UNIVERSITÀ DEGLI STUDI DI PARMA
FACOLTÀ DI INGEGNERIA
Corso di Laurea Magistrale in Ingegneria Informatica

Corso di Ricerca Operativa

Docente: Prof. MARCO LOCATELLI

Anno Accademico: 2010-2011

ANALISI DI ALCUNI ALGORITMI DI RISOLUZIONE PER IL PROBLEMA VRPTW

Studenti:
ALESSANDRO COSTALUNGA
DAVIDE VALERIANI

Introduzione

Il Vehicle Routing Problem (VRP) è una classe di problemi NP-completi della ricerca operativa caratterizzati da un numero di clienti da servire (raggiungere) mediante una flotta di veicoli. L'obiettivo dei problemi VRP è quello di minimizzare un costo associato al percorso del veicolo, come la distanza percorsa (quindi il carburante utilizzato) oppure il tempo impiegato, oppure quello di minimizzare il numero di veicoli utilizzati. Questa classe di problemi è molto utilizzata nel campo dei trasporti, della distribuzione e della logistica.

Il Vehicle Routing Problem with Time Windows (VRPTW) è un caso particolare dei problemi VRP in cui ogni cliente è caratterizzato da una finestra temporale che il veicolo che consegna il prodotto deve necessariamente rispettare. Un esempio classico in cui applicare questo problema è la consegna di pacchi da parte di un corriere espresso, il quale deve, da un lato minimizzare i costi di trasporto al fine di massimizzare il profitto, dall'altro rispettare le indicazioni fornitegli dai clienti sull'orario di consegna di ciascun pacco.

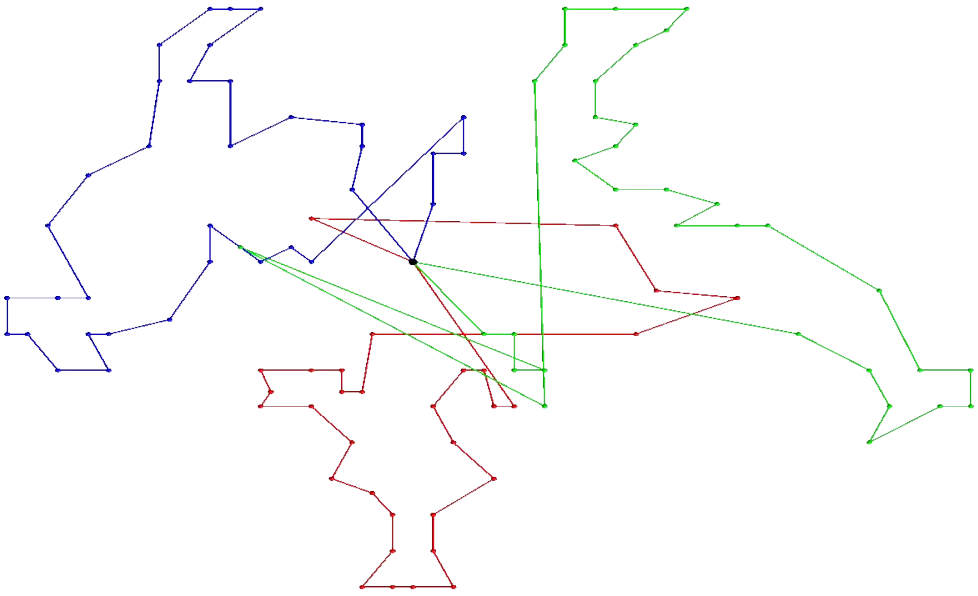

Figura 1: Rappresentazione grafica della soluzione di un problema VRPTW

In questa relazione verranno analizzati diversi algoritmi che propongono una soluzione approssimata al problema, in quanto la soluzione ottima è molto difficile, se non impossibile, da trovare a causa degli elevati costi computazionali. Per visualizzare graficamente i risultati, ci si avvarrà del software di simulazione opensource VRP Simulator.

1 Il software di simulazione VRP Simulator

L'interfaccia grafica di VRP simulator si presenta molto semplice (vedi fig. 2). La risoluzione di un problema di VRPTW è possibile in tre passi:

1. **caricare l'istanza**: dal menu *File* è possibile caricare un'istanza personalizzata oppure sceglierne una tra quelle precaricate;
2. **selezionare l'algoritmo di risoluzione**, dal menu a tendina;
3. **avviare il calcolo** della soluzione approssimata, premendo il pulsante *calculate*.

Dopo qualche istante di attesa, la soluzione numerica apparirà nella casella di testo *result* e, nel riquadro bianco sottostante, apparirà una rappresentazione grafica della soluzione stessa, in cui colori diversi indicano veicoli diversi.

Nel caso si volesse definire un'istanza personalizzata, è sufficiente creare un file di testo strutturato come segue:

- nella prima riga va indicato il nome dell'istanza;
- nella sezione dichiarativa dei veicoli, identificata dalla parola *VEHICLE*, occorre dichiarare, in formato tabellare, il numero di veicoli e la capacità di ogni veicolo;
- nella sezione dichiarativa dei clienti, identificata dalla parola *CUSTOMER*, occorre dichiarare, sempre in formato tabellare, il numero ordinale *CUST NO.* del veicolo partendo da 0 (ovvero il punto di partenza dei veicoli), le coordinate *XCOORD.* e *YCOORD.* sul piano cartesiano di dove si trova il cliente, la richiesta *DEMAND* di prodotto, il tempo di inizio *READY TIME* e di fine *DUE DATE* della finestra temporale e un tempo di servizio *SERVICE TIME*.

Un esempio di file di testo contenente un'istanza per il problema VRPTW è mostrato in fig. 3.

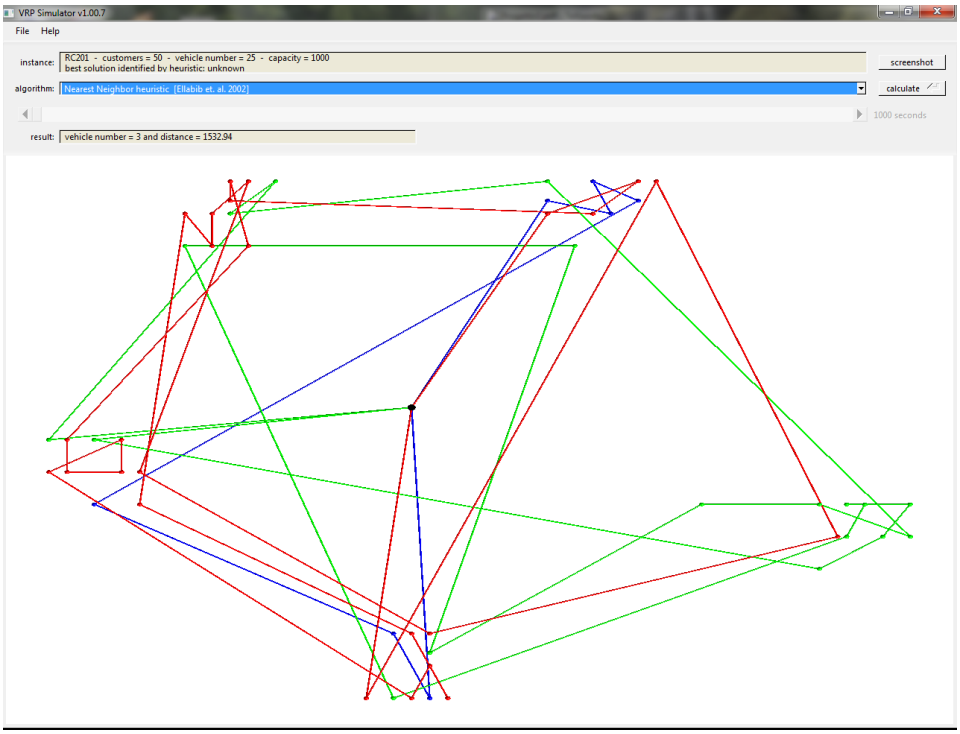

Figura 2: L'interfaccia grafica di VRP Simulator

C101

VEHICLE
NUMBER CAPACITY
5 100

CUSTOMER

CUST NO.	XCOORD.	YCOORD.	DEMAND	READY TIME	DUE DATE	SERVICE TIME
0	40	50	0	0	1236	0
1	45	68	10	912	967	90
2	45	70	30	825	870	90
3	42	66	10	65	146	90
4	42	68	10	727	782	90
5	42	65	10	15	67	90
6	40	69	20	621	702	90
7	40	66	20	170	225	90
8	38	68	20	255	324	90
9	38	70	10	534	605	90
10	35	66	10	357	410	90

Figura 3: Esempio di istanza dichiarata mediante file di testo

2 Il problema VRPTW

Il problema può essere definito come un Grafo $G(V, E)$ dove i vertici che appartengono all'insieme $V(v_1, v_2, ..., v_N)$ rappresentano i clienti, mentre gli archi che appartengono all'insieme E rappresentano la distanza metrica tra due clienti. Il primo nodo v_0 rappresenta il deposito, cioè il nodo di partenza e di arrivo di tutti i veicoli. Per ogni nodo occorre definire:

- la finestra temporale, formata da due valori temporali, e_i e l_i, che indicano la disponibilità temporale del nodo;
- il tempo necessario per servire il nodo, s_i;
- la quantità di merci da scaricare in quel nodo, q_i.

Per ogni arco si deve indicare il tempo di percorrenza necessario per percorrerlo, indicato con t_{ij}. Occorre inoltre definire:

- la quantità massima di merce Q che possono portare i veicoli;
- il numero di veicoli m da utilizzare, anche se spesso è il parametro principale da ottimizzare.

I vincoli principali sono:

- Ogni cliente è servito da un solo veicolo.
- La quantità di beni che il veicolo depositerà dovrà essere minore o uguale alla sua capacità:

$$\sum q_i \leq Q \tag{1}$$

- Il rispetto delle finestre temporali:

$$b_j = MAX\{e_j, b_i + s_i + t_{ij}\} \tag{2}$$

$$b_j + s_j < l_j \tag{3}$$

dove b_j indica il tempo effettivo per accedere al cliente j, da parte di un veicolo.

Ovviamente questi sono i vincoli di base che definiscono il problema standard, ma si potrebbero aggiungere ulteriori vincoli per delineare sottoproblemi più complessi del VRPTW.

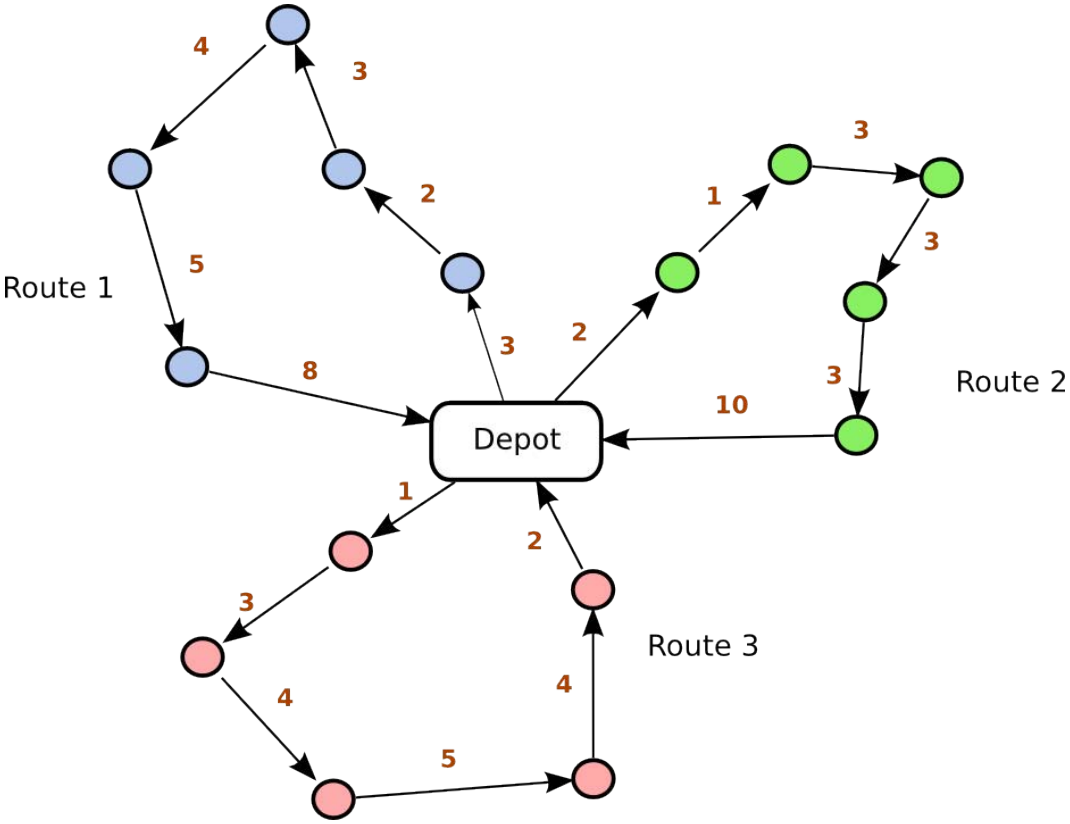

Figura 4: Architettura grafica di un problema VRPTW

Gli obiettivi da ottimizzare, generalmente, sono più di uno, e sono ordinati gerarchicamente per ordine di importanza. Tutti gli algoritmi presentati di seguito considerano come obiettivo principale la minimizzazione del numero di autoveicoli utilizzati e, come secondario, la minimizzazione della distanza totale percorsa dagli autoveicoli o del costo totale associato ai percorsi; tuttavia, in base al tipo di algoritmo, possono esservi altri parametri aggiunti o anche variazioni nel calcolo del costo totale.

3 L'algoritmo di Solomon

L'obiettivo di Solomon è il confronto dei risultati di quattro algoritmi euristici, tra cui uno di questi utilizzato nel programma VRP-Simulator, che si analizzerà in seguito, partendo da un'euristica comune. Il confronto tra essi tiene conto in modo gerarchico dei seguenti obiettivi: riduzione numero veicoli, tempo di risoluzione, costo totale dei cammini che comprende distanza totale e tempo d'attesa totale.

Le classi di algoritmi trattate da Solomon sono due: metodologie sequenziali, che creano un percorso alla volta finché non si soddisfano tutti i clienti, e metodologie parallele, dove più percorsi vengono generati simultaneamente.

La definizione del problema del VRPTW coincide con quella generale descritta precedentemente, con l'aggiunta di un coefficiente di costo per ogni arco, indicato dalla formula seguente

$$c_{ij} = \rho_1 \cdot d_{ij} + \rho_2(b_j - b_i) \tag{4}$$

$$\rho_1, \rho_2 \geq 0 \tag{5}$$

dove $b_j - b_i$ è la differenza tra i tempi di arrivo tra i due clienti i e j, mentre ρ_1 e ρ_2 sono i coefficienti di peso della funzione, per poter sbilanciare l'ottimizzazione sui tempi d'attesa o sulla distanza totale.

3.1 Euristica

Anche se Solomon ha analizzato diversi algoritmi, tuttavia la struttura generale di questi può essere espressa come segue: per prima cosa, l'algoritmo costruisce una serie di cammini, garantendo che i percorsi parziali in costruzione siano corretti, cioè rispettino i vincoli temporali imposti. Ad esempio, nel percorso I, formato da gli elementi i_p con $1 \leq p \leq m$, (dove i_1 e i_m coincidono con il deposito), si inserisce il cliente u, tra i_{p-1} e i_p, dove i tempi di inizio del servizio b_i sono conosciuti.

Si assume, inoltre, che tutti i veicoli partano al tempo e_0, poi successivamente verrà ottimizzato il tempo d'attesa facendo partire i veicoli in tempi opportuni, in modo da minimizzare anche il tempo d'attesa.

Il tempo di servizio di i_p cambia con l'aggiunta di un nuovo cliente che lo precede; indichiamo con w_{ir} il tempo d'attesa per i clienti successivi al cliente appena immesso nel percorso.

Se è stata assunta la disuguaglianza triangolare sia nelle distanze che nei tempi di percorrenza, l'inserimento di u permette di *traslare in avanti* l'orario di i_p. Valgono pertanto le seguenti equazioni.

$$PF_i = b_{ip}^{new} - b_{ip} \geq 0 \tag{6}$$

$$PF_{ir+1} = MAX\{0, PF_{ir} - w_{ir+1}\}, p \leq r \leq m-1 \tag{7}$$

Da notare che si traggono molti vantaggi nel partire inizialmente dal deposito a e_0. Esiste anche il concetto di *traslazione all'indietro*, anche se non è trattato in questa analisi.

Se $PF_{ip} > 0$, molti dei clienti successivi a quello inserito potrebbero diventare non ammissibili, quindi a meno che $PF_{ip} = 0$, bisogna controllare i clienti successivi, in modo che il veicolo che segue il tragitto non sfori il tempo massimo.

Lemma: Sono condizioni necessarie e sufficienti per inserire un cliente u tra i_{p-1} e i_p, su un cammino parzialmente costruito:

$$b_u \leq l_u \tag{8}$$

$$b_{ir} + PF_{ir} \leq l_{ir} \tag{9}$$

Notare che se il cliente aggiunto non permette di far raggiungere il deposito nel tempo l_0, allora il cliente non viene aggiunto nel percorso parziale.

3.2 Nearest-Neighbor Heuristic

L'algoritmo, euristico di classe sequenziale, inizia ogni percorso trovando il cliente più vicino al deposito, in termini di tempo e costo. Ad ogni iterazione si cercano i clienti che non fanno parte di un percorso per aggiungerli a questo; la ricerca riguarda tutti i clienti che riescono a mantenere il percorso entro i vincoli temporali.

Ogni volta che la ricerca fallisce si crea un nuovo percorso, finché tutti i clienti non appartengono ad uno dei percorsi. La tipologia di ricerca del *vicino più vicino* tiene conto sia della distanza fisica che di quella temporale dei clienti.

Possiamo indicare l'ultimo cliente di un percorso parziale come i, mentre con j possiamo indicarne uno che non fa parte di nessun percorso e potrebbe aspirare a diventare parte del percorso di i. La scelta del nuovo nodo j viene affidata al nodo che ha costo c_{ij} minore rispetto al nodo i.

Il parametro di costo è combinazione dei seguenti parametri:

- d_{ij}, che indica la distanza fisica tra i due nodi;
- T_{ij} che indica la differenza tra l'accesso al cliente j e il completamento del servizio di i;
- v_{ij} che indica l'urgenza di consegna di j, cioè il tempo che rimane prima che il cliente j termini.

Le equazioni che caratterizzano questo algoritmo sono pertanto le seguenti.

$$T_{ij} = b_j - (b_i + s_i) \tag{10}$$

$$v_{ij} = l_j - (b_i + s_i + t_{ij}) \tag{11}$$

$$c_{ij} = \delta_1 \cdot d_{ij} + \delta_2 \cdot T_{ij} + \delta_3 \cdot v_{ij} \tag{12}$$

$$\delta_1 + \delta_2 + \delta_3 = 1 \tag{13}$$

$$\delta_1, \delta_2, \delta_3 \geq 0 \tag{14}$$

Quest'algoritmo è quello utilizzato nel programma VPR-Simulator, anche se Solomon analizza anche altri tre algoritmi: *Savings Heuristics*, *Inseretion Heuristic* e *Time-Oriented Sweep Heuristic*, non analizzati in questa trattazione.

4 L'algoritmo di Taillard

L'algoritmo di Taillard si basa sulla suddivisione del problema VRPTW in Soft (VRPSTW) e Hard (VRPHTW).

Nel problema VRPSTW le finestre temporali sono rilassate: questo significa che i veicoli possono arrivare da un cliente prima dell'inizio o dopo la fine della sua finestra temporale. Se il veicolo arriva troppo presto, aspetterà l'inizio della finestra temporale, mentre se arriva troppo tardi, gli viene inferta una penalità. Questo consente, ovviamente, di avere un buon numero di soluzioni. Il VRPSTW tende ad ottimizzare la distanza totale mantenendo fissato il numero di veicoli, tentando di rispettare il più possibile i vincoli temporali, anche se non vengono scartati percorsi che li violano. Rispetto alla definizione del problema data precedentemente, i vincoli temporali non sono più veri vincoli, ma diventano funzioni di penalità, che bisogna aggiungere ai parametri da ottimizzare. L'obiettivo è minimizzare la funzione seguente.

$$f(s) = \sum_{k=1}^{m} d_k + \sum_{i=1}^{n} a_i \cdot max(0, t_i - l_i), s \in S \tag{15}$$

Il problema VRPHTW è, invece, quello che è stato definito precedentemente: i vincoli temporali sono intransigibili, per prima cosa si deve minimizzare il numero di veicoli e, successivamente, la distanza totale. Tale problema può essere visto come un sotto problema del VRPSTW.

Il metodo di ottimizzazione di Taillard riprende un algoritmo meta-euristico chiamato *Tabu Search*, che consiste nella ricerca alternata tra due vicini, attuando scambi dei cammini tra i clienti e includendo procedure di eliminazione di cammini aventi pochi clienti da servire; inoltre viene utilizzate una memoria adattiva per il salvataggio dei migliori risultati durante la ricerca.

4.1 La struttura di vicinato

Taillard definisce una nuova struttura di vicinato accoppiata con un metodo di approssimazione per la valutazione di ogni soluzione in tempo invariante.

La nuova struttura di vicinato viene costituita inizialmente da scambio di archi euristico: con una ricerca locale si cerca una soluzione, poi, applicando tutte le modifiche associate ad un determinato metodo, si crea un intorno di soluzioni rispetto a quella trovata precedentemente; infine, si seleziona la soluzione migliore in quest'intorno di soluzioni. Il cambio euristico, chiamato *CROSS-EXCHANGE*,

consiste nello scambio degli archi tra i vari percorsi in modo da rispettare le time windows e riuscire a migliorare la funzione obbiettivo. Questa procedura ha una elevata complessità:

$$O\left(\frac{n^4}{m^2}\right) \tag{16}$$

dove n indica il numero di clienti e m il numero di cammini totali.

Ad esempio, come mostrato nella fig. 5, i quadrati neri rappresentano il deposito mentre i cerchi bianchi i clienti lungo i cammini. I due archi (X_1, X_1') e (Y_1, Y_1') sono rimossi dal primo percorso mentre (X_2, X_2') e (Y_2, Y_2') sono rimossi dal secondo percorso. Quindi, i segmenti contenenti un vario numero di clienti X_1' - Y_1 e X_2' - Y_2 sono scambiati introducendo due nuovi archi: (X_1, X_2'), (Y_2, Y_1'), (X_2, X_1'), (Y_1, Y_2'). Da notare che i vincoli temporali definiscono implicitamente l'orientamento del percorso, e quindi i segmenti hanno sempre lo stesso orientamento dopo questa mossa di scambio.

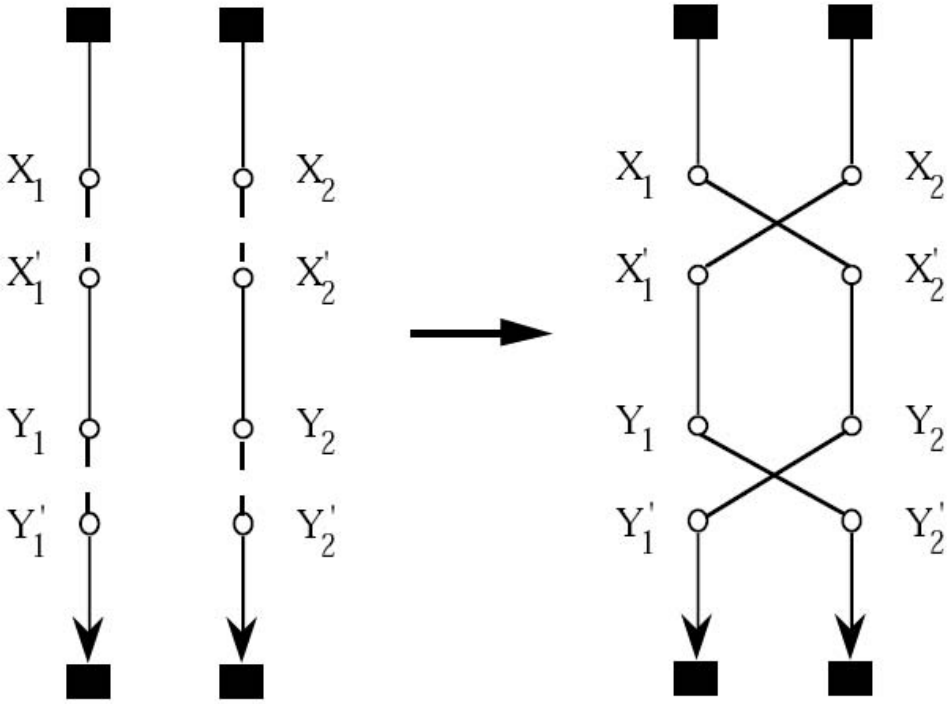

Figura 5: Esempio di struttura di vicinato

A questo punto, vengono creati i dintorni: prima si valuta il *movimento*, vedendo la differenza tra il valore di una soluzione vicina con quella della soluzione corrente: se ha valore negativo significa che questo *movimento* porta benefici che sono facilmente visibili nella differenza delle distanze totali ottenute tra le due soluzioni. I

movimenti che non portano benefici devono essere eliminati diminuendo la dimensione dei dintorni. Il tempo di computazione può essere inoltre ridotto utilizzando una matrice di approssimazione, che conterrà informazioni sui miglior *CROSS-EXCHANGE* per ogni coppia di cammino, come il valore della nuova soluzione e quali archi sono stati immessi o rimossi dalla coppia per arrivare a questa soluzione. Infine, oltre ad avere scambi di archi tra due rotte, vi sono da considerare gli scambi all'interno di un singolo cammino; le modalità di scambio avvengono similmente al *CROSS-EXCHANGE* ed anch'esse sono molto importanti per il miglioramento della soluzione.

4.2 Metodologia

La metodologia utilizzata per la risoluzione del problema VRPTW è la seguente: si costruiscono inizialmente P soluzioni differenti, utilizzando l'algoritmo *Insertion Heuristic* di Solomon (nel caso di VRP-simulator, viene utilizzata l'euristica *Nearest-Neighbor Heuristic*). A queste soluzioni viene applicata l'euristica *Tabu Search* ed i cammini risultanti vengono salvati nella memoria adattiva. Viene poi presa una soluzione nella memoria adattiva e viene definita come la soluzione corrente. Questa viene decomposta in C sottoinsiemi di percorsi (vedi fig. 6) e ad ognuno di essi viene applicata l'euristica *Tabu Search*.

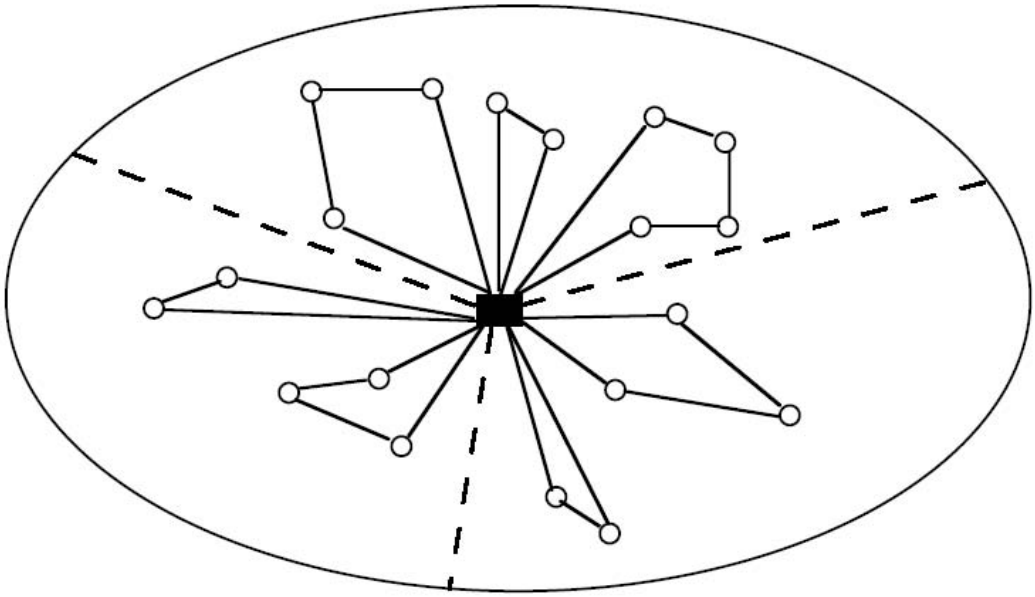

Figura 6: Funzionamento dell'euristica Tabu Search

Una volta ottimizzati, i vari sottoinsiemi di percorsi vengono rimessi insieme formando una soluzione completa, che viene risalvata nella memoria adattiva. Itera-

tivamente vengono prese tutte le soluzioni salvate nella memoria finché non si soddisfa un determinato criterio. Infine, si applica una procedura di postottimizzazione per ogni cammino della migliore soluzione.

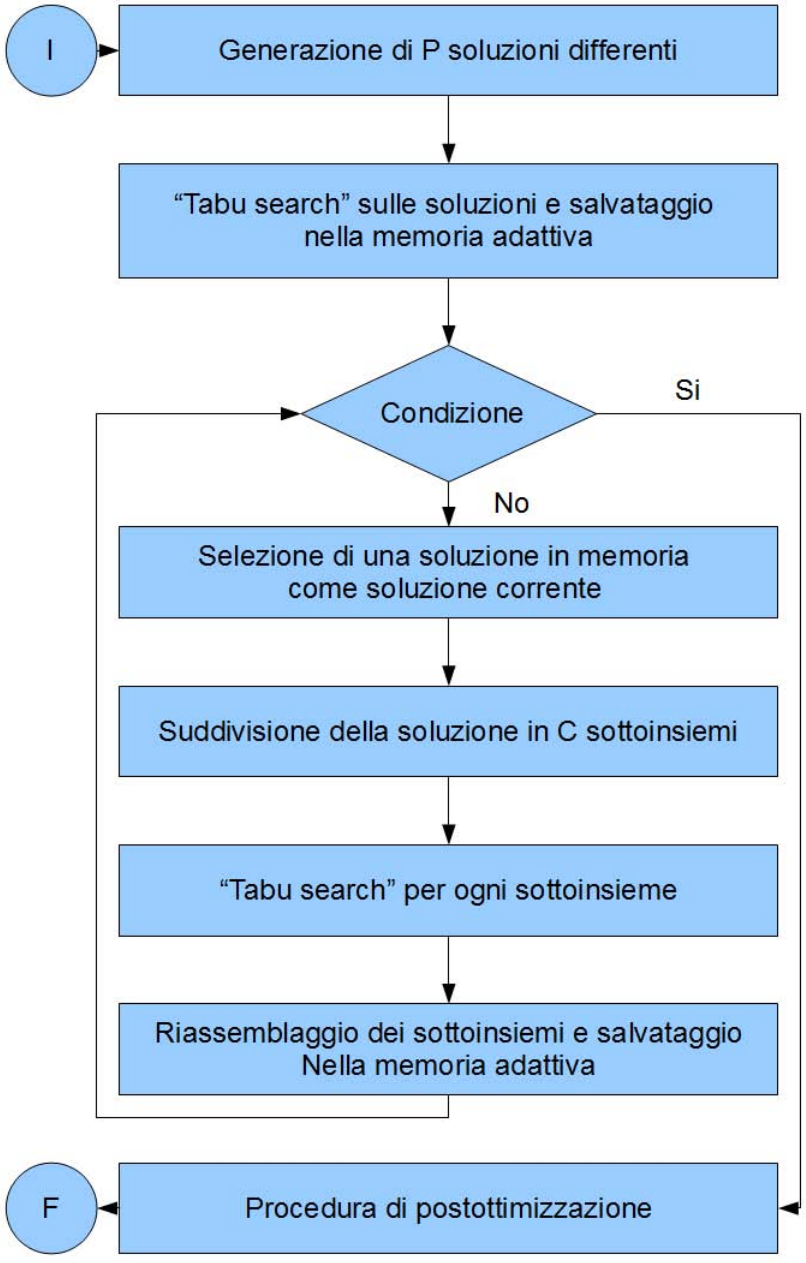

Figura 7: Diagramma di flusso per la risoluzione del problema VRPTW

L'euristica *Tabu Search* seleziona la soluzione corrente come alcuni sottoinsiemi di cammini iniziali e genera il vicinato della soluzione corrente applicando *CROSS-EXCHANGE*. Viene poi selezionata la miglior soluzione del vicinato, che diventerà

la soluzione corrente. Se questa soluzione è migliore di quella generale, si riordineranno i clienti di ogni percorso utilizzando l'algoritmo *Insertion Heuristic* e si la definisce come miglior soluzione globale. Infine, si aggiorna la *Tabu List*, che non è altro che una lista di lunghezza T che contiene in ogni posizione una soluzione. La posizione è il valore obiettivo della soluzione, mentre il valore salvato nella posizione è il numero di iterazione dopo la quale la soluzione perde il suo stato di tabu. Se il valore trovato in questa posizione è più grande del valore corrente, il movimento verrà considerato tabu, altrimenti verrà accettato.

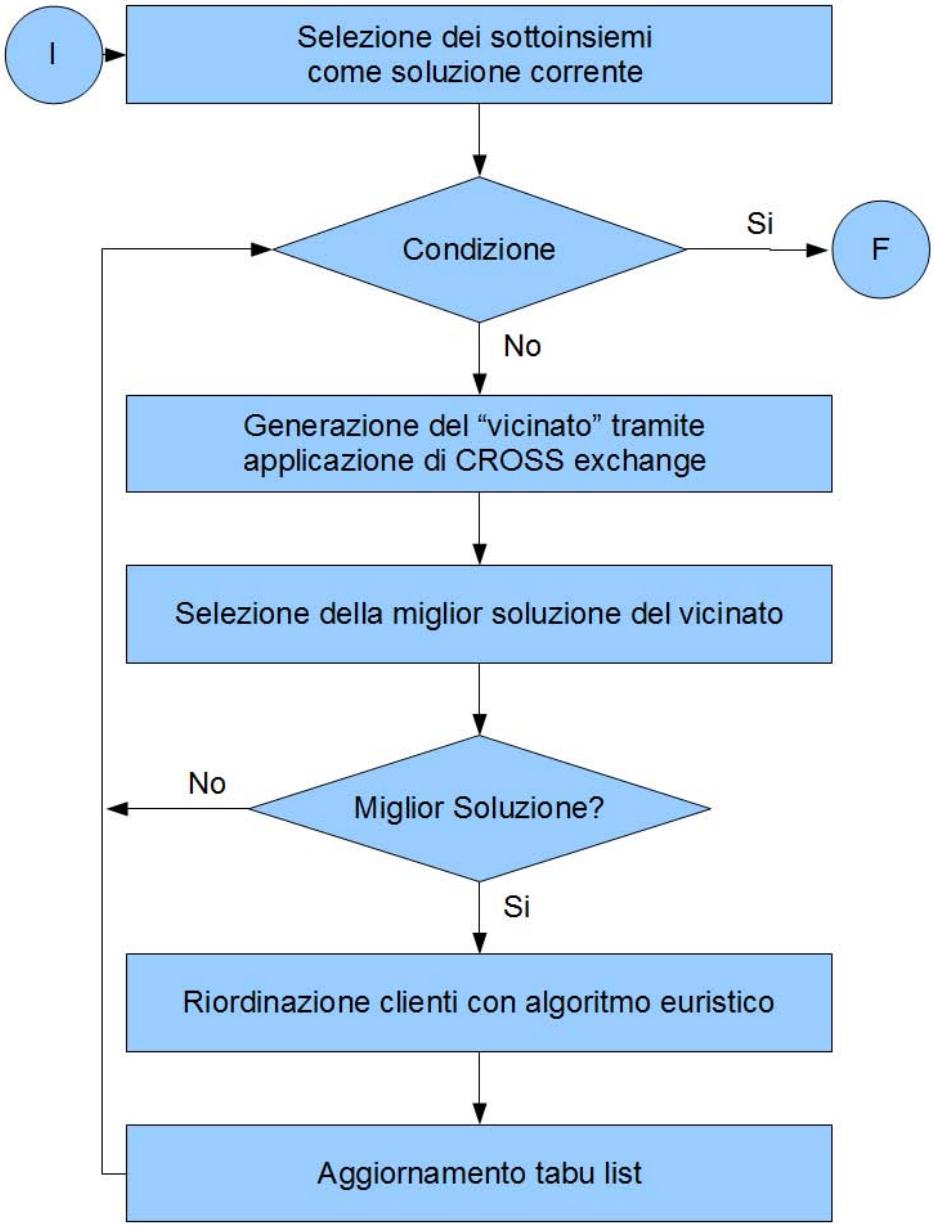

Figura 8: Diagramma di flusso dell'euristica Tabu Search

5 L'algoritmo MACS-VRPTW

Questo algoritmo è basato sulla metaeuristica ACO (Ant Colony Optimization) che si ispira al comportamento di vere colonie di formiche nell'approvvigionamento del cibo, le quali collaborano tra loro mediante il rilascio di una sostanza, il feromone, lungo il loro percorso per tenere traccia dei luoghi già visitati e della posizione del cibo.
Una delle implementazioni più efficienti basate su ACO è la ACS (Ant Colony System) che introduce una particolare procedura di aggiornamento dei percorsi di feromone per intensificare la ricerca nelle vicinanze della soluzione migliore trovata fino all'istante corrente. Il MACS-VRPTW (Multiple Ant Colony System - Vehicle Routing Problem with Time Windows) è un'estensione dell'ACS particolarmente indicata per risolvere problemi VRPTW.
Il VRPTW considerato da questo algoritmo minimizza una funzione obiettivo multipla: per prima cosa viene minimizzato il numero di veicoli utilizzati, successivamente viene minimizzato il tempo di viaggio totale.

Per adattare ACS a questo obiettivo multiplo, l'algoritmo definisce due diverse colonie ACS che ottimizzano, rispettivamente, il numero di veicoli utilizzati (ACS-VEI) e il tempo totale di viaggio (ACS-TIME), come mostrato in fig. 9. Entrambi gli obiettivi vengono ottimizzati contemporaneamente coordinando le attività delle due colonie attraverso l'aggiornamento di una variabile condivisa ψ_{gb} che inizialmente viene posta uguale a una soluzione trovata da un'euristica.
Una volta attivata, ACS-VEI cerca una soluzione che utilizzi un veicolo in meno rispetto al numero di veicoli utilizzato nella soluzione ψ_{gb}. L'obiettivo di ACS-TIME è invece quello di ottimizzare il tempo totale di viaggio per soluzioni che utilizzino lo stesso numero di veicoli della soluzione ψ_{gb}. La variabile ψ_{gb} viene poi aggiornata ogni volta che una colonia trova una soluzione migliore. Nel caso questa nuova soluzione contenga un numero inferiore di veicoli rispetto alla precedente, le due colonie ACS-VEI e ACS-TIME vengono distrutte, la variabile viene aggiornata e, successivamente, il processo itera con la creazione di due nuove colonie.

L'algoritmo termina al verificarsi di una particolare condizione che, nel caso del software di simulazione, è il superamento del tempo assegnato alla computazione.

5.1 La procedura della colonia ACS-TIME

L'obiettivo di questa colonia è minimizzare la lunghezza del percorso totale. Per fare questo, vengono attivate m formiche artificiali (veicoli) che trovano un insieme di

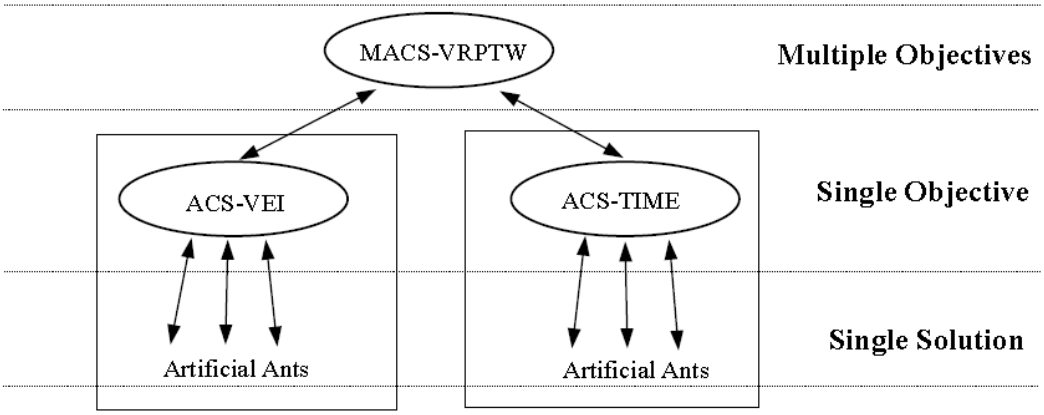

Figura 9: Architettura del MACS-VRPTW

soluzioni $\psi_1, ..., \psi_m$ mediante m chiamate alla procedura *new_active_ant* descritta in seguito. Queste soluzioni vengono poi confrontate con la soluzione ψ_{gb} comune alle due colonie e, nel caso una si migliore (ψ_i), viene spedita al problema principale MACS-VRPTW per l'aggiornamento della soluzione ψ_{gb}. Dopo la generazione dell'insieme di soluzioni $\psi_1, ..., \psi_m$, viene eseguito l'aggiornamento globale dei tempi di percorrenza secondo la formula

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \rho / J_{\psi_{gb}} \tag{17}$$

in cui ρ è un parametro tale che $0 \leq \rho \leq 1$ e $J_{\psi_{gb}}$ è la lunghezza della soluzione ψ_{gb}.

5.2 La procedura della colonia ACS-VEI

L'obiettivo di questa colonia è quello di cercare una soluzione ammissibile massimizzando il numero di clienti serviti e utilizzando $v - 1$ veicoli, dove v è il numero di veicoli utilizzati nella soluzione ψ_{gb}. La soluzione (anche non ammissibile) generata da questa procedura con il maggior numero di clienti visitati viene posta nella variabile $\psi_{ACS-VEI}$.
Al fine di massimizzare il numero di clienti serviti, la procedura ACS-VEI si avvale di un vettore di interi IN, in cui il generico elemento IN_j contiene il numero di volte che il cliente j non è stato inserito nella soluzione. Questo vettore viene utilizzato nella procedura *new_active_ant* per privilegiare i clienti che sono stati spesso esclusi dalle soluzioni generate. Alla fine di ogni ciclo, i percorsi di feromone vengono aggiornati utilizzando due soluzioni: $\psi_{ACS-VEI}$, ovvero la soluzione non ammissibile con il maggior numero di clienti visitati, e ψ_{gb}, ovvero la soluzio-

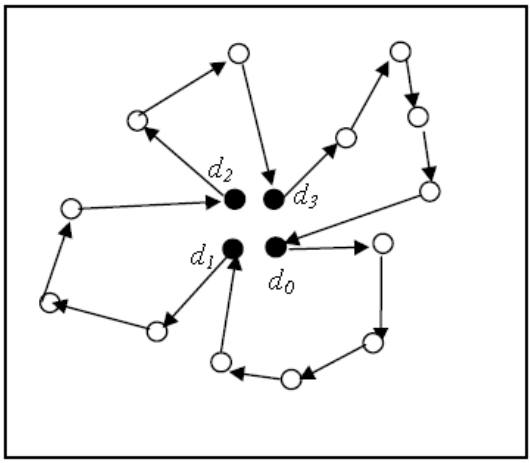

Figura 10: Esempio di soluzione ammissibile

ne ammissibile con il minor numero di veicoli e il minor tempo totale di viaggio calcolata fino all'istante corrente.

5.3 La procedura *new_active_ant*

Prima di descrivere la procedura *new_active_ant*, che si occupa di costruire una soluzione ammissibile per una formica, occorre conoscere la struttura del modello utilizzato per la ricerca di una soluzione. Per prima cosa, il nodo v_0 che rappresenta il deposito viene duplicato un numero di volte pari al numero m di formiche (veicoli) disponibili; la distanza tra le copie del deposito viene posta uguale a 0. Successivamente, si identifica una soluzione ammissibile come un cammino che visiti tutti i nodi del grafo una e una sola volta (un esempio di soluzione ammissibile è raffigurato in fig. 10).

Nella procedura *new_active_ant*, ogni formica parte da una copia scelta a caso del deposito e, a ogni passo, si muove verso un nodo non ancora visitato che non violi i vincoli del problema (time windows e capacità del veicolo). L'insieme dei nodi disponibili, nel caso in cui la formica non sia posizionata su un duplicato del deposito, include anche i duplicati del deposito non ancora visitati. Una formica posizionata nel nodo i sceglie probabilisticamente il successivo nodo j usando i meccanismi di exploration e exploitation. L'attrattività η_{ij} di un nodo è calcolata tenendo conto del tempo di viaggio t_{ij} tra i nodi i e j, della finestra temporale $[b_j, e_j]$ associata al nodo j del numero di volte IN_j che il nodo j è stato escluso da una soluzione del problema.

Ogni volta che una formica si muove da un nodo a un altro, viene eseguito un

aggiornamento locale del percorso di feromone secondo la formula

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \rho \cdot \tau_0 \tag{18}$$

dove τ_0 è il valore iniziale del percorso.

Le soluzioni così costruite, tuttavia, potrebbero essere incomplete in quanto alcuni clienti sono stati omessi: la soluzione viene quindi completata con ulteriori inserzioni. Ogni inserimento viene effettuato considerando tutti i clienti non visitati ordinati in modo decrescente per quantità da consegnare. Per ogni cliente viene ricercata la soluzione ammissibile migliore, ovvero quella con tempo di viaggio più piccolo, finché non è più possibile inserire nuovi nodi.

6 L'algoritmo EAMA di Nagata

L'algoritmo EAMA (Edge Assembly Memetic Algorithm) proposto da Nagata minimizza il numero di veicoli e la distanza totale percorsa in modo indipendente. Nel primo stadio, viene determinato il numero minimo m di veicoli necessari mediante l'euristica RM (Route Minimization) eseguita per un dato periodo di tempo. Successivamente, viene minimizzata la distanza totale percorsa mediante un ciclo che genera varie soluzioni. Ogni soluzione viene selezionata come genitore p_A e come genitore p_B in un ordine casuale. Poi, per ogni coppia di genitori p_A e p_B, l'operatore di crossover EAX genera le soluzioni discendenti N_{ch}: se queste violano i vincoli di capacità e/o di tempo (time window), viene invocata la procedura di riparazione, altrimenti, se le soluzioni sono ammissibili, vengono ulteriormente migliorate dal punto di vista della distanza percorsa mediante un'operazione di ricerca locale. Infine, se una soluzione discendente N_{ch} è migliore di p_A, p_A viene aggiornata con questa soluzione.

La funzione di costo di questo algoritmo è mostrata dall'equazione 19 ed è composta dalla distanza totale percorsa $F(\sigma)$ e dai termini penalizzanti $P_c(\sigma)$ e $P_{tw}(\sigma)$ che identificano rispettivamente la violazione dei vincoli di capacità e la violazione di quelli delle time window, moltiplicati per i coefficienti di penalità α e β.

$$F_g(\sigma) = F(\sigma) + \alpha \cdot P_c(\sigma) + \beta \cdot P_{tw}(\sigma) \tag{19}$$

6.1 L'operatore EAX

L'operatore EAX (Edge Assembly Crossover), indicato con $EAX(p_A, p_B)$, consiste in cinque step illustrati nella figura 11. Nella procedura, p_a e p_b si riferiscono alle soluzioni dei genitori, che hanno entrambe m veicoli.

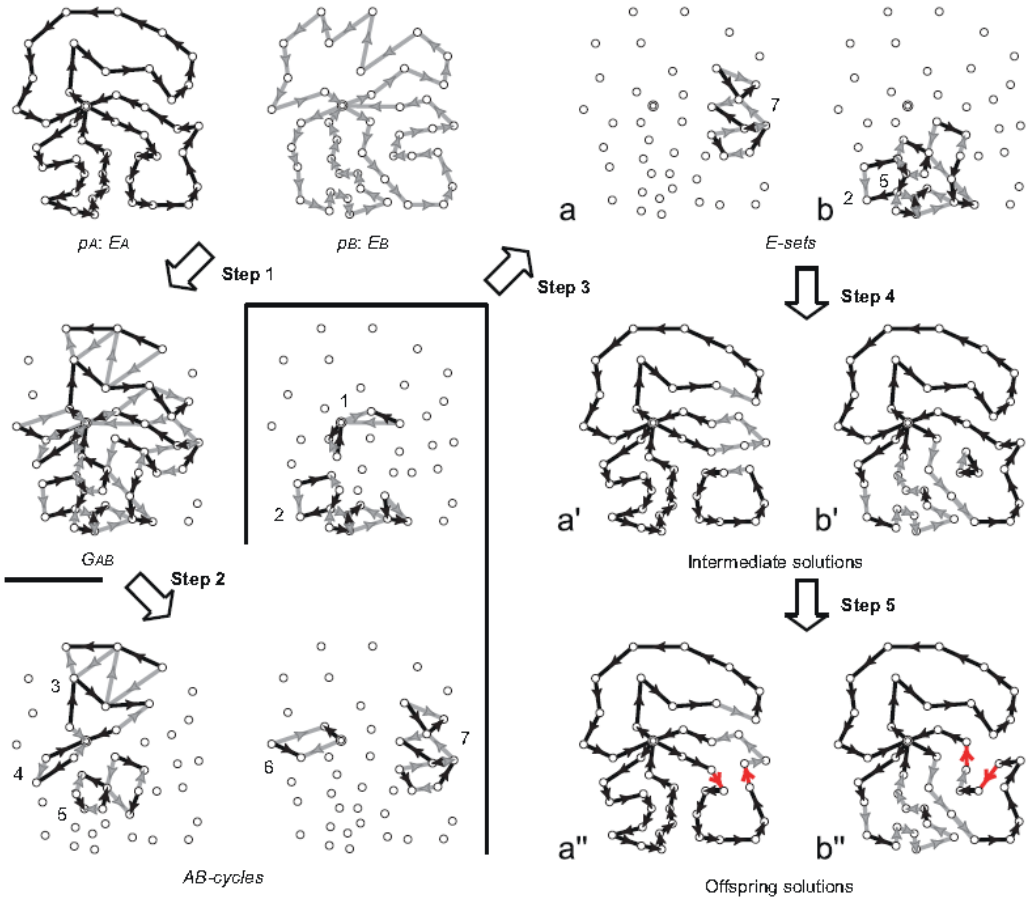

Figura 11: Illustrazione degli step dell'operatore EAX

Nel primo step, viene definito il grafo $G_{AB} = (V, E_A \cup E_B \setminus E_A \cap E_B)$, dove E_A e E_B sono gli insieme di archi dei genitori p_A e p_B.

Nel secondo step, tutti gli archi vengono divisi in *cicli AB*. Un *ciclo AB* è un ciclo del grafo G_{AB} in cui gli archi di p_A e p_B sono collegati alternativamente con orientazione opposta. Questi cicli sono formati selezionando un nodo iniziale a caso e tracciando gli archi forward verso p_A e gli archi backward verso p_B finché non viene trovato un ciclo AB. Ogni volta che viene trovato un ciclo AB, gli archi che lo compongono vengono eliminati da G_{AB} e si ricomincia a cercare un ciclo AB nel grafo risultante. La ricerca termina quando tutti gli archi di G_{AB} sono stati eliminati.

Nel terzo step, vengono costruiti i cosiddetti *insiemi E* combinando cicli AB. Per la selezione dei cicli AB si possono scegliere due strategie:

- *strategia singola*: viene selezionato un ciclo AB a caso e viene definito come insieme E;

- *strategia a blocchi*: viene inizialmente selezionato un ciclo AB a caso e, successivamente, si selezionano quei cicli che condividono almeno un nodo con il primo ciclo e contengono meno nodi del primo ciclo.

Nel quarto step, p_A viene selezionata come soluzione di base. Successivamente, viene selezionato un insieme E (E_1) e viene costruita una soluzione intermedia, partendo da quella base, rimuovendo archi presenti in $E_1 \cap E_A$ e aggiungendo archi presenti in $E_1 \cap E_B$. Ogni soluzione intermedia consiste in m veicoli che partono dal deposito e, eventualmente, da uno o più sottopercorsi.

Nell'ultimo step, i sottopercorsi vengono connessi a percorsi esistenti uno alla volta e in ordine casuale, creando le soluzioni discendenti. Viene selezionato e rimosso un arco da rimuovere comune tra il sottopercorso e il percorso esistente, mentre vengono aggiunti due nuovi archi per connettere i due percorsi tra loro, creandone uno unico.

7 Prove sperimentali

Al fine di testare l'efficienza degli algoritmi analizzati in precedenza, si sono scelte alcune istanze precaricate all'interno del programma VRP simulator e si sono tabulati i risultati sperimentali.

Le prove sono state suddivise in tre blocchi, a seconda della difficoltà. Ogni blocco di prove conta tre misurazioni riguardanti la distribuzione dei clienti:

- **C** indica una distribuzione clusterizzata;
- **R** indica una distribuzione casuale;
- **RC** indica una combinazione di distribuzione clusterizzata e casuale.

Per ogni istanza, sono presenti tre colonne:

- **V** indica il numero di veicoli della soluzione trovata;
- **D** indica la lunghezza totale del percorso nella soluzione trovata;
- **T** indica, in prima approssimazione, il tempo impiegato dall'algoritmo per trovare la soluzione, espresso in secondi.

Gli algoritmi utilizzati sono:

- **Solomon**: Nearest Neighbor heuristic [Solomon 1987];
- **Taillard**: Nearest Neighbor heuristic [Solomon 1987] + INTRA-AND-CROSS-EXCHANGE [Taillard et al. in 1997];
- **MACS**: MACS-VRPTW [Gambardella 1999], assegnando rispettivamente 5, 10 e 30 secondi all'elaborazione.

Le prove sono state condotte su un computer portatile Acer Travelmate 6292, dotato di un processore Intel Core 2 Duo 2.2 GHz, 2 GB di memoria RAM e con sistema operativo Microsoft Windows 7 Professional.

Il primo blocco di prove riguarda l'istanza 208 di Solomon [100 customers]. Il numero di veicoli a disposizione è di 25, mentre la capacità massima di trasporto di ogni veicolo è di 700 nel caso dell'istanza C208 e 1000 nelle istanze R208 e RC208.

Nella tabella seguente, sono riassunti i risultati ottenuti.

	C208			RC208			R208		
	V	D	T	V	D	T	V	D	T
Solomon	3	792.87	≈0	4	2356.84	≈0	3	1524.98	≈0
Taillard	3	664.56	≈0	4	1024.11	≈0	3	899.97	≈0
MACS	3	588.49	5	3	1426.08	5	3	1307.81	5
MACS	3	588.88	10	3	1245.06	10	3	971.75	10
MACS	3	688.44	30	3	1163.63	30	3	1307.81	30

Tabella 1. Confronto dei risultati degli algoritmi analizzati con istanza 208

Da questo primo blocco di prove, si nota come, nel caso di istanza C208, la soluzione migliore sia ottenuta con l'algoritmo MACS-VRPTW dopo 5 secondi di elaborazione; nel caso di istanza RC208, la soluzione migliore corrisponda a quella calcolata con l'algoritmo MACS-VRPTW dopo 30 secondi (ricordando che la prima variabile da minimizzare è il numero di veicoli); nel caso di istanza R208, invece, la soluzione migliore sia ottenuta con l'algoritmo di Taillard.

Il secondo blocco di prove riguarda l'istanza 2_6_9 di Homberger [600 customers]. Il numero di veicoli a disposizione è di 150, mentre la capacità massima di trasporto di ogni veicolo è di 700 nel caso dell'istanza C2_6_9 e 1000 nelle istanze R2_6_9 e RC2_6_9.

Nella tabella seguente, sono riassunti i risultati ottenuti.

	C2_6_9			RC2_6_9			R2_6_9		
	V	D	T	V	D	T	V	D	T
Solomon	25	27912.23	≈0.5	16	29424.45	≈0.5	15	38290.06	≈0
Taillard	25	10912.31	≈0.7	16	13538.01	≈1	15	18759.46	≈2
MACS	20	12555.09	5	12	19943.18	5	12	24451.18	5
MACS	20	13030.60	10	12	18264.03	10	12	23672.18	10
MACS	20	11619.93	30	12	20327.73	30	12	21669.51	30

Tabella 2. Confronto dei risultati degli algoritmi analizzati con istanza 2_6_9

Da questo secondo blocco di prove, si nota come, nel caso di istanza C2_6_9, la soluzione migliore sia ottenuta con l'algoritmo MACS-VRPTW dopo 30 secondi di elaborazione; nel caso di istanza RC2_6_9, la soluzione migliore corrisponda a quella calcolata con l'algoritmo MACS-VRPTW dopo 10 secondi; nel caso di istanza R2_6_9, la soluzione migliore sia ottenuta con l'algoritmo MACS-VRPTW dopo 30 secondi di elaborazione.

Il terzo e ultimo blocco di prove riguarda l'istanza 210_9 di Homberger [1000 customers]. Il numero di veicoli a disposizione è di 250, mentre la capacità massima di trasporto di ogni veicolo è di 700 nel caso dell'istanza C210_9 e 1000 nelle istanze R210_9 e RC210_9.

Nella tabella seguente, sono riassunti i risultati ottenuti.

	C210_9			**RC210_9**			**R210_9**		
	V	D	T	V	D	T	V	D	T
Solomon	43	59112.41	$\approx$0.5	22	72693.79	$\approx$0.5	27	96360.14	$\approx$0.5
Taillard	-	-	∞	-	-	∞	27	44753.29	$\approx$4
MACS	34	28362.31	5	20	67815.93	5	20	79046.26	5
MACS	34	30375.49	10	19	74868.03	10	20	69248.73	10
MACS	34	28341.75	30	19	72941.24	30	20	58615.43	30

Tabella 3. Confronto dei risultati degli algoritmi analizzati con istanza 210_9

Da questo ultimo blocco di prove, si nota come, in tutte e tre le istanze, la soluzione migliore sia ottenuta dall'algoritmo MACS-VRPTW dopo 30 secondi di elaborazione. In questa prova, inoltre, relativamente alle istanze C210_9 e RC210_9, non è stato possibile applicare l'algoritmo di Taillard in quanto il programma andava in crash e, pertanto, anche attendendo numerosi minuti, non restituiva mai la soluzione.

8 Conclusioni

Sono stati analizzati alcuni algoritmi per la risoluzione del problema VRPTW. Dalle prove effettuate, si nota come l'algoritmo MACS-VRPTW sia, in generale, il più performante. Infatti, ad eccezione di alcuni casi, seppure impieghi qualche secondo in più di elaborazione, l'algoritmo consente di trovare una soluzione ottima nettamente superiore (in alcuni casi) alle soluzioni trovate dagli altri due algoritmi.

La ricerca nel campo del problema VRPTW, uno dei problemi più complessi ma anche più utilizzati nella pratica, è in continua evoluzione. Pensiamo, ad esempio, ai navigatori satellitari di ultima generazione o ai software di gestione dei corrieri espressi. La ricerca sta lavorando al fine di trovare algoritmi sempre più performanti, sia dal punto di vista della soluzione ottima trovata, sia dal punto di vista del tempo impiegato per trovarla.

In questa trattazione sono stati analizzati soltanto alcuni algoritmi selezionati. Inoltre, il software di simulazione utilizzato, non consentiva la simulazione di istanze di VRPTW utilizzando tutti gli algoritmi (ad esempio, l'algoritmo di Nagata non era supportato). Una possibile evoluzione dello stesso software sarebbe quella di integrare ulteriori algoritmi di simulazione, al fine di poter effettuare un'analisi più approfondita dei metodi di risoluzione del VRPTW.

Un'altra possibile evoluzione potrebbe essere quella di preparare istanze ad-hoc, ricavate da problemi reali, e farle eseguire al simulatore al fine di analizzare quale sia l'algoritmo migliore per quella specifica istanza. Infatti, abbiamo visto che, in un caso, l'algoritmo MACS-VRPTW restituiva una soluzione ottima peggiore rispetto a quello di Taillard.

Riferimenti bibliografici

[1] G. Agazzi L. M. Gambardella, E. Taillard. Macs-vrptw: A multiple ant colony system for vehicle routing problems with time windows. *IDSIA*, 1999.

[2] W. Dullaert Y. Nagata, O. Braysy. A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows. *Elsevier*, 2009.

[3] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Informs*, 1985.

[4] M. Gendreau F. Guertin J. Y. Potvin E. Taillard, P. Badeau. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 1997.

[5] http://vrp.sourceforge.net/.

[6] http://www.idsia.ch/ luca/macs-vrptw/solutions/welcome.htm.