo l'iterazione 1000. Il numero di restart è conpreso tra 3 e 6 con una media di 4,75. I risultati mostrano 6 nuovi Best Known (in grassetto), gli stessi trovati con la versione T2L. L'algoritmo TLT ha peggiorato 3 istanze du 20 mentre le restanti 32 hanno ottenuto lo stesso valore noto in letteratura. Il confronto fra i tempi di calcolo mostra come al solito un grosso divario.

L'algoritmo TLT è stato testato anche sulle istanze proposte da Andrade, la tabella 4.6 mostra i risultati ottenuti. L'algoritmo ha avuto bisogno in media di 437 iterazioni per raggiungere le migliori soluzioni (minore del T2L e TST). La media dei restart è molto maggiore di quella mostrata nel caso delle istanze di Silva: 26 istanze su 40 hanno fatto 6 restart. Il confronto mostra che l'algoritmo TLT ha permesso di determinare 5 nuovi Best Known, in 32 istanze si è trovato il miglior risultato noto in letteratura mentre si è peggiorato il risultato relativo a 3 istanze.

Stress test Sulle istanze che che non hanno raggiunto i migliori risultati noti in letteratura, si è deciso di eseguire uno stress test, per capire se il mancato raggiungimento è legato alla particolarità dell'istanza oppure costituisce un limite della versione TLT. Per il test è stata scelta la versione di Tabu Search più completa presentata: il TLT con la condizione C_2 di restart. I test sono stati condotti provando a risolvere ogni singola istanza con una configurazione di parametri ogni volta diversa. Combinando i parametri: I_{tot},I_{c2} , I_c e $|S|$, si sono ottenute 120 configurazioni diverse. Nel test sono state inserite 3 istanze appartenenti al benchmark proposto da Silva e 3 proposte da Andrade. L'istanza matrizn400m40 ha raggiunto il valore migliore conosciuto con il parametro $I_{c2} = 5$, mantenendo invariati gli altri rispetto alla configurazione originale del TLT originale, mentre l'istanza matrizn400m80 ha raggiunto il risultato di Silva con $I_{c2} = 30$. Per l'istanza matrizn500m150 non c'è stato nulla da fare, nessuna delle 120 configurazioni di parametri ha riportato dei miglioramenti. Per quanto riguarda le 3 istanze proposte da Andrade incluse nello *stress test*, il TLT non è stato in grado di trovare valori della z^* migliori o uguali a quelli della letteratura.

	Silva		**TLT**	**Gap** $z*$		**Silva:** ***time*** (s)		**TLT**	**U.M.**		**Restart**
Istanza	z ***avg***	z^*	z^*	Δz	**%**	z ***avg***	z^*	***time*** (s)	***iteraz.***	***time*** (s)	***num.***
matrizn100m10	333	333	333	0	0,00	0,40	0,40	0,10	90	0,01	6
matrizn100m20	1195	1195	1195	0	0,00	28,20	27,40	0,37	203	0,04	5
matrizn100m30	2457	2457	2457	0	0,00	57,40	56,90	0,75	53	0,02	6
matrizn100m40	4142	4142	4142	0	0,00	89,20	88,30	1,21	113	0,07	6
matrizn200m20	1247	1247	1247	0	0,00	71,70	71,00	1,33	874	0,58	5
matrizn200m40	4449,3	4450	4450	0	0,00	510,30	507,30	5,35	1.045	2,81	5
matrizn200m60	9437	9437	9437	0	0,00	626,60	628,60	11,20	196	1,15	5
matrizn200m80	16225	16225	16225	0	0,00	1647,00	1638,60	17,39	235	2,00	5
matrizn300m30	2692	2694	2694	0	0,00	1405,30	1402,30	13,09	216	1,42	5
matrizn300m60	9682,5	9684	**9689**	**5**	**0,05**	3450,40	3613,17	39,40	629	12,41	4
matrizn300m90	20727	20734	**20743**	**9**	**0,04**	10514,60	10420,90	69,89	293	10,27	5
matrizn300m120	35881	35881	35881	0	0,00	17517,72	17562,01	93,18	1.147	53,32	5
matrizn400m40	4654,7	4658	4651	-7	-0,15	3371,00	3370,20	45,20	596	13,64	4
matrizn400m80	16946,7	16956	16935	-21	-0,12	10048,60	10161,60	128,28	1.531	97,84	4
matrizn400m120	36301	36315	**36317**	**2**	**0,01**	19201,30	19220,90	208,22	1.649	171,53	5
matrizn400m160	62478	62483	**62487**	**4**	**0,01**	32843,60	32550,50	295,32	1.390	204,76	4
matrizn500m50	7124	7130	**7141**	**11**	**0,15**	2934,90	2935,60	94,33	1.473	69,84	3
matrizn500m100	26229	26237	**26258**	**21**	**0,08**	26513,30	26473,10	272,45	361	49,44	5
matrizn500m150	57055,7	58605	56572	-2.033	-3,47	29051,40	27546,00	446,43	335	73,99	5
matrizn500m200	97333	97344	97344	0	0,00	86247,00	86513,80	627,11	968	307,78	3

Tabella 4.5: Risultati ottenuti con la versione Tabu Long Term Memory (**TLT**) sulle istanze di Silva[16]. U.M. indica l'ultimo miglioramento della z^*.

	Andrade	**TLT**	**Gap** z^*		**TLT**	**U.M.**		**Restart**
Istanza	z^*	z	Δz	***%***	***time (s)***	***iteraz.***	***time (s)***	***num.***
a050m10	492	492	0	0,00	0,04	1996	0,04	0
a050m20	1932	1932	0	0,00	0,12	1372	0,08	4
a100m20	2007	2007	0	0,00	0,39	522	0,10	4
a100m40	7730	7730	0	0,00	1,19	1237	0,74	3
a150m30	4552	4552	0	0,00	1,65	1200	0,99	4
a150m60	17482	17482	0	0,00	6,29	0	0,00	6
a200m40	8132	8132	0	0,00	5,96	1510	4,50	5
a200m80	31049	31049	0	0,00	19,28	1998	19,27	0
a250m50	12653	**12654**	**1**	**0,01**	14,94	0	0,00	6
a250m100	48384	48384	0	0,00	45,30	1999	45,30	3
b050m10	334976	334976	0	0,00	0,04	0	0,00	6
b050m20	1171416	1171416	0	0,00	0,12	1	0,00	6
b100m20	1267277	1267277	0	0,00	0,37	33	0,01	6
b100m40	4544642	4544642	0	0,00	1,17	384	0,22	6
b150m30	2758381	2758381	0	0,00	1,76	1594	1,41	5
b150m60	9960461	9960461	0	0,00	6,14	705	2,19	5
b200m40	4788086	4788086	0	0,00	5,78	480	1,30	5
b200m80	17544448	17544447	-1	0,00	18,90	82	0,78	6
b250m50	7388997	7379278	-9719	-0,13	15,99	708	5,59	5
b250m100	27162906	**27168460**	**5554**	**0,02**	44,61	155	3,37	6
c050m10	316409	316409	0	0,00	0,04	7	0,00	6
c050m20	1094343	1094343	0	0,00	0,11	1	0,00	6
c100m20	1205722	**1207522**	**1800**	**0,15**	0,34	2	0,00	6
c100m40	4219476	4219476	0	0,00	1,03	11	0,01	6
c150m30	2613286	2613286	0	0,00	1,34	16	0,02	6
c150m60	9374611	9374611	0	0,00	3,90	12	0,04	6
c200m40	4630545	4630545	0	0,00	4,73	20	0,08	6
c200m80	16759895	16759895	0	0,00	14,68	86	0,65	6
c250m50	7178043	7178043	0	0,00	11,12	73	0,46	6
c250m100	26047022	26047022	0	0,00	32,96	71	1,22	6
d050m10	381379	381379	0	0,00	0,04	43	0,00	6
d050m20	1502908	1502908	0	0,00	0,12	7	0,00	6
d100m20	1570800	1570800	0	0,00	0,36	37	0,01	6
d100m40	6067776	6067776	0	0,00	1,05	42	0,02	6
d150m30	3502567	3502567	0	0,00	1,42	59	0,05	6
d150m60	13611261	**13611262**	**1**	**0,00**	4,70	45	0,11	6
d200m40	6207580	6207580	0	0,00	4,62	84	0,20	6
d200m80	24133320	**24133321**	**1**	**0,00**	14,24	48	0,37	6
d250m50	9685430	9685430	0	0,00	13,10	824	5,49	4
d250m100	37753120	37753118	-2	0,00	33,81	17	0,33	6

Tabella 4.6: Risultati ottenuti con la versione Tabu Long Term (**TLT**) sulle istanze di Andrade [2]. U.M. indica l'ultimo miglioramento della z^*.

4.2.4 Confronto tra gli algoritmi

La tabella 4.7 mostra i confronti delle versioni di Tabu Search sulle istanze di Silva [16]. Le colonne indicano: il tipo di istanza (Istanza), i risultati di Silva in termini di z^* e z media, seguono i risultati delle versioni del nostro algoritmo: z^* indica la migliore $z(s)$ trovata, Δz è la differenza tra i nostri risultati e quelli di Silva, il gap il percentuale (gap %) è calcolato come $\frac{\Delta z}{z^*(Silva)}100$. L'ultima colonna indica i nuovi Best Known evidenziando in grassetto quelli ottenuti con il nostro algoritmo. Se prendiamo di volta in volta la migliore versione, troviamo 6 nuovi Best Known e peggioriamo una sola istanza (matrizn500m150). I risultati ci portano a concludere che si tratta di un'istanza difficile per il quale l'algoritmo non riesce a raggiungere il risultato migliore di Silva. Sulle rimanenti 13 istanze raggiungiamo gli stessi risultati in tempi minori di 2-3 ordini di grandezza.

La tabella 4.8 mostra il confronto sui tempi tra le versioni del nostro algoritmo e i risultati pubblicati da Silva [16]. I risultati mostrano che tra tutte le versioni, l'algoritmo TST trova i risultati in media nel minor numero di iterazioni (502) mentre la versione TLT ne ha bisogno in media di 670. Questo comportamento è dovuto alla presenza della Long Term Memory che permette di trovare soluzioni miglioranti dopo aver fatto qualche *restart*. Sebbene i risultati dei confronti sono stati ottenuti su computer diversi, anche se con un'architettura molto simile, si osserva che la differenza tra il nostro P4 **Mobile** 2.8 GHz ed il PC AMD 1.3 GHz di Silva non giustifica la differenza nei tempi di calcolo di 2 o 3 ordini di grandezza, come riportato in tabella 4.7.

La tabella 4.9 confronta i risultati ottenuti sulle istanze proposte da Andrade in [2] in termini di funzione obiettivo. Le intestazioni delle colonne sono analoghe alla tabella 4.7 commentata precedentemente. Il nostro algoritmo trova 5 nuovi Best Known (in grassetto), per 32 istanze sono raggiunti i migliori risultati conosciuti, solo in 3 istanze sono peggiorati.

La tabella 4.10 mostra il tempo di calcolo delle versioni di Tabu Search sulle istanze proposte da Andrade [2]. La prima colonna indica il tipo di istanza, per ogni versione di algoritmo (T2L, TST, TLT) si riportano il tempo totale necessario per eseguire il test, l'iterazione di ultimo miglioramento e il relativo tempo. I r, imponendo che tutte le soluzioni di tale intorno siano ottenuteisultati mostrano che la versione T2L ha una velocità di convergenza media (264 iterazioni) migliore delle versioni TST (441) e TLT (437).

Per quanto riguarda i tempi di calcolo, le tre versioni sono molto vicine nelle istanze di piccole dimensioni mentre in quelle di dimensioni maggiori T2L è più veloce del TLT.

	Silva		**T2L**	**Gap** z^*		**TST**	**Gap** z^*		**TLT**	**Gap** z^*		**New**
Istanza	z^*	z ***avg***	z^*	Δz	**%**	z^*	Δz	**%**	z^*	Δz	**%**	z_{best}
matrizn100m10	333	333	333	0	0,00	333	0	0,00	333	0	0,00	333
matrizn100m20	1195	1195	1195	0	0,00	1195	0	0,00	1195	0	0,00	1195
matrizn100m30	2457	2457	2457	0	0,00	2457	0	0,00	2457	0	0,00	2457
matrizn100m40	4142	4142	4142	0	0,00	4142	0	0,00	4142	0	0,00	4142
matrizn200m20	1247	1247	1247	0	0,00	1241	-6	-0,48	1247	0	0,00	1247
matrizn200m40	4450	4449,3	4450	0	0,00	4448	-2	-0,04	4450	0	0,00	4450
matrizn200m60	9437	9437	9437	0	0,00	9437	0	0,00	9437	0	0,00	9437
matrizn200m80	16225	16225	16225	0	0,00	16225	0	0,00	16225	0	0,00	16225
matrizn300m30	2694	2692	2694	0	0,00	2694	0	0,00	2694	0	0,00	2694
matrizn300m60	9684	9682,5	**9689**	**5**	**0,05**	**9689**	**5**	**0,05**	**9689**	**5**	**0,05**	***9689***
matrizn300m90	20734	20727	**20743**	**9**	**0,04**	**20743**	**9**	**0,04**	**20743**	**9**	**0,04**	***20743***
matrizn300m120	35881	35881	35875	-6	-0,02	35881	0	0,00	35881	0	0,00	35881
matrizn400m40	4658	4654,7	4658	0	0,00	4651	-7	-0,15	4651	-7	-0,15	4658
matrizn400m80	16956	16946,7	16931	-25	-0,15	16956	0	0,00	16935	-21	-0,12	16956
matrizn400m120	36315	36301	**36317**	**2**	**0,01**	**36317**	**2**	**0,01**	**36317**	**2**	**0,01**	***36317***
matrizn400m160	62483	62478	**62487**	**4**	**0,01**	62484	1	0,00	**62487**	**4**	**0,01**	***62487***
matrizn500m50	7130	7124	**7141**	**11**	**0,15**	7129	-1	-0,01	**7141**	**11**	**0,15**	***7141***
matrizn500m100	26237	26229	**26258**	**21**	**0,08**	**26258**	**21**	**0,08**	**26258**	**21**	**0,08**	***26258***
matrizn500m150	58605	57055,7	56572	-2.033	-3,47	56572	-2.033	-3,47	56572	-2.033	-3,47	58605
matrizn500m200	97344	97333	97344	0	0,00	97344	0	0,00	97344	0	0,00	97344

Tabella 4.7: Confronto delle euristiche sulle istanze di Silva [16] (**z**). U.M. indica l'ultimo miglioramento della z^*.

Istanza	**Silva: *time* (s)**		**T2L**		**U.M.**		**TST**		**U.M.**		**TLT**		**U.M.**	
	z ***avg***	z^*	$time\ (s)$	***Iteraz.***	$time\ (s)$		$time\ (s)$	***Iteraz.***	$time\ (s)$		$time\ (s)$	**Iteraz.**	$time\ (s)$	
matrizn100m10	0,40	0,40	0,05	34	0,00		0,06	90	0,01		0,10	90	0,01	
matrizn100m20	28,20	27,40	0,21	76	0,02		0,23	203	0,04		0,37	203	0,04	
matrizn100m30	57,40	56,90	0,44	119	0,05		0,41	53	0,02		0,75	53	0,02	
matrizn100m40	89,20	88,30	0,70	103	0,06		0,70	113	0,07		1,21	113	0,07	
matrizn200m20	71,70	71,00	0,73	66	0,07		0,61	26	0,04		1,33	874	0,58	
matrizn200m40	510,30	507,30	4,35	576	1,57		3,08	191	0,49		5,35	1.045	2,81	
matrizn200m60	626,60	628,60	6,25	139	0,79		6,44	196	1,12		11,20	196	1,15	
matrizn200m80	1647,00	1638,60	10,37	272	2,18		10,39	235	1,97		17,39	235	2,00	
matrizn300m30	1405,30	1402,30	10,60	590	3,90		8,65	216	1,54		13,09	216	1,42	
matrizn300m60	3450,40	3613,17	37,19	830	16,71		34,36	629	12,99		39,40	629	12,41	
matrizn300m90	10514,60	10420,90	51,85	552	18,87		44,97	293	10,34		69,89	293	10,27	
matrizn300m120	17517,72	17562,01	79,56	715	33,17		84,62	789	37,45		93,18	1.147	53,32	
matrizn400m40	3371,00	3370,20	44,50	1342	30,22		35,99	596	13,83		45,20	596	13,64	
matrizn400m80	10048,60	10161,60	94,36	507	32,02		127,04	1471	93,27		128,28	1.531	97,84	
matrizn400m120	19201,30	19220,90	168,47	607	62,26		194,15	868	88,77		208,22	1.649	171,53	
matrizn400m160	32843,60	32550,50	255,48	747	107,81		247,10	700	100,61		295,32	1.390	204,76	
matrizn500m50	2934,90	2935,60	91,04	1171	53,78		93,11	1720	80,25		94,33	1.473	69,84	
matrizn500m100	26513,30	26473,10	259,61	884	119,55		186,36	361	48,54		272,45	361	49,44	
matrizn500m150	29051,40	27546,00	239,80	102	21,65		294,64	335	72,89		446,43	335	73,99	
matrizn500m200	86247,00	86513,80	640,29	1292	413,20		614,86	968	304,41		627,11	968	307,78	

Tabella 4.8: Confronto delle euristiche sulle istanze di Silva [16](***CPU time***). U.M. indica l'ultimo miglioramento della z^*.

	Andrade	**T2L**	**Gap** z^*		**TST**	**Gap** z^*		**TLT**	**Gap** z^*		**New**
Istanza	z^*	z^*	Δz	**%**	z^*	Δz	**%**	z^*	Δz	**%**	z_{best}
a050m10	492	492	0	0,00	492	0	0,00	492	0	0,00	492
a050m20	1932	1932	0	0,00	1932	0	0,00	1932	0	0,00	1932
a100m20	2007	2007	0	0,00	2007	0	0,00	2007	0	0,00	2007
a100m40	7730	7730	0	0,00	7730	0	0,00	7730	0	0,00	7730
a150m30	4552	4552	0	0,00	4552	0	0,00	4552	0	0,00	4552
a150m60	17482	17482	0	0,00	17482	0	0,00	17482	0	0,00	17482
a200m40	8132	8132	0	0,00	8132	0	0,00	8132	0	0,00	8132
a200m80	31049	31049	0	0,00	31049	0	0,00	31049	0	0,00	31049
a250m50	12653	**12654**	**1**	**0,01**	**12654**	**1**	**0,01**	**12654**	**1**	**0,01**	**12654**
a250m100	48384	48384	0	0,00	48384	0	0,00	48384	0	0,00	48384
b050m10	334976	334976	0	0,00	334976	0	0,00	334976	0	0,00	334976
b050m20	1171416	1171416	0	0,00	1171416	0	0,00	1171416	0	0,00	1171416
b100m20	1267277	1267277	0	0,00	1267277	0	0,00	1267277	0	0,00	1267277
b100m40	4544642	4544642	0	0,00	4544642	0	0,00	4544642	0	0,00	4544642
b150m30	2758381	2758381	0	0,00	2758381	0	0,00	2758381	0	0,00	2758381
b150m60	9960461	9960461	0	0,00	9959120	-1.341	-0,01	9960461	0	0,00	9960461
b200m40	4788086	4788086	0	0,00	4788086	0	0,00	4788086	0	0,00	4788086
b200m80	17544448	17544447	-1	0,00	17544447	-1	0,00	17544447	-1	0,00	17544448
b250m50	7388997	7378891	-10.106	-0,14	7379797	-9.200	-0,12	7379278	-9719	-0,13	7388997
b250m100	27162906	**27168460**	**5.554**	**0,02**	**27168460**	**5.554**	**0,02**	**27168460**	**5554**	**0,02**	**27168460**
c050m10	316409	316409	0	0,00	316409	0	0,00	316409	0	0,00	316409
c050m20	1094343	1094343	0	0,00	1094343	0	0,00	1094343	0	0,00	1094343
c100m20	1205722	**1207522**	**1.800**	**0,15**	**1207522**	**1.800**	**0,15**	**1207522**	**1800**	**0,15**	**1207522**
c100m40	4219476	4219476	0	0,00	4219476	0	0,00	4219476	0	0,00	4219476
c150m30	2613286	2613286	0	0,00	2613286	0	0,00	2613286	0	0,00	2613286
c150m60	9374611	9374611	0	0,00	9374611	0	0,00	9374611	0	0,00	9374611
c200m40	4630545	4630545	0	0,00	4630545	0	0,00	4630545	0	0,00	4630545
c200m80	16759895	16759895	0	0,00	16759895	0	0,00	16759895	0	0,00	16759895
c250m50	7178043	7178043	0	0,00	7178043	0	0,00	7178043	0	0,00	7178043
c250m100	26047022	26047022	0	0,00	26047022	0	0,00	26047022	0	0,00	26047022
d050m10	381379	381379	0	0,00	381379	0	0,00	381379	0	0,00	381379
d050m20	1502908	1502908	0	0,00	1502908	0	0,00	1502908	0	0,00	1502908
d100m20	1570800	1570800	0	0,00	1570800	0	0,00	1570800	0	0,00	1570800
d100m40	6067776	6067776	0	0,00	6067776	0	0,00	6067776	0	0,00	6067776
d150m30	3502567	3502567	0	0,00	3502567	0	0,00	3502567	0	0,00	3502567
d150m60	13611261	**13611262**	**1**	**0,00**	**13611262**	**1**	**0,00**	**13611262**	**1**	**0,00**	**13611262**
d200m40	6207580	6207580	0	0,00	6207580	0	0,00	6207580	0	0,00	6207580
d200m80	24133320	**24133321**	**1**	**0,00**	**24133321**	**1**	**0,00**	**24133321**	**1**	**0,00**	**24133321**
d250m50	9685430	9685430	0	0,00	9685430	0	0,00	9685430	0	0,00	9685430
d250m100	37753120	37753118	-2	0,00	37753118	-2	0,00	37753118	-2	0,00	37753120

Tabella 4.9: Confronto delle euristiche sulle istanze di Andrade [2] (**z**). U.M. indica l'ultimo miglioramento della z^*.

	T2L	**U.M.**		**TST**	**U.M.**		**TLT**	**U.M.**	
Istanza	*time* ***(s)***	***Iteraz.***	*time* ***(s)***	*time* ***(s)***	***Iteraz.***	*time* ***(s)***	*time* ***(s)***	**Iteraz.**	*time* ***(s)***
a050m10	0,04	909	0,02	0,04	1996	0,04	0,04	1996	0,04
a050m20	0,13	1982	0,13	0,13	1998	0,13	0,12	1372	0,08
a100m20	0,23	158	0,04	0,40	1588	0,31	0,39	522	0,10
a100m40	0,72	168	0,11	1,21	1995	1,21	1,19	1237	0,74
a150m30	1,63	890	0,77	1,50	851	0,70	1,65	1200	0,99
a150m60	3,17	0	0,00	3,16	0	0,00	6,29	0	0,00
a200m40	2,97	4	0,04	5,23	879	2,38	5,96	1510	4,50
a200m80	18,82	1998	18,81	18,81	1998	18,80	19,28	1998	19,27
a250m50	7,55	0	0,00	9,13	0	0,00	14,94	0	0,00
a250m100	24,32	115	2,54	37,91	624	14,62	45,30	1999	45,30
b050m10	0,02	0	0,00	0,02	0	0,00	0,04	0	0,00
b050m20	0,07	1	0,00	0,06	1	0,00	0,12	1	0,00
b100m20	0,21	36	0,01	0,21	33	0,01	0,37	33	0,01
b100m40	0,67	67	0,04	0,88	404	0,25	1,17	384	0,22
b150m30	1,11	266	0,25	1,41	642	0,56	1,76	1594	1,41
b150m60	3,67	206	0,64	6,11	1186	3,60	6,14	705	2,19
b200m40	4,56	495	1,50	4,37	480	1,37	5,78	480	1,30
b200m80	10,83	103	0,99	10,47	82	0,79	18,90	82	0,78
b250m50	14,54	836	6,69	19,57	1164	11,31	15,99	708	5,59
b250m100	22,87	27	0,63	26,66	155	3,54	44,61	155	3,37
c050m10	0,02	7	0,00	0,02	7	0,00	0,04	7	0,00
c050m20	0,06	1	0,00	0,06	1	0,00	0,11	1	0,00
c100m20	0,18	2	0,00	0,18	2	0,00	0,34	2	0,00
c100m40	0,53	11	0,01	0,52	11	0,01	1,03	11	0,01
c150m30	0,72	16	0,03	0,69	16	0,02	1,34	16	0,02
c150m60	2,00	12	0,03	2,03	12	0,04	3,90	12	0,04
c200m40	2,59	20	0,07	2,43	20	0,08	4,73	20	0,08
c200m80	9,42	219	1,74	8,47	86	0,69	14,68	86	0,65
c250m50	7,30	283	1,63	6,31	73	0,48	11,12	73	0,46
c250m100	20,62	285	4,62	18,72	71	1,28	32,96	71	1,22
d050m10	0,02	90	0,00	0,02	43	0,00	0,04	43	0,00
d050m20	0,06	7	0,00	0,06	7	0,00	0,12	7	0,00
d100m20	0,24	251	0,05	0,20	37	0,01	0,36	37	0,01
d100m40	0,64	170	0,09	0,60	42	0,03	1,05	42	0,02
d150m30	0,82	59	0,06	0,80	59	0,05	1,42	59	0,05
d150m60	2,59	36	0,10	2,61	45	0,13	4,70	45	0,11
d200m40	2,29	46	0,12	2,51	84	0,23	4,62	84	0,20
d200m80	8,00	141	1,02	7,28	48	0,36	14,24	48	0,37
d250m50	9,99	637	3,92	12,59	896	6,00	13,10	824	5,49
d250m100	16,72	17	0,32	17,29	17	0,33	33,81	17	0,33

Tabella 4.10: Confronto delle euristiche sulle istanze di Andrade [2] (***CPU time***). U.M. indica l'ultimo miglioramento della z^*.

Capitolo 5

Conclusioni

La prima parte del lavoro si è concentrata sull'analisi della letteratura scoprendo che quasi tutte le euristiche proposte per affrontare MDP sono varianti del GRASP [1, 2, 6, 16]. Le nostre idee si sono focalizzate sullo sviluppo di un algoritmo di Tabu Search, per cercare di migliorare i risultati pubblicati. Nella seconda fase del lavori si sono alternati momenti di implementazione a momenti di analisi dei risultati per definire di volta in volta le strategie di miglioramento. Sin dalle prime versioni dell'algoritmo si è visto che il Tabu Search è un approccio efficace alla ricerca delle soluzioni di MDP. I test ci hanno permesso di determinare 11 nuovi Best Known e di raggiungere il miglior risultato noto per 45 istanze su 60. Nonostante lo stress test, l'algoritmo di Tabu Search non è riuscito a raggiungere il valore Best Known relativo a 4 istanze su di 60. Questi risultati sono stati ottenuti con un tempo di calcolo minore di almeno due ordini di grandezza rispetto a quelli riportati in letteratura. Concludiamo che per risolvere MDP, i metodi basati sul Tabu Search sono più efficienti rispetto a quelli basati su euristiche GRASP.

Si è visto che il rischio dell'approccio Tabu Search consiste nel pericolo di incorrere in una sequenza di soluzioni che determina un ciclo. Per limitare ulteriormente questo rischio si potrebbero realizzare delle Tabu List con **hash** per individuare cicli molto lunghi. Un modo di realizzarle consiste nel mantenere una lista, contenente il valore della funzione Hash calcolata sulle soluzioni trovate nell'esplorazione dell'intorno. Una soluzione vine considerata Tabu se il suo hash compare in tale lista.

La memoria a lungo termine è stata introdotta nella versione TLT per intensificare la ricerca nelle aree dello spazio promettenti. La seconda migliore soluzione è determinata sulla base della bontà delle soluzioni presenti nella lista S. E' possibile che ripristinando una soluzione di minor valore si possano raggiungere ottimi locali migliori. Per una più accurata scelta della seconda soluzione, si potrebbe estrarre dall'insieme S della Long Term Memory le soluzioni, non per il valore z associato alla soluzione, ma sulla base del valore ottenuto applicando una breve ricerca locale: al prezzo di un piccolo aggravio del peso computazionale potremmo avere una migliore scelta della seconda soluzione.

Al fine di sfruttare l'idea contenuta nelle euristiche di tipo GRASP, si potrebbe pro-

vare a sostituire l'algoritmo (Greedy) che costruisce la soluzione iniziale ammissibile, con una delle versioni riportate nella sezione 2.4. Come ulteriore sviluppo futuro, si potrebbe provare a sostituire l'intorno di ricerca locale, alla base dell'algoritmo di Tabu Search, con uno più vasto, ottenuto rimpiazzando una coppia di elementi della soluzione, con una coppia non appartenente, come mostrato in 2.5.2.

Appendice A

Modellazione di MDP con linguaggio AMPL

I *generatori algebrici di modelli*, fra cui **AMPL** [15] , costituiscono un'interfaccia verso il risolutore, ovvero un software in grado di gestire modelli e dati reali per fornire una soluzione. Le caratteristiche principali dei *generatori algebrici di modelli* sono le seguenti:

- fornire un linguaggio semplice per descrivere modelli complessi, un linguaggio che sia contemporaneamente ad *alto livello* e *formalmente strutturato* cioè accessibile ad un risolutore;
- permettere all'utente di comunicare con il risolutore attraverso file di testo anziché mediante strutture dati, in modo da non richiedergli conoscenze informatiche approfondite e da poter formulare il modello con un semplice editor di testo, svincolato dalla piattaforma sulla quale poi verrà risolto.
- permettere all'utente di comunicare con diversi risolutori, in modo da poter sfruttare i più potenti disponibili sul mercato;
- tenere distinti la *struttura logica* del modello (variabili di decisione, obiettivi, vincoli e le loro relazioni) dal valore di *dati numerici.*

Sebbene sia lecito scrivere in un solo file **AMPL** sia il modello sia i dati, è concettualmente preferibile tenere separati questi due termini, costruendo:

- un *file di modello*, obbligatoriamente di estensione **.mod**, che descrive la struttura logica del modello, (indici, variabili di decisione, funzione obiettivo e vincoli)
- un *file di dati*, obbligatoriamente di estensione **.dat**, che contiene l'istanza del problema

param n;	Dimensione della popolazione
param m;	Numero oggetti da scegliere
set Oggetti := 1..n;	Insieme degli oggetti
param D {Oggetti,Oggetti};	Matrice delle distanze
var x {Oggetti} binary;	Variabili decisionali x
var y {Oggetti,Oggetti} binary;	Variabili decisionali y
maximize z : (sum{i in Oggetti, j in Oggetti} D[i,j] * y[i,j]);	Funzione obiettivo
subject to Cardinalita : sum{i in Oggetti} x[i] = m;	Vincolo di cardinalità
subject to Simmetria {i in Oggetti, j in Oggetti} : y[i,j] = y[j,i];	Vincolo di simmetria
subject to AttivazioneI {i in Oggetti, j in Oggetti} : y[i,j] <= x[i];	Vincolo di attivazione per i
subject to AttivazioneJ {i in Oggetti, j in Oggetti} : y[i,j] <= x[j];	Vincolo di attivazione per j
subject to Taglio{i in Oggetti}:sum{j in Oggetti} y [i,j]=(m-1)*x[i];	Vincolo di taglio
end;	Fine del file di modello

Tabella A.1: File del modello *.mod*

Mantenendo fisicamente separato il modello dai dati, è possibile applicare il modello per la risoluzione di istanze differenti.

Il software **AMPL** è in gado di tradurre modelli di Programmazione Matematica scritti in linguaggio AMPL in un formato comprensibile a un generico risolutore di programmazione matematica.

A.1 Il file del modello *.mod*

Il file del modello (Tabella A.1) descrive gli indici, le variabili di decisione, funzione obiettivo e i vincoli traducendo in linguaggio AMPL il modello descritto nella sezione 2.1.1.

I parametri n, m sono rispettivamente la cardinalità dell'insieme N e la cardinalità dell'insieme M. Si definisccono un insieme di oggetti, la matrice delle distanze D, le variabili decisionali x_i e y_{ij}, la funzione obiettivo z e i vincoli nell'ordine: di cardinalità, di simmetria, di attivazione (per i e per j) e di taglio (per rafforzare la formulazione).

A.2 Il file dei dati *.dat*

L'istanza del problema è contenuta nel file *.dat* (tabella A.2) e contiene il valore dei parametri specificati nel file di modello cioè la dimensione della popolazione, il numero di oggetti che costituiscono la soluzione e la matrice delle distanze.

La distanza è espressa nella forma $[i, j]\ d_{ij}$. I valori presenti sono stati messi a titolo di esempio e costituiscono un problema di piccole dimensioni $(n < 30)$.

param n := 7;	Numero oggetti della popolazione totale
param m := 4;	Numero oggetti scelti dalla popolazione
param D :=	Matrice delle distanze definita per ogni coppia di oggetti
[1,1] 0 [1,2] 3 [1,3] 5 [1,4] 1 [1,5] 7 [1,6] 4 [1,7] 3	
[2,1] 3 [2,2] 0 [2,3] 5 [2,4] 2 [2,5] 6 [2,6] 2 [2,7] 2	
[3,1] 5 [3,2] 5 [3,3] 0 [3,4] 4 [3,5] 5 [3,6] 5 [3,7] 6	
[4,1] 1 [4,2] 2 [4,3] 4 [4,4] 0 [4,5] 4 [4,6] 2 [4,7] 1	
[5,1] 7 [5,2] 6 [5,3] 5 [5,4] 4 [5,5] 0 [5,6] 6 [5,7] 2	
[6,1] 4 [6,2] 2 [6,3] 5 [6,4] 2 [6,5] 6 [6,6] 0 [6,7] 6	
[7,1] 3 [7,2] 2 [7,3] 6 [7,4] 1 [7,5] 2 [7,6] 6 [7,7] 0;	
end;	Fine dell'istanza

Tabella A.2: File dei dati *.dat*

Bibliografia

[1] Andrade, P. M. D., Plastino, A., Ochi, L. S., Martins, S. L.: "GRASP for the Maximum Diversity Problem", Proceedings of Metaheuristics International Conference 2003, (2003).

[2] Andrade, P. M. D., Plastino, L. S., Martins, S. L.: "GRASP with Path-Relinking for the Maximum Diversity Problem", Workshop on Efficient Algorithms 2005.

[3] Kuo, C.C, F. Glover and K.S. Dhir: "Analyzing and modeling the maximum diversity problem by zero-one programming", Dec. Sci. 24 (1993), 1171-1185.

[4] F. Glover, Kuo, C.C, S. Dhir Krishna: "A discrete optimization model for preserving biological diversity", Appl. Math. Modelling 1995, Vol. 19, November.

[5] Ghosh, J. B.: "Computational aspects of maximum diversity problem", Operation Research Letters 19 (1996), 175-181.

[6] Glover, F., Laguna, M. Marti R.: "Fundamentals of scatter search and path-relinking", Control and Cybernetics, 19, (1997), 653-684.

[7] Glover, F. Kuo, C-C., Dhir, K. S.: "Integer programming and heuristic approaches to the minimum diversity problem", Journal of Business and Management 4(1) (1996), 93-111.

[8] Glover F., Laguna M.: "Tabu Search", Kluwer Academic Publishers (1997).

[9] ITCS Vittorio Bachelet - Roma: "Progetto di educazione ambientale", http://www.ips.it/scuola/concorso/bachelet/biodiver.htm, Roma, 21 marzo 1998.

[10] Kochenberger, G., Glover, F.: "Diversity data mining", Working Paper Series, The University of Mississipi, (1999).

[11] M. Dell'Amico, M. Trubian, "Solution of large weighted equicut problems", European Jurnal of Operational Research 106(1998) 500-521.

[12] P.M. Pardalos and G.P. Rodgers, "Computational Aspects of branch and bound algorithm for quadratic zero-one programming", Computing 45 (1990), 131-144 .

[13] Prais, M., Ribeiro, C. C.: "Reactive GRASP: an application to a matrix decomposition problem in TDMA traffic assignment", INFORMS Journal on Computing 12 (2000), 164-176.

[14] Resende, M. G. C., Ribeiro, C.C.: "GRASP with path-relinking: Recent advances and applications, Metaheuristics: Progress as Real Problem Solvers", (T. Ibaraki, K. Nonobe e M. Yagiura, eds.), (2005), 29-63.

[15] Cordone R.: "Appunti sul linguaggio di programmazione AMPL", Milano, Ottobre 2001.

[16] Silva, G.C., Ochi, L.S., Martins, S.L.: "Experimental comparison of greedy randomized adaptive search procedure for the maximum diversity problem", Lectures Notes on Computer Science 3059 (2004), 498-512.

[17] Weitz, R., Lakshminarayanan, S.: "An empirical comparison of heuristic methods for creating maximally diverse group", Journal of the operational Research Society 49 (1998), 635-646.