UNIVERSITÀ DEGLI STUDI DI MILANO
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica

Algoritmi euristici per il "multi-vehicle pick-up and delivery problem with Rear-Loading constraints"

Relatore
Prof. Giovanni RIGHINI

Tesi di Laurea di
Giulio Moretti
Matr. 643288

Anno Accademico 2004/2005

Al Prof. Giovanni Righini per l'aiuto

durante lo sviluppo della tesi.

Ai miei genitori, grazie ai quali ho

potuto raggiungere questo traguardo.

Indice

Introduzione

Negli ultimi decenni si è verificato un crescente utilizzo di pacchetti software basati su tecniche di ricerca operativa e programmazione matematica per la gestione efficiente della fornitura di beni e servizi nei sistemi di distribuzione.

Il grande numero di applicazioni reali, sia nel Nord America sia in Europa, ha ampiamente dimostrato che l'utilizzo di software per la pianificazione dei processi di distribuzione produce un sostanziale risparmio (generalmente dal 5% al 20%) nei costi globali di trasporto. E' facile osservare come l'impatto di questo risparmio sul sistema economico di un Paese sia significativo dal momento che i processi di trasporto riguardano tutte le fasi della produzione dei beni ed i costi relativi rappresentano una componente rilevante (generalmente dal 10% al 20%) del costo finale.

Il successo nell'utilizzo di tecniche di ricerca operativa è dovuto non solo allo sviluppo hardware e software nel campo dell'informatica e alla crescente integrazione dei sistemi informativi nel processo produttivo ed in quello commerciale ma soprattutto allo sviluppo di nuovi modelli che cercano di prendere in considerazione tutte le caratteristiche dei problemi reali ed alla concezione di nuovi algoritmi che permettono di trovare buone soluzioni in tempi di calcolo accettabili.

Il Capacitated Vehicle Routing Problem (CVRP) consiste nell'ottimizzare i percorsi di un insieme di veicoli di una data capacità che devono consegnare merce a un insieme di clienti viaggiando lungo la rete di trasporto esistente. In [4] Savelsbergh descrive algoritmi euristici per il VRP with Time Windows in cui il prelievo della merce e di consegna del materiale di scarto deve essere effettuato in un intervallo i tempo chiamato appunto time windows; in [5] si considera infine il VRP with Pick-up and Delivery (VRPPD) in cui i punti di prelievo della merce e di consegna del materiale di scarto non coincidono necessariamente con il deposito.

In questa tesi è preso in esame il Pick-up Delivery Problem with rear-loading (PDPRL) in cui i veicoli devono trasportare oggetti da clienti-origine a clienti-destinazione, ed è presente un vincolo del rear-loading, cioè ogni veicolo funziona come uno stack che può quindi scaricare solo l'ultimo oggetto che ha caricato.

Nel capitolo 1 è descritto il problema con una panoramica sui metodi risolutivi, nei capitoli successivi vengono presentati nuovi algoritmi di approssimazione che possono essere utilizzati per risolvere il problema**.** Nel capitolo 2 sono descritte le euristiche di costruzione del tuor iniziale con una relativa analisi dei risultati sperimentali. Nel capitolo 3 si definiscono intorni per la costruzione di efficienti algoritmi di ricerca locale; il loro funzionamento si basa sull'idea di migliorare una soluzione ammissibile eseguendo una sequenza di scambi di archi o spigoli oppure di vertici o nodi all'interno del singolo percorso o tra percorsi diversi. Nel capitolo 4 viene infine presentato un tabu search basato sugli intorni definiti nel capitolo precedente. Il tabu search riceve in ingresso una soluzione ammissibile x_1, passa iterativamente dalla soluzione x_t alla soluzione x_{t+1} che

appartiene all'intorno $N(x_t)$ e termina al verificarsi di determinate condizioni; se $f(x)$ rappresenta il costo di x, allora $f(x_{t+1})$ non necessariamente è minore di $f(x_t)$, di conseguenza occorre prestare particolare attenzione alla possibilità che l'algoritmo cicli su un insieme di soluzioni. I risultati dei test svolti mostrano come il tabu search sia l'algoritmo che garantisce il calcolo delle migliori soluzioni a fronte di tempi di calcolo accettabili.

E' presente anche un appendice contenente una spiegazione dettagliata dei formati dei files implementati e del loro uso.

Capitolo 1

Descrizione del problema

1.1 Pick-up Delivery Problem con Rear Loading

I problemi inerenti la distribuzione di beni materiali tra un deposito o un insieme di depositi e i clienti sono generalmente noti con il nome di Vehicle Routing Problems (VRPs) o Vehicle Scheduling Problems.

Distribuire beni significa servire un insieme di clienti grazie a veicoli che sono localizzati in uno o più depositi e che effettuano i loro spostamenti utilizzando la rete stradale esistente. La soluzione di un VRP è quindi rappresentata da un insieme di percorsi che iniziano e terminano nei depositi, ognuno dei quali deve essere effettuato da un unico veicolo. Tali percorsi devono essere determinati in maniera tale che tutte le richieste dei clienti siano esaudite, tutti i vincoli operazionali siano soddisfatti ed il costo globale del trasporto sia minimizzato.

Prima di cominciare la descrizione del problema occorre definire due termini che nel corso della trattazione verranno usati: il termine cliente e il termine nodo. Con il termine

cliente si intende una coppia di punti O_i e D_i (posizioni identificate geograficamente) detti nodi; di cui uno definito come nodo-origine di raccolta mentre l'altro definito come nodo-destinazione di scarico della quantità q_i raccolta.

In questa tesi è preso in esame il Pick-up Delivery Problem con Rear Loading (PDPRL), un problema meno conosciuto del più famoso e studiato VRPPD.

Per questo problema ad ogni cliente i è associata un quantità q_i, ed ogni veicolo parte e torna vuoto dal deposito perché deve solo trasportare oggetti da nodi-origine a nodi-destinazione. Quindi nel PDPRL si assume che durante la visita a un cliente venga visitato per prima il nodo-origine e successivamente il nodo-destinazione.

Risolvere il PDPRL consiste nel trovare un insieme di V percorsi aventi costo minimo (definito come la lunghezza del percorso o il tempo necessario per effettuarlo) che soddisfano i seguenti vincoli:

- ogni veicolo durante il suo percorso deve partire da e tornare al deposito (indicato con 0);
- ogni cliente deve essere servito da un unico veicolo;
- il carico corrente del veicolo lungo il percorso non può eccedere la capacità Q del veicolo;
- per ogni cliente i, il suo nodo-origine O_i, deve essere visitato all'interno dello stesso percorso e prima del suo nodo-destinazione Di;

Il problema PDPRL è un problema di ottimizzazione NP-hard in senso forte [1].

1.2 Formulazione del PDPRL

In questa versione del PDPRL si assume che la domanda di un cliente sia indivisibile, ossia deve essere soddisfatta in un'unica visita.

Dati:

- una flotta di V veicoli con identica capacità Q;
- un insieme di M clienti;
- un insieme di N nodi suddivisi in due sottoinsiemi disgiunti O e D, rispettivamente l'insieme dei nodi-origine (*pick-up demand*) e l'insieme dei nodi-destinazione (*delivery demand*) (definendo $N_o = |O|$ e $N_d = |D|$ si ha che $N = N_o + N_d$);
- q_i è la quantità da raccogliere e scaricare del cliente i ($i \in M$) ;
- c_{uw} rappresenta la distanza da percorrere per andare da qualsiasi nodo u a qualsiasi nodo w, con $u,w \in \{0..N\}$.

Variabili:

- x^+_{ij} è associata all'arco (O_i,O_j) e vale 1 se e solo se un veicolo transita dal nodo origine del cliente i al nodo origine del cliente j, con $i,j \in M$;
- x^-_{ij} è associata all'arco (D_i,D_j) e vale 1 se e solo se un veicolo transita dal nodo destinazione del cliente i al nodo destinazione del cliente j, con $i,j \in M$;
- x_{ij} è associata all'arco (D_i,O_j) e vale 1 se e solo se un veicolo transita dal nodo destinazione del cliente i al nodo origine del cliente j, con $i,j \in M$;
- x_{0i} è associata all'arco $(0,O_i)$ e vale 1 se e solo se un veicolo transita dal deposito al

nodo origine del cliente i, con $i \in M$;

- x_{j0} è associata all'arco $(D_j,0)$ e vale 1 se e solo se un veicolo transita dal nodo destinazione del cliente j al deposito , con $j \in M$;
- k_i è associata all'arco (O_i,D_i) e vale 1 se e solo se un veicolo transita dal nodo origine del cliente i direttamente al nodo destinazione del cliente i, con $i \in M$;
- y_{uw} è associata all'arco (u,w) e il suo valore indica l'ammontare del carico presente sul veicolo che viene trasportato da qualsiasi nodo u a qualsiasi nodo w, con $u,w \in N$;
- w_{ij} prende valore 1 se e solo se il cliente j e contenuto nel cliente i, il che significa che l'origine di i è visitata prima dell'origine di j e la destinazione di j è visitata prima della destinazione di i, con $i,j \in M$.
- p_{ij} prende valore 1 se e solo se il cliente i viene visitato prima del cliente j, il che significa che sia il nodo origine di i e sia il nodo destinazione di i sono visitati prima del nodo origine e del nodo destinazione di j, con $i,j \in M$.

Funzione obiettivo:

L'obiettivo è quello di minimizzare la lunghezza totale dei percorsi.

$$\min \sum_{i=1}^{M}\sum_{j=1}^{M} c_{Di,Oj}\cdot x_{ij} + \sum_{i=1}^{M}\sum_{j=1}^{M} c_{Oi,Oj}\cdot x^{+}_{ij} + \sum_{i=1}^{M}\sum_{j=1}^{M} c_{Di,Dj}\cdot x^{-}_{ij} + \sum_{j=0}^{M} c_{Oj,Dj}\cdot k_j +$$

$$+ \sum_{i=0}^{M} c_{0i}\cdot x_{0i} + \sum_{j=0}^{M} c_{j0}\cdot x_{j0} \qquad (1.1)$$

$$\text{s.t.} \quad \sum_{i=1}^{M} x_{ij} + \sum_{i=1}^{M} x^{+}_{ij} + x_{0j} = 1 \qquad \text{IN } O_j \qquad \forall\, j = 1..M \qquad (1.2a)$$

$$\sum_{i=1}^{M} x^{+}_{ji} + k_j = 1 \qquad \text{OUT } O_j \qquad \forall\, j = 1..M \qquad (1.2b)$$

$$\sum_{j=1}^{M} x^{-}_{ji} + k_i = 1 \qquad \text{IN } D_i \qquad \forall\, i = 1..M \qquad (1.3a)$$

$$\sum_{j=1}^{M} x_{ij} + \sum_{j=1}^{M} x^{-}_{ij} + x_{i0} = 1 \qquad \text{OUT } D_i \qquad \forall\, i = 1..M \qquad (1.3b)$$

$$\sum_{j=1}^{M} x_{0j} = V \qquad (1.4a)$$

$$\sum_{i=1}^{M} x_{i0} = V \qquad (1.4b)$$

$$\sum_{w=1}^{N} y_{u,w} - \sum_{w=1}^{N} y_{w,u} = \begin{cases} q_j & u \in O_j \\ -q_j & u \in D_j \\ 0 & u = 0 \end{cases} \qquad \forall\, j = 1..M \qquad (1.5)$$

$$y_{uw} \le Q\cdot x_{ij} \qquad \text{con } u = D_i \text{ e } w = O_j \qquad \forall\, i,j = 1..M \qquad (1.6a)$$

$$y_{uw} \le Q\cdot x^{+}_{ij} \qquad \text{con } u = O_i \text{ e } w = O_j \qquad \forall\, i,j = 1..M \qquad (1.6b)$$

$$y_{uw} \leq Q \cdot x^{-}_{ij} \qquad \text{con } u = D_i \text{ e } w = D_j \qquad \forall\, i,j = 1..M \qquad (1.6c)$$

$$y_{uw} \leq Q \cdot k_j \qquad \text{con } u = O_j \text{ e } w = D_j \qquad \forall\, j = 1..M \qquad (1.6d)$$

$$y_{uw} = 0 \qquad \text{con } u = D_j \text{ e } w = 0 \qquad \forall\, j = 1..M \qquad (1.6e)$$

$$y_{uw} = 0 \qquad \text{con } u = 0 \text{ e } w = O_j \qquad \forall\, j = 1..M \qquad (1.6f)$$

$$x^{+}_{ij} \leq w_{ij} \qquad \forall\, i,j = 1..M \qquad (1.7)$$

$$x^{-}_{ij} \leq w_{ij} \qquad \forall\, i,j = 1..M \qquad (1.8)$$

$$x_{jh} \leq p_{jh} \qquad \forall\, j,h = 1..M \qquad (1.9)$$

$$w_{ij} + w_{jh} \leq 1 + w_{ih} \qquad \forall\, i,j,h = 1..M \qquad (1.10)$$

$$w_{ij} + x_{jh} \leq 1 + w_{ih} \qquad \forall\, i,j,h = 1..M \qquad (1.11)$$

$$x_{jh} + w_{ih} \leq 1 + w_{ij} \qquad \forall\, i,j,h = 1..M \qquad (1.12)$$

$$p_{ij} + p_{jh} \leq 1 + p_{ih} \qquad \forall\, i,j,h = 1..M \qquad (1.13)$$

$$p_{ij} + w_{jh} \leq 1 + p_{ih} \qquad \forall\, i,j,h = 1..M \qquad (1.14)$$

$$w_{jh} + p_{ji} \leq 1 + p_{hi} \qquad \forall\, i,j,h = 1..M \qquad (1.15)$$

$$w_{ij} + w_{ji} + p_{ij} + p_{ji} = 1 \qquad \forall\, i,j = 1..M \qquad (1.16)$$

$$x_{ij},\ x^{+}_{ij},\ x^{-}_{ij},\ x_{0j},\ x_{i0},\ k_i,\ w_{ij},\ p_{ij} \in \{0,1\} \qquad \forall\, i,j = 1..M \qquad (1.17)$$

$$y_{uw} \in \{0,1\} \qquad \forall\, u,w = 1..N \qquad (1.18)$$

I vincoli (1.2a) (1.2b) e (1.3a) (1.3b) assicurano che ogni cliente venga visitato da un unico veicolo; i vincoli (1.4a) (1.4b) impongono che ogni veicolo durante il suo percorso parta dal e arrivi al deposito; il vincolo (1.5) è un vincoli di flusso; i vincoli dal (1.6a) al (1.6f) assicurano che il carico corrente del veicolo durante tutto il percorso non ecceda la capacità Q del veicolo stesso. I vincoli (1.7) e (1.8) forzano la variabile w ad assumere valore 1 ogni volta che due origini o due destinazioni sono visitate consecutivamente; il vincolo (1.9) forza la variabile p ad assumere valore 1 ogni volta che si passa da un nodo destinazioni a un nodo origine; il vincolo (1.10) propaga alla relazione di contenimento la proprietà transitiva: se i contiene j e j contiene k allora i contiene k. I vincoli (1.11) e (1.12) sono necessari per estendere la relazione di contenimento a tutti i clienti appartenenti a una sequenza che parte e finisce con un cliente contenitore. Il vincolo (1.13) propaga alla relazione di visitato prima la proprietà transitiva: se i è visitato prima di j e j è visitati prima di k allora i è visitato prima di k; i vincoli (1.14) e (1.15) sono necessari per estendere la relazione di visitato prima a tutti i clienti appartenenti a una sequenza che parte e finisce con un cliente;il vincolo (1.16) produce la politica LIFO e impedisce la presenza di sottocicli isolati.

Capitolo 2

Euristiche di costruzione

Le *euristiche costruttive* definiscono gradualmente una soluzione ammissibile tenendo in considerazione il costo della soluzione che si sta costruendo; ma esse non contengono una fase di miglioramento. Per costruire delle soluzioni per il PDPRL ho sperimentato due tecniche: la prima assegna gradualmente i vertici alle rotte dei veicoli usando un costo di inserimento, la seconda unisce delle rotte già esistenti attraverso dei criteri di "risparmio".

Sono stati sviluppati 5 algoritmi costruttivi, i primi quattro: Nearest Neighbour, Reversed Nearest Neighbour, Randomized Nearest Neighbour, Reversed Randomized Nearest Neighbour utilizzano la prima tecnica, mentre l'ultimo, l'algoritmo Clarke – Wright, utilizza il criterio del "risparmio".

Ognuno di questi algoritmi riceve in ingresso :

- le posizioni delle città nel piano bidimensionale;
- gli accoppiamenti tra esse;

e restituisce in uscita dei tour che rispettano i vincoli del rear-loading.

2.1 Algoritmo Nearest Neighbour

L'algoritmo inizialmente considera *V* cammini composti dal solo nodo deposito. Ad ogni iterazione si sceglie il nodo più vicino all'ultimo inserito tra quelli ammissibili, e lo si aggiunge al cammino che presenta il costo minore di inserimento (costo inteso come distanza tra il nodo scelto e l'ultimo inserito nel cammino) (fig. 2.1).

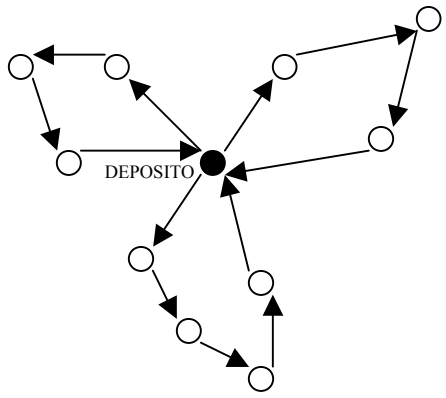

Figura 2.1 : algoritmo Nearest Neighbour

Per rispettare il vincolo di rear-loading è necessario controllare il tipo del nodo da aggiungere : un nodo pick-up può essere sempre aggiunto a qualsiasi cammino (tranne per il vincolo di capacità), mentre un nodo delivery può essere inserito solo se il proprio partner è l'ultimo nodo pick-up visitato del quale non è ancora stato raggiunto il relativo delivery (fig. 2.2). L'uso di stack permette di gestire con facilità questo vincolo: se si

incontra un nodo di tipo pick-up lo si inserisce con una procedura *Push*, altrimenti si elimina il nodo pick-up in cima al relativo stack con una procedura *Pop*.

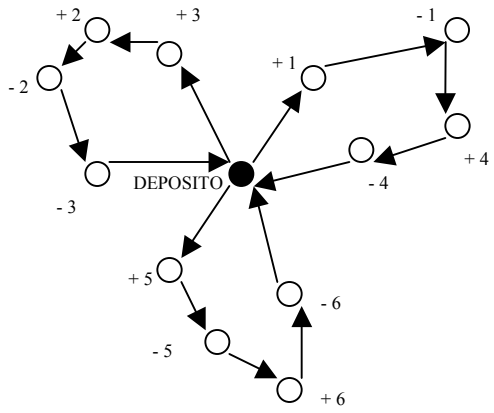

Figura 2.2: algoritmo Nearest Neighbour per il VRPSPD with Rear-Loading

2.2 Algoritmo Reversed Nearest Neighbour

E' una variante dell'algoritmo precedente. Il tour viene costruito muovendosi "a marcia indietro", vale a dire al contrario del Nearest Neighbour. Si parte quindi da V cammini costituiti solo dal nodo deposito e ad ogni iterazione si aggiunge il nodo più vicino: analogamente al caso precedente, se è di tipo delivery non ci sono problemi (tranne per il vincolo di capacità), mentre se è di tipo pick-up bisogna controllare se e in che cammino il nodo partner si trova in cima allo stack.

L’esempio in figura 2.3 mostra che il Nearest Neighbour (fig. 2.3a) e il Reversed Nearest Neighbour (fig. 2.3b) non danno necessariamente gli stessi risultati.

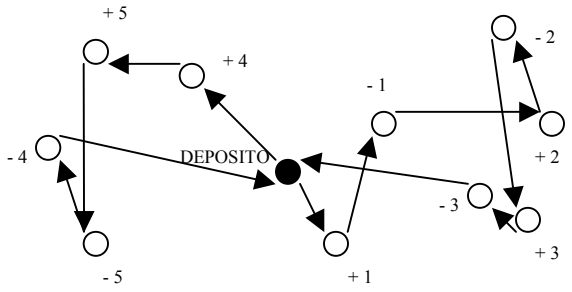

Figura 2.3a Nearest Neighbour

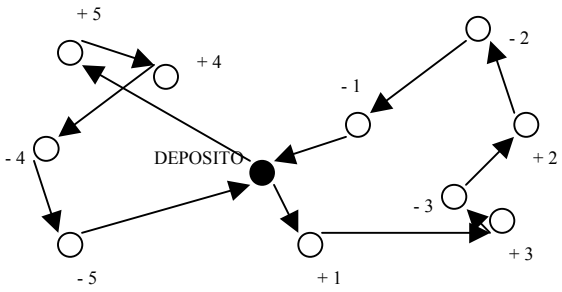

Figura 2.3b Reversed Nearest Neighbour

2.3 Algoritmo Randomized Nearest Neighbour

L’ algoritmo è una variante del Nearest Neighbour. Inizialmente si fissa un parametro λ. Ad ogni iterazione del programma per ogni nodo non ancora visitato si ordinano i rimanenti nodi ammissibili per distanza crescente. Cosi facendo ad ogni tappa si determina il prossimo nodo scegliendone casualmente uno tra i primi λ e lo si inserisce nel cammino con costo minore se di tipo pick-up, oppure se di tipo delivery nel cammino che presenta il relativo partner in cima allo stack.

L’algoritmo è di tipo multistart: eseguendolo più volte si possono ottenere soluzioni

iniziali differenti, permettendo all'euristica di ricerca locale di posizionarsi su diversi ottimi locali (fig. 2.4).

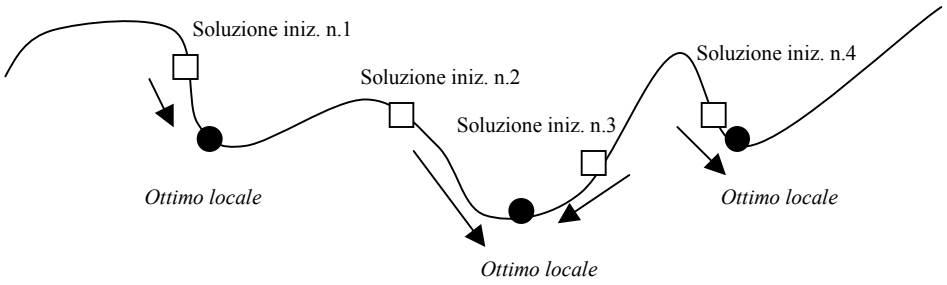

Figura 2.4: ripetendo l'inizializzazione l'algoritmo di ricerca locale può posizionarsi su differenti ottimi locali.

2.4 Algoritmo Reversed Randomized Nearest Neighbour

L'algoritmo è una variante del precedente, in cui i cammini sono costruiti muovendosi all'indietro (cioè se il nodo è di tipo delivery si inserisce nel cammino con costo minore, se il nodo è di tipo pick-up si inserisce solo nel tour con il relativo partner in cima allo stack). I valori generati casualmente sono diversi da quelli del Randomized Nearest Neighbour: in questo modo si riduce la possibilità di ottenere gli stessi tour iniziali.

2.5 Algoritmo Clarke – Wright VRP

L'algoritmo di Clarke e Wright [13] è senz'altro l'euristica più largamente conosciuta per il VRP. Si basa sulla nozione di "risparmio". Supponiamo che inizialmente la soluzione del VRP consista nell'usare per ogni nodo un veicolo differente per visitarlo (quindi n veicoli per n nodi). Se ora noi consideriamo due cammini $(0,i,0)$ e $(0,j,0)$ (fig. 2.5a) e utilizziamo un singolo veicolo per visitarli, la distanza percorsa dal veicolo sarà $d_{ij} = c_{i0} + c_{0j} + c_{ji}$.

La quantità

$$s_{ij} = c_{0i} + c_{j0} - c_{ji}.$$

è conosciuta come il "risparmio" risultante dalla combinazione del nodo i e j nel medesimo cammino (fig. 2.5b).

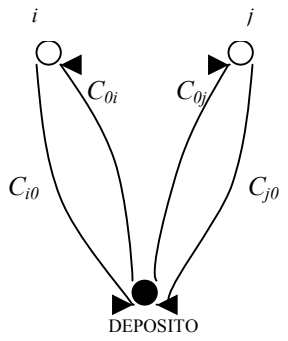

Fig. 2.5a: due singoli cammini

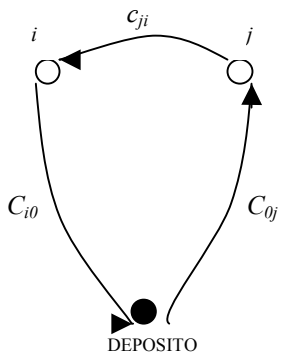

Fig. 2.5b: singolo cammino

L'algoritmo è il seguente:

- Calcolare i risparmi s_{ij} per ogni $i,j = 1..n$ con $i \neq j$. Creare per ogni nodo un cammino. Ordinare i risparmi in modo decrescente.
- Considerare a turno ogni cammino. Determinare il migliore risparmio s_{ij} che può essere ottenuto. Implementare l'unione dei due tour singoli in uno solo. Continuare finché il numero dei tour è uguale a V.

Questo algoritmo è stato creato per il VRP; in questa tesi è stato modificato in modo tale da poter funzionare anche per il PDPRL.

Nell'algoritmo sono stati aggiunti i vincoli di capacità e rear-loading, ed è stata mantenuta la clausola di chiusura del programma: si continua a iterare finché il numero dei tour è uguale a V.

Inoltre ad ogni iterazione viene controllato se è conveniente (quindi risparmio maggiore) unire tra di loro due cammini già esistenti nel seguente modo: partendo dai k cammini generati dall'iterazione precedente, prendendo in considerazione per ogni cammino solo i "percorsi/archi" che escono o entrano nel deposito (in fig. 2.6a evidenziati dal tratteggio), si cerca, controllando tutte le possibili combinazioni dei soli "percorsi/archi" presi in considerazione, di trovare quella con costo minore che unisce due cammini in un unico cammino. Questa ricerca a costo costante e in più mi da il vantaggio di non dover più controllare i vincoli di capacità e rear-loading in quanto sicuramente soddisfatti (fig. 2.6b trovo la combinazione con costo minore per unire due cammini togliendo i due archi evidenziati dal tratteggio).

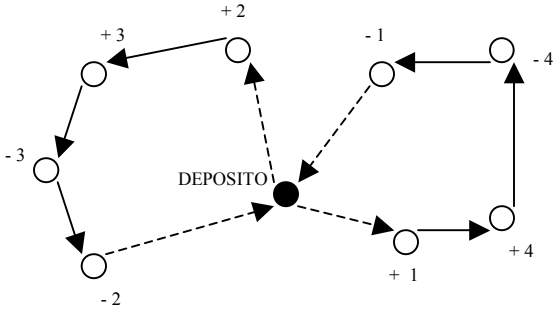

Fig. 2.6a: due singoli cammini

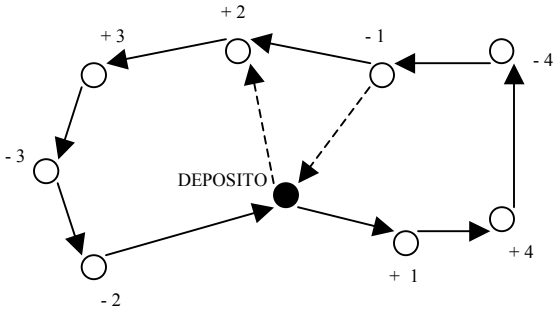

Fig. 2.6b: singolo cammino

Si continuano a unire a due a due i cammini fino a ottenere un numero totale di cammini pari al numero dei veicoli V .

2.6 Risultati sperimentali

Gli esperimenti condotti sui cinque algoritmi costruttivi presentati nella sezione precedenti sono stati effettuati in modo tale da verificare la qualità delle soluzioni calcolate dalle cinque euristiche.

Gli esperimenti sono stati condotti in modo tale da far variare sia il numero dei nodi, sia il numero dei veicoli utilizzati e sia la capacità di ogni veicolo.

Per tutti i test sono state utilizzate le mappe della libreria TSPLIB in cui le coordinate dei nodi sono espresse tramite una coppia di numeri interi (vedere appendice A).

Nella tabella 2.1 sono riportati i test effettuati sulla mappa “d18512” facendo variare il numero dei nodi, tendo fissi il numero dei veicoli pari a 4 e la capacità totale di ognuno di essi pari a 100. In questa tabella sono riportati anche i tempi totali di esecuzione di tutti gli algoritmi di costruzione dei cammini; come si può notare risultano essere molto bassi, anche con un numero levato di nodi. Inoltre sono evidenziati in grassetto i valori migliori trovati.

Risulta evidente che l’algoritmo costruttivo di Clarke e Wright risulta essere il migliore.

Nella tabella 2.2 si sono effettuati dei test facendo variare sia il numero dei nodi, sia la capacità totale di ogni veicolo, mantenendo costante il numero di questi ultimi pari a 4; si è verificato che all’aumentare della capacità dei veicoli il costo totale dei tour diminuisce.

Nella tabella 2.3 si sono effettuati dei test facendo variare sia il numero dei nodi, sia il numero di veicoli impiegati nella costruzione dei tour, mantenendo costante la loro capacità pari a 100; si è verificato che all’aumentare del numero dei veicoli il costo totale dei tour peggiora.

In tutte le prove effettuate l’algoritmo di Clarke e Wright ha sempre dato i risultati migliori, seguito dagli algoritmi Randomized Nearest Neighbour e Reversed Randomized Nearest Neighbour. Gli algoritmi Nearest Neighbour e Reversed Nearest Neighbour hanno sempre dato risultati molto lontani dai migliori.

num. clienti	NN	Rev-NN	Random-NN	Rev-Random-NN	Clarke Wright	Durata alg.
131	76555	77597	57268	68038	**31160**	< 1 sec.
337	188845	203805	154630	156439	**76126**	< 1 sec.
627	394689	420080	290881	284656	**161506**	2 sec.
889	674675	639032	463599	437653	**242730**	5 sec.
1001	779818	760127	523878	516743	**282807**	8 sec.

Tabella 2.1: test effettuati facendo variare il numero dei nodi.

num. clienti	capacità	NN	Rev-NN	Random-NN	Rev-Random-NN	Clarke Wright
131	80	83541	76548	60132	61286	**36548**
	100	78954	77697	57152	63038	**32654**
	140	76548	74569	58426	57630	**29636**
	200	67564	73654	55689	55535	**25874**
	220	68660	68974	54699	56842	**24999**
337	80	205487	213598	160312	157719	**82466**
	100	203128	209475	154630	156439	**76126**
	140	194574	206548	153984	171065	**67160**
	200	192782	203805	149925	158286	**66485**
	220	188845	201058	149659	172185	**62245**
627	80	430562	435487	322154	306457	**178215**
	100	415689	419015	299654	305656	**159687**
	140	401285	418764	301564	297205	**144602**
	200	388541	412185	300433	289128	**133954**
	220	375795	409058	289566	298064	**133854**
889	80	688452	643049	459687	457004	**258741**
	100	674541	639111	461489	446875	**246111**
	140	667412	631458	453902	439568	**220548**
	200	670241	622965	456871	437894	**195874**
	220	632146	618218	451205	434449	**193199**
1001	80	744698	761110	512048	533254	**306548**
	100	746514	759641	522365	511245	**296541**
	140	728547	754123	518954	509542	**273410**
	200	724569	751248	528555	513204	**246524**
	220	718549	719658	498775	508523	**254026**

Tabella 2.2: test effettuati facendo variare sia il numero dei nodi sia la capacità complessiva dei veicoli

num. clienti	num. Veicoli	NN	Rev-NN	Random-NN	Rev-Random-NN	Clarke Wright
131	4	76555	77597	57268	68038	**31160**
	6	70984	71158	58096	67660	**31550**
	8	68643	71023	68669	64057	**34760**
	10	68776	68156	62988	67014	**39208**
337	4	188845	203805	154630	156439	**76126**
	6	191038	196054	141269	162979	**76689**
	8	170436	198002	142797	157843	**79287**
	10	175263	199596	150267	155505	**82125**
627	4	394689	420080	290881	284656	**161506**
	6	390066	416801	299693	313822	**161564**
	8	404577	392071	319425	296695	**163568**
	10	382793	402611	309378	299561	**166567**
889	4	674675	639032	463599	437653	**242730**
	6	658688	629906	478279	468677	**247437**
	8	651730	615852	469637	470757	**248856**
	10	661177	615078	466852	463864	**252602**
1001	4	779818	760127	523878	516743	**282807**
	6	730439	743058	543303	519352	**284791**
	8	721046	724401	548358	522455	**285719**
	10	761158	687463	557901	528756	**287824**

Tabella 2.3: *test effettuati facendo variare sia il numero dei nodi sia il numero dei veicoli.*

Capitolo 3

Ricerca locale

Durante la definizione di algoritmi di ricerca locale per il PDPRL è importante considerare il fatto che nei cicli che formano la soluzione è molto probabile che i nodi caratterizzati da *pick-up demand* siano serviti all'inizio mentre i nodi caratterizzati da *delivery demand* alla fine; di conseguenza se si inverte un ciclo è molto probabile che i vincoli di capacità siano violati. E' quindi consigliabile utilizzare definizioni di intorni simili a quelli efficacemente utilizzati nella versione asimmetrica del CVRP che non si basano sull'inversione di parte della soluzione. Tre intorni identificati con i nomi RELOCATE, EXCHANGE utilizzati per risolvere il CVRP asimmetrico con algoritmi di ricerca locale sono illustrati in [14] e [15] e possono essere adattati per risolvere il PDPRL.

Nella definizione dei nuovi intorni è da tener infine presente che accettare soluzioni debolmente ammissibili è un modo intelligente di esplorare vasti intorni rimanendo vicini alla regione ammissibile.

Durante la ricerca locale una soluzione accettabile è rappresentata da un insieme di *h*

cicli fortemente ammissibili; ogni ciclo t è rappresentato da una lista ordinata di C_t visite ai clienti, identificate da $v_{t,k}$ con $k = 1..C_t$; ogni visita $v_{t,k} = (i,r_i)$ è rappresentata dall'indice i del cliente che assume valori tra 1 e M e un ammontare r_i del carico da raccogliere dal ($r_i > 0$) e da consegnare ($r_i < 0$) al cliente i. Al fine di controllare l'ammissibilità forte la seguente informazione addizionale è associata ad ogni visita $v_{t,k}$:

- l'ammontare totale del carico che si trova a bordo del veicolo t subito prima e subito dopo la visita ad un nodo, indicando rispettivamente con $c^{<}_{t,k}$ e $c^{>}_{t,k}$;

Valgono le seguenti relazioni:

Flusso inizializzazione (deposito):

$c^{<}_{t,1} = 0 \qquad \forall\, t = 1..h$

$c^{>}_{t,Ct} = 0 \qquad \forall\, t = 1..h$

Flusso conservazione (archi):

$c^{>}_{t,k} = c^{<}_{t,k+1} \qquad \forall\, t = 1..h \;\; \forall k = 1..C_t - 1$

Flusso conservazione (nodi):

$c^{<}_{t,k} + r_i = c^{>}_{t,k} \qquad \forall\, t = 1..h \;\; \forall k = 1..C_t - 1$

Gli algoritmi di ricerca locale che si basano sui nuovi intorni vengono inizializzati con una soluzione calcolata dalle euristiche costruttive esaminate nel capitolo precedente.

Nel resto della sezione S indica il numero delle visite ai clienti nella soluzione con cui

l'algoritmo di ricerca locale viene inizializzato: il suo valore è pari al numero dei clienti nel caso in cui la domanda sia indivisibile.

Per ogni nuovo intorno che viene di seguito definito vengono esaminate le procedure per generare nuove soluzioni, per controllare la loro ammissibilità, per valutarle e per aggiornare la soluzione corrente.

3.1 Relocate

Questo intorno include tutte le soluzioni fortemente ammissibili ottenute cancellando la visita ad una coppia di nodi origine $v_{t1,k1} = (i,r_i)$ e di nodi destinazione dal ciclo t_1 a cui appartengono e reinserendole in un ciclo t_2 (fig. 3.1).

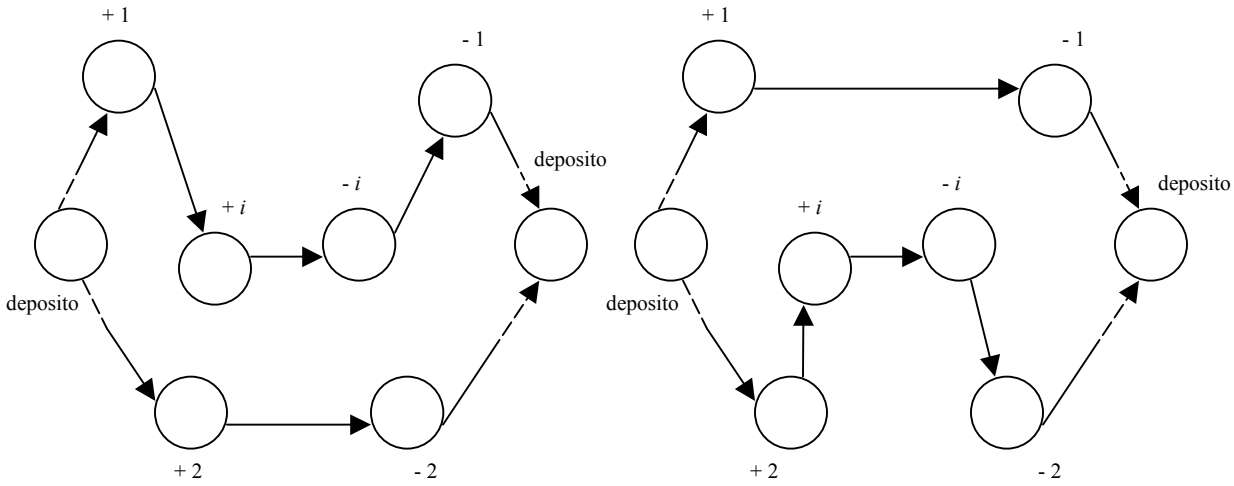

Figura 3.1: intorno Relocate per un cliente (coppia di nodi)

Generazione. Una nuova soluzione è generata per ogni cliente e per ogni coppia di posizioni di inserimento ammissibili; è possibile che $t_1 = t_2$ e cioè che le visite siano reinserite nello stesso ciclo in posizioni più convenienti. Il numero delle soluzioni considerate è $O(S^2)$.

Controllo dell'ammissibilità. E' effettuato in tempo costante. Devono essere verificate le seguenti condizioni perchè le visite (i,r_i) possa essere inserita tra $v_{t2,k2}$ e $v_{t2,k2+1}$:

$$c^{<}_{t2,k2+1} + r_i \leq Q$$

$$c^{>}_{t2,k2} + r_i \leq Q$$

queste condizioni devono essere valutate per tutti i nodi tra il nodo origine e il nodo destinazione del cliente i inseriti nel nuovo cammino.

Valutazione. Dipende dal numero di archi tra il nodo origine e il nodo destinazione del cliente i inseriti nel nuovo cammino : due archi sono sostituiti da uno quando la visita è eliminata dal ciclo a cui appartiene ed un arco è sostituito da due quando la visita è inserita nella nuova posizione, questo per entrambe le visite ai nodi; la differenza tra la lunghezza degli archi inseriti e quelli cancellati rappresenta il costo del passo di ricerca locale.

Aggiornamento. I valori associati ai cicli t_1 e t_2 devono essere parzialmente ricalcolati.

3.2 Exchange

Questo intorno include tutte le soluzioni fortemente ammissibili ottenute da quella corrente selezionando quattro nodi (due coppie) e scambiandoli fra di loro (fig. 3.2).

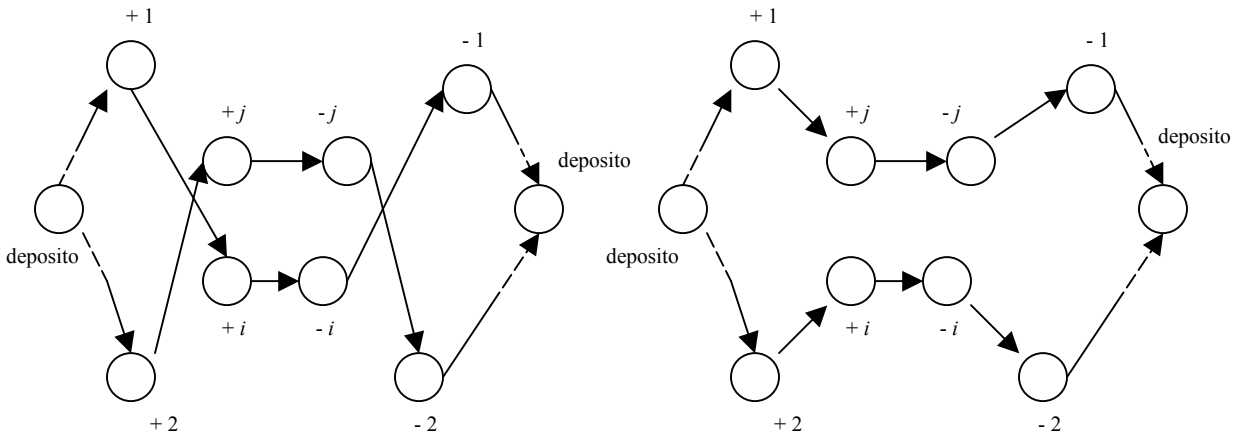

Figura 3.2 : intorno Exchange per due coppie (quattro nodi)

Generazione. Sono considerate tutte le coppie ordinate di visite; lo scambio è ammesso anche se le visite appartengono allo stesso ciclo. Il numero delle soluzioni considerate è quindi pari a $O(S^2)$.

Controllo dell'ammissibilità. Se $t_1 \neq t_2$, perchè la visita $v_{t2,k2} = (j,r_j)$ con la relativa partner possa sostituire la visita $v_{t1,k1} = (i,r_i)$ e partner, devono essere verificate le seguenti condizioni relativamente a t_1:

$c^{>}_{t1,k1} + r_j - r_i \leq Q$ $\quad \forall$ *r* carico da raccogliere

$c^{<}_{t1,k1} + r_j - r_i \leq Q$ $\quad \forall$ *r* carico da consegnare

queste condizioni devono essere valutate per tutti i nodi tra il nodo origine e il nodo destinazione del cliente *i* inseriti nel nuovo cammino e analoghe condizioni devono essere soddisfatte per t_2; l'operazione richiede quindi tempo che dipende dal numero di nodi

presenti tra i nodi origine e i nodi destinazione dei clienti i e j inseriti nei nuovi cammini. Se invece $t_1 = t_2$ (supponendo senza perdita di generalità che $k_1 < k_2$) il controllo è effettuato verificando che sia rispettato il vincolo di capacità su ogni arco compreso nel segmento del ciclo delimitato dalle visite $v_{t1,k1}$ e $v_{t2,k2}$.

Valutazione. Il valore delle nuove soluzioni dipende solo dagli archi coinvolti nello scambio e può essere quindi calcolato in tempo costante.

Aggiornamento. I valori delle strutture dati associate ai cicli t_1 e t_2 devono essere parzialmente ricalcolati.

3.3 Intorno variabile e complesso

Tutti gli intorni definiti in precedenza vengono definiti intorni semplici. Quando sono disponibili differenti intorni semplici su cui basare l'algoritmo di ricerca locale, è possibile combinarli in vari modi per trovare nuove strategie di ricerca.

In particolare, quando tutti gli intorni semplici vengono esplorati ad ogni passo della ricerca locale, scegliendo ogni volta la soluzione migliore, l'ottimo viene cercato con strategia *Complex Neighbourhood Descent* (CND) formando un intorno complesso.

Quando invece gli intorni semplici sono utilizzati alternativamente in modo che se in un intorno non esistono soluzioni miglioranti ne viene esplorato un altro, e se si migliora si ricomincia dal primo intorno, l'ottimo viene cercato con strategia *Variable Neighbourhood Descent* (VND) definendo cosi un intorno variabile.

Capitolo 4

Tabu Search

Il tabu search (TS) è un algoritmo di ricerca locale in cui si permette alla funzione obiettivo di peggiorare; in questo modo la ricerca non termina appena viene trovato un minimo locale e lo spazio delle soluzioni del problema in questione viene ampiamente esplorato. Per evitare che l'algoritmo continui a considerare sempre le stesse soluzioni, quelle recentemente esaminate vengono inserite in una lista delle soluzioni proibite (tabu list) e vi rimangono per un certo numero di iterazioni; un passo di ricerca locale che porta a considerare una soluzione presente nella tabu list è detto mossa tabu. Tipicamente la ricerca termina dopo che è stato eseguito un determinato numero di iterazioni senza ottenere dei miglioramenti o più semplicemente dopo un numero fisso di iterazioni. Il tabu search è una tecnica generica i cui parametri di funzionamento devono essere opportunamente tarati a seconda del tipo di problema. Per una descrizione più esauriente dell'algoritmo si faccia riferimento a [16].

Nella letteratura scientifica relativa ai VRPs molti ricercatori hanno riportato buoni risultati ottenuti attraverso il TS; per es. l'algoritmo è stato applicato con successo alla

risoluzione del TSPSPD in [17] e del CVRP in [18] e [19].

4.1 Descrizione generale dell'algoritmo

Per rendere efficiente l'algoritmo, invece di memorizzare l'intera soluzione è possibile memorizzare alcuni suoi attributi e considerare tabu tutte quelle soluzioni in possesso di tali attributi.

Basandosi su questa osservazione, sugli intorni semplici definiti nel capitolo precedente (RELOCATE, EXCHANGE) e sulle strategie di ricerca, sono state realizzate varie implementazioni del TS per il PDPRL che hanno in comune le seguenti caratteristiche:

- ad ogni iterazione effettuano un passo di ricerca locale che porta sempre a considerare soluzioni ammissibili;
- effettuano passi di ricerca locale verso soluzioni tabu (soluzioni ammissibili i cui attributi appartengono alla tabu list) solo se il loro costo e minore di quello della migliore soluzione trovata fino a quel momento;
- terminano dopo *T* iterazioni oppure quando a partire dalla soluzione corrente non è più possibile trovare soluzioni ammissibili non tabu .

Negli algoritmi TS la tabu list è implementata attraverso una matrice (tabu(i,j)) che memorizza per ogni coppia la più recente iterazione in cui è stata inclusa. Ogni elemento

delle matrici è inizializzato a -∞ (in realtà ad un numero negativo molto grande).

L'utilizzo delle matrici consente di verificare se una mossa è tabu o meno in tempo costante.

La lunghezza della tabu list è determinata grazie ad un parametro L: all'iterazione l-esima sono presenti nella tabu list tutte le coppie per cui $l - l_c \leq L$, dove l_c indica l'iterazione più recente in cui la coppia c è stata inclusa. Quindi il controllo per verificare se una mossa è tabu è il seguente:

$$l_c + L \leq l$$

La scelta del valore di L però deve tenere conto di alcune problematiche, se L risulta essere troppo grande vincola troppo la ricerca favorendo la *diversificazione* (cioè cercare la soluzione migliore campionando uniformemente tutta l'area di ricerca), mentre se L risulta essere troppo piccolo si privilegia l'*intensificazione* (cioè cercare la soluzione migliore attraverso una ricerca raffinata in un'area delimitata di soluzioni) con lo svantaggio di favorire la creazione di sotto cicli. Per questo si è scelto una politica adattativa per cui il valore di L viene fatto variare in un intervallo $[L_{min},L_{max}]$ nel seguente modo:

- se se viene fatta una mossa migliorante (un passo di ricerca locale che porta a considerare una soluzione migliore di quella attuale) il nuovo valore di L è $max\{L - 1,L_{min}\}$;

- se se viene fatta una mossa peggiorante il nuovo valore di L è $min\{L + 1, L_{max}\}$;

Per ogni TS continuano a valere le analisi delle operazioni di generazione, controllo dell'ammissibilità, valutazione e aggiornamento relative all'intorno utilizzato.

4.2 Algoritmo TS con intorno Relocate

L'algoritmo TS si basa sull'intorno Relocate, e ad ogni interazione, se il passo di ricerca prevede che la visita *pick-up* $v_{t1,k1} = (i,r_i)$ con la relativa visita *delivery* vengano spostate dal ciclo t_1 e reinserite in un altro ciclo, è memorizzato nella tabu list la coppia di attributi (t_1, i).

Mentre si sta cercando di effettuare un passo di ricerca locale, per verificare se una mossa è tabu o meno bisogna effettuare il seguente controllo: spostare la visita *pick-up* $v_{t1,k1} = (i_1,r_i)$ con la relativa visita *delivery* dal ciclo t_1 al ciclo t_2, fra le visite $v_{t2,k2} = (i_2,r_{i2})$ e $v_{t2,k2+1} = (j_2,r_{j2})$, è una mossa tabu se è presente nella tabu list la coppia di attributi (t_2, i_1) (fig. 4.1).

```
l  := 0;

repeat
{
        trovo visita i1 da spostare dal ciclo t1 al ciclo t2 con costo minore;
        if (tabu(i1,t2) + L <= l) then
         {
               Applico lo scambio di posizioni dei nodi nel nuovo tour;
               L = max{L - 1,Lmin};
         }
        else
         {
               if (posso accettare mossa tabu (il costo è minore di quello
                   della migliore soluzione)) then
                {
                    Applico lo scambio di posizioni dei nodi nel nuovo tour;
                    L = max{L - 1,Lmin};
                }
               else
                {
                    L = min{L + 1,Lmax};
                }
         }
        l := l+1;
        if (nessuna mossa migliorante) then break;
}
until l < T;
```

Figura 4.1: pseudo codice tabu search con intorno Relocate

4.3 Algoritmo TS con intorno Exchange

L'algoritmo TS si basa sull'intorno Exchange, e ad ogni interazione, se il passo di ricerca prevede che la visita *pick-up* $v_{t1,k1} = (i,r_i)$ con la relativa visita *delivery* vengano scambiate con la visita *pick-up* $v_{t2,k2} = (j,r_j)$ e la relativa visita *delivery*, sono memorizzate nella tabu list le coppie di attributi (t_1, i) e (t_2, j).

Mentre si sta cercando di effettuare un passo di ricerca locale, per verificare se una mossa è tabu o meno bisogna effettuare il seguente controllo: spostare la visita *pick-up* $v_{t1,k1} = (i_1,r_i)$ con la relativa visita *delivery* al posto della visita $v_{t2,k2} = (j,r_j)$ e la relativa visita

delivery, è una mossa tabu se è presente nella tabu list la coppia di attributi (t_2, i_1) oppure (t_1, i_2) (fig. 4.2).

```
l  := 0;

repeat
{
        trovo visita i1 da spostare al posto di visita i2 con costo minore;
        if (tabu(i1,t2) + L <= l)&&( tabu(i2,t1) + L <= l) then
         {
               Applico lo scambio di posizioni dei nodi;
               L = max{L - 1,Lmin};
         }
        else
         {
               if (posso accettare mossa tabu (il costo è minore di quello
                   della migliore soluzione)) then
                {
                    Applico lo scambio di posizioni dei nodi;
                    L = max{L - 1,Lmin};
                }
               else
                {
                    L = min{L + 1,Lmax};
                }
         }
        l := l+1;
        if (nessuna mossa migliorante) then break;
}
until l < T;
```

Figura 4.2: pseudo codice tabu search con intorno Exchange

4.4 Algoritmo TS con intorno VND

In questo algoritmo TS l'ottimo viene cercato con strategia Variable Neighbourhood Descent, cioè le euristiche descritte sopra vengono ordinate secondo un criterio stabilito e si comincia a cercare la soluzione migliore attraverso il primo algoritmo. Quando non si può più migliorare si passa al secondo. Se si migliora si ricomincia dal primo, altrimenti si

passa al successivo e cosi via. Nel nostro caso gli algoritmi da utilizzare sono due (fig. 4.3).

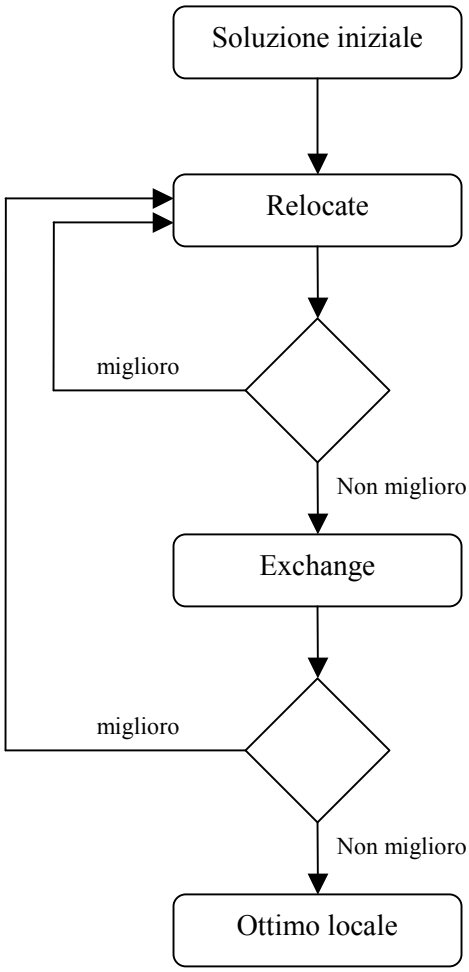

Figura 4.3: funzionamento della strategia Variable Neighbourhood Descent

4.5 Algoritmo TS con intorno CND

In questo algoritmo TS l'ottimo viene cercato con strategia Complex Neighbourhood Descent. Con questa tecnica, l'algoritmo TS cerca l'ottimo esplorando tutti gli intorni ad ogni iterazione, scegliendo ogni volta la soluzione migliore, altrimenti si arresta su un ottimo locale (fig. 4.4).

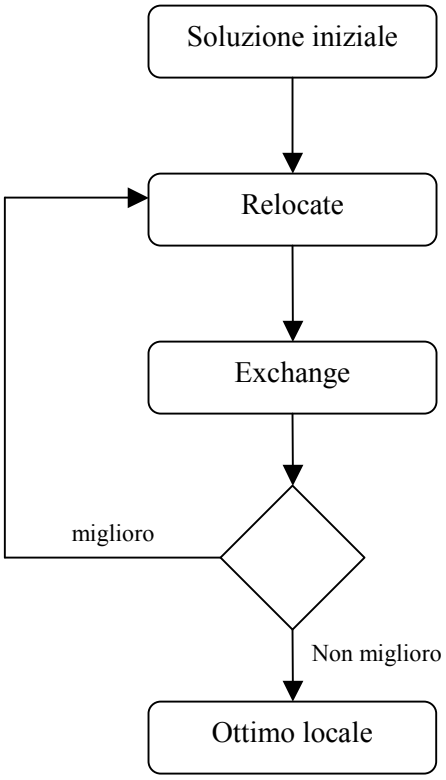

Figura 4.4: funzionamento della strategia Complex Neighbourhood Descent

4.6 Tecniche per evitare ricalcalo del costo di tutte le mosse

Per entrambi gli algoritmi tabu search con intorno semplice sono state utilizzate delle tecniche per evitare il ricalcalo del costo di tutte le mosse, intese come costo di inserimento di un nodo in un cammino.

Innanzi tutto va definito che durante l'esecuzione degli algoritmi il costo delle mosse viene memorizzato in una matrice che presenta come numero di righe il numero N di nodi, mentre come numero di colonne il numero V di veicoli.

In questo modo dopo aver scelto la mossa migliore da effettuare (in base al tipo di ricerca locale utilizzata), all'iterazione successiva viene ricalcolato solo il costo del cammino (colonna) in cui sono stati inseriti i nodi scelti per lo spostamento e i costi relativi ai nodi (riga) utilizzati nello spostamento.

In questo modo viene risparmiata una ingente quantità di calcoli e di tempo.

4.7 Risultati sperimentali

Gli esperimenti condotti sui quattro algoritmi tabu search presentati sono stati effettuati in modo tale da verificare la qualità delle soluzioni calcolate.

Gli esperimenti sono stati condotti in modo tale da far variare sia il numero dei nodi, sia il numero dei veicoli utilizzati per effettuare le commissioni e sia la capacità totale di ogni veicolo.

Per tutti i test sono state utilizzate le mappe della libreria TSPLIB in cui le coordinate dei nodi sono espressi tramite una coppia di numeri interi (vedere appendice A).

Per effettuare i test sono stati utilizzati come tour in ingresso ai tabu search quelli costruiti attraverso l'algoritmo Nearest Neighbour e Clarke e Wright, quindi i due algoritmi che hanno dato rispettivamente i peggiori e i migliori risultati.

In tutte le tabelle seguenti sono riportati i costi (intesi come lunghezza dei tour) trovati dagli algoritmi con indicato il relativo tempo di esecuzione espresso in secondi.

Nella tabella 4.1 sono presentati i valori relativi alla lunghezza della tabu list L_{max} e L_{min}. E' stato notato nell'implementazione delle prove che l'algoritmo tabu search con intorno Relocate e gli algoritmi VND e CND risultano essere molto sensibili a ogni variazione dei due parametri L_{max} e L_{min}, mentre l'algoritmo con intorno Exchange pare non risentire molto di questo fattore.

Nelle tabelle 4.2 e 4.3 si sono effettuati dei test facendo variare sia il numero dei nodi, sia la capacità totale di ogni veicolo, mantenendo costante il numero di questi ultimi pari a 4; nella prima tabella si sono utilizzati come costi iniziali quelli calcolati dall'algoritmo Nearest Neighbour, mentre nella seconda tabella quelli calcolati dall'algoritmo di Clarke e Wright.

Si è verificato che l'algoritmo che presenta i risultati con costo minore (evidenziati in nero) è il tabu search con strategia di ricerca VND, seguito dal tabu search con strategia di ricerca CND. Si nota inoltre che l'algoritmo tabu search con intorno Exchange risulta essere il più veloce a scapito però di risultati molto lontani dai migliori trovati.

Nelle tabelle 4.4 e 4.5 si sono effettuati dei test facendo variare sia il numero dei nodi,

sia il numero di veicoli impiegati nella costruzione dei tour, mantenendo costante la loro capacità pari a 100; nella prima tabella si sono utilizzati come costi iniziali quelli calcolati dall'algoritmo Nearest Neighbour, mentre nella seconda tabella quelli calcolati dall'algoritmo di Clarke e Wright.

Anche questi test hanno confermato sia la qualità delle soluzioni calcolate con il tabu search con strategia di ricerca VND, sia i scarsi risultati trovati con l'algoritmo tabu search con intorno Exchange.

Prendendo in considerazione solo i test che utilizzano come tour iniziali quelli calcolati dall'algoritmo di Clarke e Wright, si è trovato che calcolare il costo dei tour con il tabu search con strategia di ricerca VND da un risparmio che varia dal 10% al 20% rispetto al costo iniziale.

La tabella 4.6 presenta dei test effettuati su un numero di 529 nodi con un numero di veicoli pari a 4 e una capacità totale uguale a 100. In questa tabella sono stati applicati gli algoritmi tabu search su i tour iniziali calcolati con tutti gli algoritmi costruttivi presentati precedentemente. In questi test il tabu search con intorno Relocate e quello con strategia di ricerca VND hanno dato gli stessi risultati; però sono stati scelti come migliori (evidenziati in nero) quelli calcolati dall'algoritmo con intorno semplice per il minore tempo in cui sono stati calcolati.

	Relocate		Exchange		VND		CND	
	L min	*L max*	*L min*	*L max*	*L min*	*L max*	*L min*	*L max*
NN	10	50	10	40	15	55	10	35
Rev-NN	20	40	15	45	10	45	10	35
Random-NN	15	40	10	40	10	50	15	30
Rev-Random-NN	15	40	10	45	10	50	15	30
Clarke Wright	20	55	10	50	15	45	10	30

Tabella 4.1: lunghezza parametro L

n. clienti	capacità	Costo iniziale	Relocate		Exchange		VND		CND	
			costo	tempo	costo	tempo	costo	tempo	costo	tempo
	80	55738	33735	1	38504	1	**24714**	1	25511	2
	100	48914	29271	1	30697	1	**17957**	2	22345	2
89	140	46213	21393	1	41605	1	**19584**	13	20184	12
	200	38888	29307	1	31703	1	**19176**	7	23391	13
	220	38888	29307	1	22669	1	**19571**	2	22669	3
	80	69288	31086	2	53812	1	**26680**	14	29997	13
	100	64875	39126	2	39126	1	**24339**	3	25666	10
131	140	65048	44425	3	44425	1	**23762**	7	24524	7
	200	55654	48098	3	48098	1	**21713**	10	23297	6
	220	60259	38152	3	38152	1	28253	7	**23732**	5
	80	149541	63109	8	134498	1	**63076**	20	63109	28
	100	135875	65630	9	135875	1	79979	20	**65630**	25
247	140	139906	65493	15	118259	1	**55622**	31	64114	51
	200	130120	54539	17	128974	1	**54506**	35	54539	41
	220	135737	49439	21	90522	1	**37451**	51	49439	41
	80	193946	92417	22	193946	1	92417	22	**75621**	65
	100	199062	75950	20	177924	1	**75924**	40	75924	50
337	140	214566	68123	25	170632	1	**67972**	60	68123	90
	200	192939	134896	45	184834	2	**68652**	72	71280	83
	220	190600	69639	50	190600	1	**63888**	101	67492	81
	80	250977	111552	33	250977	1	**105641**	70	111552	90
	100	257253	**105762**	31	257253	1	105762	58	105762	60
401	140	255647	92089	45	233859	2	**91869**	102	92089	113
	200	250533	129453	46	250533	2	**129357**	109	129357	141
	220	241615	156553	61	224601	2	**77874**	209	154483	256

Tabella 4.2: test effettuati facendo variare sia il numero dei nodi sia la capacità complessiva dei veicoli, con costi iniziali calcolati con algoritmo Neareast Neighbour

n. clienti	capacità	Costo iniziale	Relocate		Exchange		VND		CND	
			costo	tempo	costo	tempo	costo	tempo	costo	tempo
89	80	24278	**20857**	1	23562	1	20807	1	21092	2
	100	24061	22534	1	23140	1	21361	2	**21044**	2
	140	21806	19680	1	21125	1	**16429**	5	16429	11
	200	21420	20850	1	21131	1	17679	7	**16829**	8
	220	21187	22145	1	21187	1	**20944**	6	20944	5
131	80	31713	31713	2	**31713**	1	31713	4	31713	11
	100	31869	24698	2	58923	1	26249	5	**25650**	10
	140	26380	25725	2	27920	1	**24416**	7	25725	7
	200	25656	21961	2	25269	1	**19619**	9	21961	8
	220	25992	23585	2	25902	1	**20366**	9	20488	6
247	80	65923	58790	8	64468	1	58790	20	**58652**	24
	100	60583	50582	10	59320	1	**50386**	26	50582	27
	140	54851	**49887**	15	53933	1	49887	33	49887	51
	200	49127	43280	16	48454	1	**40505**	35	42214	46
	220	47344	39156	22	47344	1	**38885**	49	39768	43
337	80	79278	64694	20	78467	1	**64694**	20	64694	65
	100	72330	64875	22	71889	1	**64520**	43	64875	50
	140	69571	61501	25	69571	1	**51501**	59	61406	86
	200	62929	56496	43	62929	2	**55277**	78	56496	85
	220	61061	50975	48	60955	2	60955	89	**50975**	87
401	80	108298	97674	30	107563	2	**97620**	62	97674	83
	100	99199	87534	32	98039	2	**84659**	54	87534	64
	140	87993	83611	41	87399	2	**72918**	102	82388	112
	200	80728	74103	48	80728	2	**68928**	116	71459	145
	220	79313	75570	61	78973	3	**67632**	194	75570	238

Tabella 4.3: test effettuati facendo variare sia il numero dei nodi sia la capacità complessiva dei veicoli, con costi iniziali calcolati con algoritmo di Clarke e Wright

n. clienti	Veicoli	Costo iniziale	Relocate		Exchange		VND		CND	
			costo	tempo	costo	tempo	costo	tempo	costo	tempo
89	4	48914	29271	1	30697	1	**17957**	2	22345	2
	6	46342	32327	1	27691	1	**22407**	2	24110	1
	8	46340	32327	1	28971	1	24513	11	**24110**	1
	10	46340	32327	1	28971	1	24513	11	**24110**	1
131	4	64875	39126	2	39126	1	**24339**	3	25566	10
	6	50380	34812	2	36767	1	29678	4	**27604**	8
	8	47676	28645	2	34724	1	**23686**	4	27206	9
	10	47676	28645	2	34724	1	**23686**	4	27206	9
247	4	135875	**65630**	7	135875	1	65630	21	79979	21
	6	140372	57040	10	103675	1	**56454**	22	57040	11
	8	140399	61676	14	94071	1	**54568**	21	56310	32
	10	130582	63392	12	89861	1	**57543**	18	58001	22
337	4	199062	**75950**	14	177924	1	75950	42	75950	54
	6	190293	72352	26	153469	1	**70581**	56	72352	73
	8	193679	129076	28	160762	1	78555	73	**75895**	78
	10	184226	93820	27	121837	2	**89330**	58	89330	62
401	4	257253	**105762**	30	257253	1	105762	58	105762	101
	6	244848	100184	44	203534	2	**95279**	103	99485	89
	8	236186	97028	45	172299	2	92037	102	**86699**	153
	10	228707	103831	42	146998	2	**87647**	95	95279	91

Tabella 4.4: test effettuati facendo variare sia il numero dei nodi sia il numero dei veicoli, con costi iniziali calcolati con algoritmo Neareast Neighbour

n. clienti	Veicoli	Costo iniziale	Relocate		Exchange		VND		CND	
			costo	tempo	costo	tempo	costo	tempo	costo	tempo
89	4	24061	22534	1	23140	1	21361	2	**21044**	3
	6	23456	19326	1	23456	1	**17912**	16	19326	2
	8	23456	19326	1	23240	1	**17912**	16	19326	2
	10	23456	19326	1	23240	1	**17912**	16	19326	2
131	4	31869	24698	1	30779	1	24670	4	**24249**	6
	6	35430	35430	2	34419	1	**28669**	5	28687	8
	8	36184	32746	2	33645	1	**25483**	7	32369	7
	10	38677	33787	2	35955	1	**28669**	5	28687	8
247	4	60583	50582	8	59320	1	50582	21	**50386**	23
	6	63322	52340	10	60255	1	**50643**	22	52340	11
	8	65152	55128	16	63675	1	**50258**	21	55128	31
	10	65567	**51826**	14	63516	1	59928	16	51826	23
337	4	72330	64875	21	71889	1	**64520**	38	64875	43
	6	75601	68649	26	75070	1	66002	53	**65148**	73
	8	77390	63675	28	75272	1	**63398**	74	63675	78
	10	78103	62611	26	75712	1	**62243**	58	62611	62
401	4	99199	87534	29	98039	1	**84659**	57	87534	61
	6	103812	87193	47	100925	1	**84667**	104	87193	84
	8	105209	89425	45	101282	2	**87550**	106	89425	126
	10	106701	88541	42	100897	2	**84672**	96	88541	49

Tabella 4.5: test effettuati facendo variare sia il numero dei nodi sia il numero dei veicoli, con costi iniziali calcolati con algoritmo di Clarke e Wright

	Costo iniziale	Relocate	Exchange	VND	CND
NN	293158	**209603**	293158	209603	215600
Rev-NN	364652	**134612**	364652	134612	134612
Random-NN	255493	**185448**	251475	185448	188529
Rev-Random-NN	251084	**122862**	244659	122862	122862
Clarke Wright	128952	124612	126635	118545	**117796**
Tempo		279	11	389	493

Tabella 4.6: test effettuati su un numero di nodi pari a 529, utilizzando 4 veicoli con capacità pari a 100.

Conclusioni

Confronto fra gli algoritmi

Nella presente trattazione sono state descritte euristiche per risolvere il PDPRL, sono stati implementati algoritmi costruttivi e algoritmi tabu search con diversi intorni.
Gli esperimenti svolti hanno permesso di individuare quale o quali delle euristiche fosse la migliore.

Dai test risulta che tra gli algoritmi costruttivi il migliore sia l'algoritmo di Clarke and Wright modificato per il PDPRL, mentre gli algoritmi di costruzione dei cammini basati sul Nearest Neighbour in generale non hanno un buon comportamento dato che spesso producono valori molto alti.

Tra i tabu search con intorno semplice il migliore è il Relocate, mentre è stato osservato che il tabu search con intorno Exchange con questo tipo di problema a volte non produce miglioramenti, in termine di costo, rispetto ai tour iniziali a causa dei troppi vincoli presenti.

La strategia di tipo Variable Neighbourhood Descent si è dimostrata la migliore sia come qualità delle soluzione sia come tempi di esecuzione rispetto la strategia Complex Neighbourhood Descent.

Per quanto riguarda possibili futuri sviluppi si possono considerare nuovi algoritmi euristici ma anche algoritmi esatti con cui poter confrontare i risultati trovati.

Bibliografia

[1] P. Toth, D. Vigo, *The Vehicle Routing Problem.* Paolo Toth and Daniele Vigo, University of Bologna, SIAM Monographs on Discrete Mathematics and Applications, vol. 9, (2002), 1-26.

[2] P. Toth, D. Vigo, *A heusristic algorithm for the symmetric and asymmetric Vehicle Routing Problem with backhauls.* European Journal of Operational Research, 113, (1999), 528-543.

[3] P. Toth, D. Vigo, *Exact algorithms for the Vehicle Routing Problem with backhauls.* Transportation Science, 31, (1997), 60-71.

[4] M.W.P. Savelsbergh, *Local search in routing problems with time windows*. Annals of Operations Research, 4:285-305, 1985.

[5] M. W. P. Savelsbergh, M. Sol, *The general pickup and delivery problem.* Transportation Science, 29, (1995), 17-29.

[6] G. Mosheiov, *Vehicle Routing with pick-up and delivery: tour partitioning heuristics.* Computer and Industrial Engineering, 34, 3, (1998), 669-684.

[7] G. Mosheiov, *The travelling salesman problem with pick-up and delivery*. European Journal of Operational Research, 72, (1994), 299-310.

[8] S. Anily, G. Mosheiov, *The traveling salesman problem with delivery and backhauls.* Operations Research Letters, 16, (1994), 11-18.

[9] S. Anily, J. Bramel, *Approximation Algorithms for the Capacited Traveling Salesman Problem with Pick-up and Delivery.* Tristan III, Puerto Rico, (1998).

[10] G. Righini, *The Largest Insertion algorithm for the Travelling Salesman Problem.* Note del Polo – Ricerca, 29, Polo Didattico e di Ricerca di Crema, Università di Milano, (2000).

[11] G. Laporte, F. Semet, *The Vehicle Routing Problem.* Paolo Toth and Daniele Vigo, DEIS – University of Bologna, SIAM Monographs on Discrete Mathematics and Applications, vol. 9, (2002), 109-128.

[12] M. Gendreau, G. Laporte, J. Y. Potvin, *The Vehicle Routing Problem.* Paolo Toth and Daniele Vigo, DEIS – University of Bologna, SIAM Monographs on Discrete Mathematics and Applications, vol. 9, (2002), 129-154.

[13] G. Clarke and J. V. Wright. *Scheduling of vehicle from a central depot to a number of delivery points.* Operations Research, 12:568-581, 1964.

[14] A. P. Kindervater, M. W. P. Savelsbergh, *Vehicle routing: handling edge exchanges in Local Search in Combinatorial Optimization.* Aarts E. and Lenstra J. K. eds., Wiley-Interscience Series in Discrete Mathematics and Optimization, Chichester, (1997).

[15] D. Vigo, *A heuristic algorithm for the asymmetric capacitated vehicle routing problem.* European Journal of Operational Research, 89, (1996), 108-126.

[16] F. Glover, E'. Taillard, D. de Werra, *A user's guide to tabu search.* Annals of

Operations Research, 41, (1993), 3-28.

[17] M. Gendrau, G. Laporte, D. Vigo, *Heuristics for the traveling salesman problem with pick-up and delivery.* Computers and Operations Research, 26, (1999), 699-714.

[18] M. Gendrau, G. Laporte, J. Y. Potvin, *Vehicle routine: modern heuristics in Local Search in Combinatorial Optimizatio.* Aarts E. and Lenstra J. K. eds., Wiley-Interscience Series in Discrete Mathematics and Optimization, Chichester, (1999).

[19] M. Gendrau, A. Hertz, G. Laporte, *A Tabu Search Heuristic for the Vehicle Routine Problem.* Report CRT-777, Centre de recherché sur les transport, Université de Montreal, (1992), Management sci. to appear.

Appendice A

A.1 Specifiche tecniche della macchina

Gli algoritmi sono stati scritti in C standard, e compilati con Bloodshed Dev-C++ versione 4.9.9.2. Tutti i test sono stati eseguiti su un PC con processore Intel-Pentium 4 2.66 GHz, dotato di 512 MB di memoria RAM e sistema operativo Microsoft Windows XP Home Edition.

A.2 Formato dei file

I programmi ricevono in ingresso un file di testo con estensione ".tsp", appartenente alla nota libreria TSPLIB, che riporta le coordinate sul piano dei nodi. Modificando il parametro "DIMENSION" del file è possibile scegliere con quanti nodi lavorare (fig. A.1). In questo file di testo dopo la parola chiave "NODE_COORD_SECTION" vengono riportate le coordinate dei nodi sul piano cartesiano; la prima colonna riporta un numero

intero che identifica il nodo, mentre le due rimanenti colonne riportano le coordinate rispettivamente sull'asse delle ordinate e sull'asse delle ascisse.

```
NAME : d18512
COMMENT : Bundesrepublik Deutschland (mit Ex-DDR) (Bachem/Wottawa)
TYPE : TSP
DIMENSION : 1001
EDGE_WEIGHT_TYPE : EUC_2D
NODE_COORD_SECTION
    1    2918    6528
    2    2925    6597
    3    2926    6609
    4    2927    6312
    5    2930    6328
    6    2934    6545
    7    2938    7412
    8    2941    6456
    9    2945    6284
   10    2947    6663
   11    2948    5475
      […]
```

Figura A.1 : esempio di contenuto di un file TSPLIB

Gli accoppiamenti tra le città sono scritti in un file con estensione ".mtc", creato da un programma che ho appositamente creato. Dopo aver suddiviso casualmente, utilizzando il generatore di numeri casuali forniti nella libreria standard del C, i nodi in due gruppi di ugual numero, ho definito un gruppo l'insieme dei nodi origine, mentre l'altro, l'insieme dei nodi destinazione. Utilizzando nuovamente il generatore di numeri casuali, ho associato casualmente ad ogni nodo origine un nodo destinazione generando anche la quantità del carico associata ad ogni coppia. Il seme del generatore è determinato dal valore dell'orologio di sistema.

Il file cosi formato contiene, per ogni nodo, il tipo nella seconda colonna (zero se di tipo nodo destinazione, uno se nodo origine), il nodo partner nella terza colonna e il

valore del carico, da caricare o raccogliere, nella quarta colonna (positivo se di pick-up, negativo se di delivery) (fig. A.2).

Per il nodo deposito il tipo, il nodo partner e il carico sono valori fittizi.

```
0 3 0 0
1 1 8 16
2 0 3 -33
3 1 2 33
4 0 5 -38
5 1 4 38
6 1 7 31
7 0 6 -31
8 0 1 -16
```

Figura A.2 : esempio di accoppiamento dei nodi. La prima colonna indica il nodo, la seconda il tipo, la terza la partner e la quarta il carico

Il programma di costruzione dei commini iniziali riceve in input la mappa e le coppie. L'output consiste in un file per ogni euristica, che ha estensione ".str" e contiene i cammini per ogni veicolo, identificati dalla sequenza di nodi da visitare, e la lunghezza totale dei percorsi effettuati (fig A.3).

```
0
5
1
2
8
6
3
0
Fine tour:1
0
4
7
0
Fine tour:2

Lunghezza totale dei tour: 3262
```

Figura A.3 : esempio di file di output di un algoritmo costruttivo calcolato con il Nearest Neighbour

I programmi tabu search ricevono in ingresso la mappa, le coppie e i cammini generati dalle euristiche costruttive. L'output consistono in un file per ogni tabu search implementato, con estensione ".str" che contiene il cammino per ogni veicolo e la lunghezza totale dei percorsi effettuati.

Inoltre in output vengono generati, per ogni euristica, dei file con estensione ".txt" indicanti il tempo di esecuzione del programma e il valore minimo trovato (fig A.4).

```
Duration of construction is 0 minutes and 55 seconds
Minimum value is 62611
```

Figura A.4 : esempio di file di output indicante il tempo e il valore minimo calcolato con il tabu search con intorno Relocate