ALMA MATER STUDIORUM – UNIVERSITA' DI BOLOGNA
SEDE DI CESENA

FACOLTA' DI SCIENZE MATEMATICHE FISICHE E NATURALI
CORSO DI LAUREA MAGISTRALE IN SCIENZE E TECNOLOGIE INFORMATICHE

PROGETTO E SVILUPPO DI UN ALGORITMO PER LA PIANIFICAZIONE OTTIMIZZATA DELLA DISTRIBUZIONE CON VIAGGI SINCRONIZZATI

Relazione finale in:
Algoritmi per Applicazioni Aziendali

Relatore:
Chiar.mo Prof. Vittorio Maniezzo

Presentata da:
Elena Rocchi

Sessione I
Anno Accademico 2011-2012

Ringraziamenti

Ringrazio il Professor Vittorio Maniezzo, che in tutti questi mesi di lavoro e di sviluppo mi ha seguita in quello che per me è stato un lavoro di tesi appassionante e stupendo.

Ringrazio tutta la mia famiglia, mio padre Marino, mia madre Liliana, mio fratello Matteo e i miei nonni e ringrazio Marco; è anche grazie a tutti voi che sono arrivata fino a questo traguardo, nuovo punto di partenza. Grazie per tutte le lezioni di vita vissuta che mi avete insegnato; grazie Marco per la tua presenza ed il tuo sostegno in ogni mia scelta, che sono stati, sono e continueranno ad essere importanti.

Ringrazio tutti i miei amici; per il vostro appoggio e per esserci, a condividere il presente.

Ringrazio tutti i miei carissimi compagni di università, noi immatricolati A.A.2005/2006; grazie per la vostra amicizia e per tutti i momenti di allegria condivisi all'università e al di fuori.

Indice

Introduzione

Quadro di riferimento

I problemi riguardanti la distribuzione di merci o servizi tra un insieme di depositi e un insieme di clienti finali sono generalmente noti come *Vehicle Routing Problems* (VRPs). Questi problemi riguardano l'individuazione di un insieme di percorsi di distribuzione, che possa rispondere e soddisfare le esigenze di tutti i clienti coinvolti. Ogni percorso di trasporto viene portato a termine da un singolo veicolo; tutti i percorsi iniziano e terminano il proprio tragitto presso un deposito centrale; ogni percorso deve permettere di soddisfare le richieste dei clienti collocati su di esso. I vincoli che i clienti possiedono possono essere di diversa natura, e i VRPs si caratterizzano secondo i vincoli che questi devono gestire.

Una tipologia di vincoli tra le più studiate è quella che comprende la presenza di finestre temporali (o *time windows*): ogni cliente possiede una propria predefinita finestra temporale, che determina l'orario giornaliero in cui il cliente può ricevere consegne di materiale da parte di un veicolo.

Al contrario, la presenza di vincoli di sincronizzazione sui tempi di consegna merce o visita presso i clienti rappresenta una tipologia di vincoli ancora poco studiata e di applicazione relativamente recente nel campo dei VRPs. Generalmente, solamente un sottoinsieme di clienti possiede questo vincolo, la cui presenza impone che un insieme di due o più veicoli si incontri presso uno stesso cliente nello stesso momento. Questo tipo di problematica è assai comune e può riscontrarsi in molte situazioni reali, specialmente nell'area della gestione del personale su campo: nonostante la maggior parte dei compiti richieda una visita singola, molte organizzazioni nell'area del servizio tecnico e dell'assistenza di malati a domicilio necessitano di pianificare viaggi di visita ai propri clienti o assistiti che prevedano, in caso di necessità di questi ultimi, l'intervento simultaneo di più specialisti. I VRPTWPSs (*Vehicle Routing Problem with Time Windows and Pairwise Synchronization constraints*) sono i problemi di routing in cui ogni cliente possiede

un vincolo di finestra temporale, ed un sottoinsieme di essi possiede un vincoli di sincronizzazione. I VRPs sono noti come problemi NP-hard; ciò significa che lo spazio delle soluzioni di questi problemi cresce in maniera esponenziale all'aumentare della dimensione del problema stesso (in questo caso, all'aumentare del numero di clienti).

Obiettivi

L'obiettivo del lavoro di tesi riguarda la progettazione e lo sviluppo di un algoritmo per la pianificazione ottimizzata della distribuzione con viaggi sincronizzati; il metodo si occupa, dunque, di risolvere in maniera efficiente problemi di routing, caratterizzati dalla presenza di vincoli di sincronizzazione su un sottoinsieme di clienti.

L'algoritmo di risoluzione che si vuole progettare per risolvere questi problemi è un algoritmo mateuristico. I metodi mateuristici sono algoritmi di ottimizzazione, basati sull'interazione tra metodi metaeuristici e tecniche di programmazione matematica; l'impiego di questi metodi nel campo dei problemi di ottimizzazione combinatoria ha permesso di migliorare lo stato dell'arte su diversi problemi, sia di interesse teorico sia emergenti dal mondo reale. I metodi mateuristici permettono di coniugare i pregi che caratterizzano algoritmi esatti e metaeuristiche: grazie all'utilizzo di metodi esatti, è possibile trattare in maniera più specifica alcuni vincoli dei problemi studiati, mentre, grazie alle metaeuristiche, è possibile esplorare ampie zone dello spazio delle soluzioni dei problemi in un tempo accettabile.

Sommario

La trattazione della tesi si articola in cinque capitoli.

Il primo capitolo introduce i VRPs, illustrandone le componenti principali, le loro caratteristiche e i vincoli che sussistono; si discute riguardo ai vincoli di sincronizzazione, enunciando diversi problemi reali in cui questi vincoli possono

essere incontrati; si introduce il VRPTWPS e si fornisce una formulazione matematica del problema.

Il secondo capitolo si occupa di illustrare un insieme di algoritmi e tecniche utilizzati storicamente nell'ambito della risoluzione di problemi NP-hard (metodi euristici e metaeuristiche); si introducono, inoltre, gli algoritmi mateuristici.

Il terzo capitolo affronta nel dettaglio la descrizione dell'algoritmo realizzato per risolvere il VRPTWPS; si illustrano il procedimento metaeuristico implementato, il funzionamento della componente esatta e come questa componente venga integrata all'interno della metaeuristica.

Il quarto capitolo affronta il problema di come configurare in maniera ottimale i parametri utilizzati dall'algoritmo di risoluzione; come illustrato in questo capitolo, la configurazione dei parametri dell'algoritmo riveste un ruolo importante, che può contribuire in maniera forte per ottenere un algoritmo che garantisca buone performace.

Il quinto capitolo illustra i risultati dell'algoritmo applicato per risolvere i problemi Solomon; questo insieme di problemi rappresenta un punto di riferimento per potere valutare le prestazioni dei metodi implementati.

1. Problemi di routing

1.1 Introduzione ai Vehicle Routing Problems

Il trasporto è un dominio importante dell'attività umana. Supporta e rende possibili molte altre attività sociali ed economiche. Ogni volta che si utilizza un telefono, si fanno acquisti in un negozio di alimentari, si legge la propria posta o si viaggia su un aereo per motivi di lavoro o di piacere, si è beneficiari di un sistema che si occupa di spedire messaggi, prodotti o persone da un luogo ad un altro. In particolare, il trasporto delle merci è una delle più importanti attività di oggi [1].

I problemi riguardanti la distribuzione di merci o servizi tra un insieme di depositi e un insieme di clienti finali sono generalmente noti come *Vehicle Routing Problems* (VRPs). Risolvere questi problemi richiede di individuare un insieme di percorsi, ciascuno dei quali sarà eseguito da un singolo veicolo, che abbiano inizio e fine al deposito centrale, tali che tutte le richieste dei clienti lungo il percorso e i vincoli operazionali che sussistono siano soddisfatti e che il costo di trasporto globale sia minimo. Le componenti principali dei VRPs sono diverse:

- rete stradale
- clienti da servire
- veicoli utilizzati per il trasporto merci
- conducenti dei veicoli

La rete stradale utilizzata per il trasporto delle merci può essere generalmente descritta attraverso un grafo, i cui archi rappresentano le strade da percorrere e i cui vertici individuano sia il deposito sia i clienti. Gli archi possono essere orientati o non orientati, a seconda del modo in cui questi possono essere attraversati: nel primo caso in una sola direzione (a causa, ad esempio, della presenza di sensi unici), nel

secondo caso in entrambe le direzioni. Ogni arco della rete stradale ha un costo associato (generalmente rappresentato dalla sua lunghezza) e un tempo di attraversamento, dipendente dal tipo di veicolo utilizzato per il trasporto o dal periodo di attraversamento dell'arco stesso.

Tipici attributi che caratterizzano i clienti nei VRPs sono:

- una coppia di coordinate, che individuano il vertice del grafo corrispondente al cliente stesso
- una quantità richiesta di beni di consumo che devono essere consegnati al cliente
- il periodo temporale giornaliero in cui il cliente può ricevere le consegne di prodotti (detto anche finestra temporale)
- il tempo richiesto per depositare la merce dal cliente
- il sottoinsieme di veicoli che possono servire le richieste del cliente (nel caso in cui, ad esempio, il cliente risieda in una zona a traffico limitato)

Il deposito può essere caratterizzato da un dato numero di veicoli presenti e dall'insieme di tipologie di veicoli utilizzabili per il trasporto; inoltre, ogni deposito possiede limiti di capacità di immagazzinamento della merce.

Il trasporto dei prodotti viene effettuato utilizzando una flotta di veicoli, le cui caratteristiche di composizione e capacità possono essere prefissate o definite a seconda delle richieste dei clienti. Caratteristiche tipiche dei veicoli sono:

- il deposito in cui questo viene posteggiato abitualmente e la possibilità di poter terminare un viaggio di consegna in un deposito diverso da quello in cui il veicolo solitamente viene lasciato in sosta (nel caso in cui il VRP specifico possieda più di un deposito)
- la capacità del veicolo, espressa in termini di peso massimo, volume o numero di pallet trasportabili
- una possibile suddivisione del veicolo in compartimenti, ciascuno dei quali caratterizzato da differenti capacità e tipi di merce caricabili

- la disponibilità di strumenti utilizzabili nelle fasi di carico e scarico merce
- il sottoinsieme di archi della rete stradale che il veicolo può attraversare
- i costi associati all'utilizzo del veicolo stesso (misurabili in termini di distanza coperta, tempo di percorrenza, numero di percorsi effettuati, ecc.)

Tipici vincoli che caratterizzano il lavoro dei conducenti sono rappresentati dalle normative dei contratti sindacali e delle compagnie (ad esempio, periodi lavorativi giornalieri, numero e durata delle pause durante il servizio, ecc.).

Diversi obiettivi possono essere considerati per i VRPs. Alcuni di questi sono:

- la minimizzazione del costo di trasporto globale (dipendente dalla distanza globale percorsa o dal tempo di percorrenza totale o dai costi fissi associati all'utilizzo dei veicoli)
- la minimizzazione del numero di veicoli richiesti per servire tutti i clienti
- il bilanciamento dei percorsi per tempi di percorrenza e carico dei veicoli
- una qualsiasi combinazione pesata di questi obiettivi

I percorsi generati come soluzione ai VRPs devono soddisfare diversi vincoli operazionali, che possono dipendere dalla natura delle merci trasportate, dalle richieste dei clienti da servire e dalle caratteristiche dei veicoli utilizzati. Un esempio tipico di vincolo presente nei VRPs è rappresentato dal vincolo di capacità dei veicoli, per il quale la quantità di merce trasportata in un percorso di servizio ai clienti non può eccedere i limiti di capacità del veicolo; un altro esempio ricorrente è rappresentato dalla presenza di finestre temporali di servizio ai clienti, a causa delle quali un cliente può ricevere le merci richieste solo all'interno di un intervallo temporale limitato. I VRPs possono, inoltre, presentare vincoli di precedenza sull'ordine di consegna delle merci, che eventualmente impongono un ordinamento parziale sulle visite ai clienti all'interno di uno stesso percorso [2].

1.2 Vincoli di sincronizzazione nei Vehicle Routing Problems

Esistono ulteriori tipologie di vincoli nei VRPs: i vincoli di sincronizzazione sulle visite a determinati clienti. La presenza di vincoli di questa natura impone che un insieme di due o più veicoli o tecnici si incontri presso uno stesso cliente nello stesso momento. Questo tipo di problematica è assai comune e può riscontrarsi in molte situazioni reali, specialmente nell'area della gestione del personale su campo: nonostante la maggior parte dei compiti richieda una visita singola (e quindi la pianificazione di viaggi giornalieri di visita ai clienti individuali), molte organizzazioni nell'area del servizio tecnico e dell'assistenza di malati a domicilio necessitano di pianificare viaggi di visita ai propri clienti o assistiti che prevedano, in caso di necessità di questi ultimi, l'intervento simultaneo di più tecnici. Situazioni di questo tipo possono verificarsi durante l'esecuzione di lavori di manutenzione per i quali è indispensabile l'utilizzo, ad esempio, di attrezzature, come scale, che eccedono una certa altezza: infatti, è fondamentale che un tecnico si occupi del montaggio e della messa in sicurezza della scala durante l'esecuzione delle attività di manutenzione [3].

Un altro esempio di compiti che richiedono l'intervento di più persone contemporaneamente è presente nell'area dell'assistenza domiciliare ai malati. L'individuazione di un piano di visita dei malati efficiente è un problema che ha un orizzonte quotidiano, la cui soluzione dipende dalla situazione attuale dei malati e del personale che si occupa delle visite (gestione, ad esempio, dei turni di visita di un infermiere malato, presenza di nuovi pazienti da accudire, ecc.). Ogni membro del personale possiede un insieme di competenze (tipologia di specializzazione) e un tipo di contratto (lavoro a tempo pieno o part time); solitamente, l'itinerario di visita dei pazienti parte da un unico punto centrale di riferimento per tutto lo staff medico, dove vengono elaborati i viaggi di visita ai malati per tutti i membri dello staff. Ogni controllo a domicilio può essere eseguito in certi orari, prevede una certa durata e richiede particolari competenze: ad esempio, se un malato necessita di assistenza per la preparazione del pranzo, la finestra temporale di servizio potrebbe essere

compresa tra le ore 11:00 e le ore 13:00, per una durata complessiva di un'ora. Se la visita richiede altre azioni quotidiane, come lavare un paziente, la finestra temporale di servizio potrebbe essere più ampia, ad esempio delimitata dalle ore 8:00 alle ore 15:00. Per questo tipo di mansione potrebbe rendersi, inoltre, necessaria la presenza di più di una persona: se il malato è in sovrappeso, per poterlo sollevare e permettergli di fare il bagno, possono essere necessari due assistenti [4]. Un altro tipo di situazione in cui potrebbe essere indispensabile la presenza di due collaboratori consiste nel predisporre il paziente ad un trasporto esterno: il personale si assicurerà della salute del malato, occupandosi di vestirlo in maniera adeguata alle condizioni climatiche esterne, oppure preparandolo per il trasporto su sedia a rotelle [5].

Un'altra area in cui è possibile trovare problematiche di sincronizzazione è presente nella gestione di zone forestali, in cui è importante la pianificazione dei percorsi di raccolta del legname. Nelle aree boschive, esistono zone adibite alla raccolta del legname e due tipologie di veicoli si occupano dell'attività di estrazione e raccolta: una tipologia ha il compito di abbattere gli alberi, tagliarli a tronchi e disporli in pila all'interno delle aree destinate alla raccolta, mentre un'altra tipologia si occupa dello spostamento delle pile di tronchi ai margini delle strade che circondano i boschi, dove appositi veicoli, a loro volta, caricheranno e trasporteranno le pile di tronchi agli utenti finali. La pianificazione dei viaggi di raccolta del legname ha un orizzonte annuale e tiene conto della locazione delle aree adibite alla raccolta e delle richieste degli utenti finali. La pianificazione dei percorsi di trasporto dai punti di raccolta sui margini delle aree forestali agli utenti finali viene elaborata con un orizzonte giornaliero o settimanale. Si distinguono due varietà di veicoli: veicoli dotati di gru per il caricamento e scaricamento del legname e veicoli privi di gru. Quest'ultima tipologia di veicoli necessita della presenza di un mezzo in grado di caricare e scaricare legname; questi mezzi sono situati solamente nelle aree di raccolta in cui il livello di legname tagliato supera una certa soglia. Inoltre, non si limitano ad operare in una singola area, ma ne ricoprono diverse, muovendosi da un'area verso un'altra per eseguire carichi di legname sui veicoli. Il vincolo di sincronizzazione presente in questo problema richiede che il mezzo usato per il caricamento e lo scaricamento del legname e il veicolo che deve trasportarlo si trovino nello stesso momento ad uno stesso punto di raccolta [4].

Vincoli di sincronizzazione possono riscontrarsi anche nell'area della gestione di team di lavoro: si consideri il problema dell'assegnamento di diversi gruppi di tecnici ad un insieme di compiti da svolgere. Un gruppo di lavoro può essere composto da diversi membri, ciascuno dei quali possiede diverse competenze. Un team può essere impiegato per svolgere più attività: ad esempio, un gruppo di tecnici può essere richiesto per diversi compiti all'interno di uno stabilimento, mentre un team che si occupa delle vendite di una vasta gamma di prodotti può essere richiesto in svariate regioni del paese. In questo problema si definiscono, quindi, un insieme di attività (progetti, incontri, …), in cui ognuna di esse possiede le seguenti proprietà:

- luogo in cui l'attività si svolge
- finestra temporale di esecuzione (indica il tempo in cui è possibile iniziare quella attività al più presto e quello in cui la si può terminare al più tardi)
- team di lavoro competente in merito allo svolgimento dell'attività
- durata dell'attività

Un compito viene adempiuto se il team di lavoro ad esso preposto lo svolge nel tempo necessario al suo termine e all'interno della finestra temporale prestabilita. Tutti i membri dei team iniziano il loro lavoro partendo da una stessa centrale di riferimento (chiamata "centro servizi"); quando un tecnico giunge nel luogo in cui dovrà svolgere la sua prossima attività, egli dovrà eventualmente attendere l'arrivo sul posto degli altri membri del gruppo di lavoro prescelto per lo svolgimento di quell'attività [6].

I requisiti di sincronizzazione nei VRPs non riguardano solamente la visita simultanea di uno stesso cliente da parte di un sottoinsieme di veicoli, ma possono riguardare anche altri ambiti, estendendo il concetto di sincronizzazione ad aspetti che controllano spazio e carico. Si consideri, ad esempio, il Vehicle Routing Problem with Trailers and Transshipments (VRPTT). Un insieme di produttori fornisce merce, che verrà successivamente raccolta in un deposito centrale. Il problema richiede di elaborare itinerari a costo minimo di raccolta e consegna merce al deposito. Il VRPTT consente, inoltre, di poter eseguire trasferimenti di carico tra

veicoli in luoghi dedicati, come ad esempio nei parcheggi. Un problema di questa natura si può riscontrare in molte situazioni reali: si pensi, ad esempio, al trasporto di latte prodotto da alcuni allevatori verso un deposito centrale. Si supponga che il trasporto possa essere eseguito utilizzando un insieme eterogeneo di veicoli residenti nel deposito, come quelli mostrati in figura 1.1.

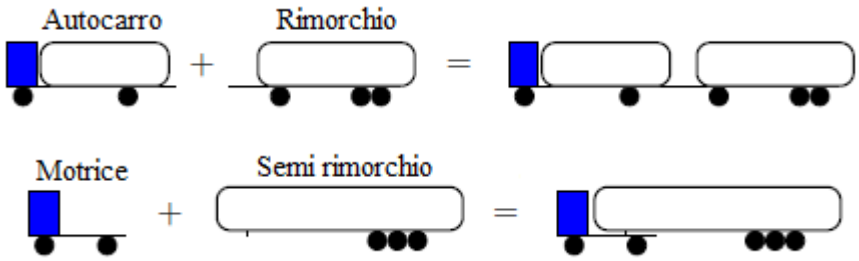

Figura 1.1 Insieme di veicoli disponibili per il trasporto merce

I veicoli considerati differiscono tra loro secondo due criteri indipendenti: il primo di questi è dato dal fatto che gli autocarri e le motrici sono veicoli autonomi, in grado di muoversi liberamente nello spazio, mentre rimorchi e semi rimorchi non sono autonomi, ma devono essere trainati da un opportuno mezzo di trasporto per potersi muovere. Il secondo criterio consiste nel fatto che gli autocarri e le motrici sono veicoli tecnicamente attrezzati per effettuare visite ed interventi ai diversi clienti, mentre i rimorchi e i semi rimorchi non lo sono: possono essere utilizzati solamente come mezzi di supporto, ovvero come depositi mobili nei quali trasferire il carico.

La maggior parte dei produttori non richiede la presenza di un rimorchio, mentre i restanti possono necessitare di autocarri che ne siano dotati; i produttori possono avere delle finestre temporali di consegna, così come le aree dedicate ai trasferimenti di carico possono essere utilizzate all'interno di orari predefiniti. Tutti i veicoli iniziano e terminano il loro percorso al deposito centrale; non è presente un assegnamento prefissato di un rimorchio ad un autocarro o di un semi rimorchio ad una motrice. Ogni veicolo può essere trainato in tutto o parte del suo percorso da un veicolo autonomo compatibile. Inoltre, ogni mezzo può trasferire il suo carico parzialmente o completamente in ogni altro veicolo all'interno di una qualsiasi area

dedicata; in questi luoghi, si può effettuare un trasferimento merce per volta tra una coppia di mezzi.

Il VRPTT presenta diverse proprietà connesse a vincoli di sincronizzazione:

- i produttori possono ricevere visite da due veicoli
- rimorchi e semi rimorchi non sono veicoli autonomi e devono essere trainati da veicoli che lo siano
- rimorchi e semi rimorchi potrebbero essere trainati da diversi veicoli sul loro itinerario
- i trasferimenti di merce coinvolgono veicoli in maniera arbitraria
- nei luoghi adibiti al trasferimento merce, solo un trasbordo per volta può essere eseguito

Le tipologie di vincoli di sincronizzazione presenti sono numerose:

- sincronizzazione sui clienti; questo tipo di sincronizzazione richiede di identificare quale veicolo visita quali clienti, sapendo che le merci devono essere caricate una volta sola da ogni cliente, utilizzando un autocarro e al massimo un rimorchio
- sincronizzazione delle operazioni; questo vincolo richiede la sincronizzazione delle azioni svolte da diversi veicoli presso una stessa locazione (si consideri il caso del trasferimento merce da un mezzo ad un altro, che richiede la presenza nello stesso intervallo temporale di entrambe i veicoli interessati)
- sincronizzazione dei movimenti; il vincolo stabilisce che rimorchi e semi rimorchi debbano essere trainati da veicoli compatibili lungo alcuni archi della rete stradale, qualora il trasporto renda necessario il loro utilizzo. Perciò, è necessario che sia rimorchi e semi rimorchi sia autocarri e motrici si incontrino in un luogo preciso nello stesso momento per poter iniziare assieme il percorso [7]

1.3 Definizione del problema analizzato

Durante il lavoro di tesi è stato studiato il Vehicle Routing Problem with Time Windows and Pairwise Synchronization (VRPTWPS). Questo problema di routing richiede di determinare diversi percorsi di distribuzione merci, che portino un insieme di prodotti a svariati clienti; ciascun viaggio di consegna merci parte da un deposito centrale. Il deposito ospita i veicoli che dovranno essere utilizzati per il trasporto merce ai clienti. Ogni cliente può ricevere la consegna delle merci solamente all'interno di una propria finestra temporale predefinita. Inoltre, un sottoinsieme di questi clienti richiede che il servizio di consegna merce venga svolto da esattamente una coppia di veicoli. La presenza di questo vincolo prevede, quindi, che una coppia qualsiasi di mezzi, indipendentemente dai viaggi di consegna merce che essi svolgono, si incontrino presso uno stesso cliente, consegnino la merce e ripartano per proseguire autonomamente i propri viaggi di consegna.

Il VRPTWPS può essere formulato tramite un grafo $G=(V,A)$, dove V indica l'insieme dei vertici e A l'insieme degli archi. Per facilitare l'esposizione, di seguito sono specificati vertici differenti per individuare i depositi di partenza e di arrivo di ogni percorso, nonostante facciano tutti riferimento ad uno stesso vertice. Si definiscono $D_o=\{n+1 \dots n+k \dots n+m\}$ l'insieme dei depositi di partenza di ogni k-esimo percorso e $D_d=\{n+m+1 \dots n+m+k \dots n+2m\}$ l'insieme dei depositi di destinazione. Un dato numero di clienti n deve ricevere consegne di merci. Si indica con V' l'insieme di questi clienti. L'i-esimo cliente ha associati una richiesta di prodotti d_i, una finestra temporale di servizio $[e_i,l_i]$ e un tempo di durata del servizio s_i. I clienti che richiedono una visita sincronizzata sono modellati come due vertici distinti; quindi, si indica con $S\subset V'$ l'insieme di coppie di vertici da sincronizzare. Ogni veicolo utilizzato per la distribuzione dei prodotti possiede una capacità C. L'attraversamento di un arco(i,j) richiede un costo c_{ij} e un tempo t_{ij}. Si utilizzano variabili binarie x_{ij}^k per indicare se l'arco (i,j) è attraversato dal k-esimo percorso e le variabili continue B_i per indicare il tempo di inizio servizio presso il cliente i. Si riporta di seguito la formulazione matematica del problema.

$$min \sum_{(i,j)\in A} x_{ij}^{k}\, c_{ij} \tag{1}$$

subject to:

$$\sum_{k\in K} \sum_{j|(i,j)\in A} x_{ij}^{k} \leq 1 \qquad \forall i \in V' \tag{2}$$

$$\sum_{j|(j,i)\in A} x_{ji}^{k} - \sum_{j|(i,j)\in A} x_{ij}^{k} = 0 \qquad \forall i \in V', k \in K \tag{3}$$

$$\sum_{j|(n+k,j)\in A} x_{n+k,j}^{k} = 1 \qquad \forall k \in K \tag{4}$$

$$\sum_{j|(j,n+m+k)\in A} x_{j,n+m+k}^{k} = 1 \qquad \forall k \in K \tag{5}$$

$$\sum_{i\in V'} d_i \sum_{j|(i,j)\in A} x_{ij}^{k} \leq C \qquad \forall k \in K \tag{6}$$

$$B_j \geq (B_i + s_i + t_{ij}) x_{ij}^{k} \qquad \forall (i,j) \in A, k \in K \tag{7}$$

$$e_i \leq B_i \leq l_i \qquad \forall i \in V \tag{8}$$

$$B_i = B_j \qquad \forall (i,j) \in S \tag{9}$$

$$x_{ij}^{k} \in \{0,1\} \qquad \forall (i,j) \in A, k \in K \tag{10}$$

La funzione obiettivo (1) richiede di minimizzare i costi totali dei percorsi; i vincoli (2) e (3) impongono che ogni vertice sia visitato al massimo una volta sola e che, se visitato, il veicolo entri ed esca dal vertice. I vincoli (4) e (5) controllano che ogni percorso parta dal deposito di origine e finisca al deposito destinazione, mentre il vincolo (6) garantisce che il carico di ogni veicolo non ecceda i limiti di capacità. Il vincolo (7) stabilisce il tempo di inizio servizio di ogni cliente e il vincolo (8) impone che il suo valore sia compreso tra il tempo di apertura e chiusura della finestra temporale associata a quel cliente. Infine, l'uguaglianza (9) impone che le coppie di vertici su cui insiste un vincolo di sincronizzazione siano visitate da una coppia di veicoli nello stesso momento [3].

Il VRPTWPS è un problema di ottimizzazione combinatoria NP-hard. La risoluzione di questo problema è legata alla soluzione di altri problemi NP-hard: esiste un problema del commesso viaggiatore su ogni percorso di servizio merce creato ed esiste un problema di Bin Packing legato al carico di ogni veicolo. Poiché il VRPTWPS è un problema NP-hard, si ha che il tempo necessario per risolverlo utilizzando algoritmi che ricercano la soluzione esatta del problema cresce in maniera esponenziale con la dimensione del problema stesso. Per questo motivo, per la risoluzione dei problemi NP-hard e quindi anche del VRPTWPS, si utilizzano i cosiddetti *algoritmi euristici* e schemi di ricerca delle soluzioni di tipo *metaeuristico*; tali metodi sono progettati per fornire soluzioni approssimate al problema, che possiedano buone caratteristiche di "vicinanza" alla soluzione esatta del problema e che siano calcolabili in maniera veloce e semplice.

2. Analisi dei metodi di risoluzione

2.1 Algoritmi euristici

Il termine “euristico” deriva dal verbo greco ευρίσκω, il cui significato è trovare. I metodi che sono contraddistinti da questo aggettivo sono procedimenti che, applicati ad un problema complesso, hanno il compito di individuarne una possibile soluzione; le caratteristiche che questi algoritmi possiedono sono le seguenti:

- hanno come scopo quello di individuare una soluzione che possa essere il più possibile “vicina” alla soluzione ottima del problema
- non forniscono garanzie sul grado di vicinanza della soluzione trovata rispetto alla soluzione ottima
- trovano una soluzione in maniera veloce e semplice

Gli algoritmi euristici racchiudono un numeroso insieme di metodi, ciascuno dei quali si basa su un principio ben definito utilizzato per la ricerca di una soluzione. Il più semplice algoritmo di ricerca consiste nella *ricerca esaustiva*: questo schema di ricerca enumera tutte le possibili soluzioni del problema, e restituisce come risultato la soluzione migliore [8]. Questo metodo non può essere applicato nella pratica, poiché, come detto in precedenza, lo spazio delle soluzioni dei problemi di natura combinatoria è molto elevato e non è, quindi, possibile riuscire ad elencare tutte le soluzioni in tempo polinomiale. È, dunque, necessario utilizzare altri metodi di ricerca nello spazio delle soluzioni. Uno schema molto utilizzato è dato dall’applicazione delle *ricerche locali*. Questi pattern di ricerca esaminano una limitata regione dello spazio delle soluzioni (detta *insieme di vicinanza*), individuabile in un intorno di una data soluzione. I metodi di ricerca locale esplorano

l'intorno da essi definito, cercando di individuare la soluzione migliore all'interno di questa area.

Per definire un algoritmo di ricerca locale, è necessario specificare diversi elementi:

- come creare la soluzione iniziale che inizializzi il procedimento di ricerca locale
- quale tipologia di insieme di vicinanza esplorare durante la ricerca locale
- quale criterio di accettazione utilizzare
- quale criterio di terminazione della ricerca scegliere

Un insieme di vicinanza può essere generato a partire da una soluzione iniziale cambiando un sottoinsieme di componenti all'interno della soluzione (per quanto riguarda i problemi di routing, cambiare un sottoinsieme di componenti della soluzione può significare, ad esempio, eseguire uno scambio di archi o di vertici). La ricombinazione di componenti della soluzione viene detta "*mossa*", e, tramite una sequenza di mosse, può avvenire l'esplorazione dell'insieme di vicinanza di una data soluzione. Dopo aver definito quali mosse eseguire per esplorare l'insieme di vicinanza, è necessario stabilire un criterio secondo cui comprendere se le soluzioni individuate all'interno di questo insieme risultano migliori rispetto alla soluzione di partenza; se una soluzione individuata nell'insieme di vicinanza risulta migliore rispetto alla soluzione corrente, tale soluzione viene accettata continuando la ricerca a partire da essa. Si possono distinguere due strategie di accettazione di una soluzione: le strategie *first-accept* (FA) e le strategie *best-accept* (BA). Le strategie FA accettano la prima soluzione migliore che si incontra durante l'analisi dell'insieme di vicinanza, mentre le strategie BA esaminano interamente l'insieme di vicinanza e successivamente selezionano la soluzione migliore al suo interno.

Un'altra famiglia di algoritmi euristici è costituita dai *metodi costruttivi*: questi algoritmi costruiscono una soluzione ammissibile al problema trattato, determinando in maniera iterativa gli elementi da aggiungere, fino ad ottenere una soluzione completa. I metodi costruttivi utilizzano un determinato *criterio di espansione* per

determinare, durante ogni iterazione, quale elemento aggiungere in soluzione. Tale criterio può basarsi su un ordinamento preliminare degli elementi da aggiungere in soluzione, oppure su una regola di ottimalità locale (in quest'ultimo caso, si parla di *algoritmi greedy*).

Di seguito, si analizzano alcuni metodi euristici costruttivi e di ricerca locale utilizzati nell'ambito dei problemi di routing.

2.1.1 Metodi euristici per la costruzione di una soluzione ammissibile

2.1.1.1 Algoritmo dei risparmi Clarke-Wright

L'algoritmo dei risparmi Clarke-Wright deriva il proprio nome dai suoi ideatori, che lo svilupparono nel 1964 [16]. Questo rappresenta un metodo costruttivo, utilizzato generalmente per la risoluzione dei VRPs. Il concetto alla base della costruzione di una soluzione usando questo algoritmo è quello di "risparmio". Questo termine esprime i risparmi sui costi ottenuti unendo due percorsi in uno solo, come mostrato in figura 2.1 (il punto 0 rappresenta il deposito).

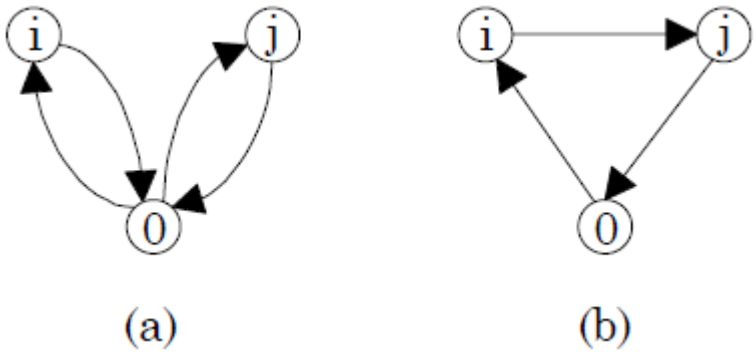

Figura 2.1 Processo di fusione di due percorsi in uno solo

Si indichi con D_a il costo di trasporto totale mantenendo i due percorsi separati e con D_b il costo di trasporto totale ottenuto fondendo i due percorsi; si ha che

$$D_a = c_{0i} + c_{i0} + c_{0j} + c_{j0}$$

e

$$D_b = c_{0i} + c_{ij} + c_{j0}$$

Si ricava che, fondendo i due percorsi in uno solo, si ottiene un risparmio, in termini di costi di trasporto, pari a

$$S_{ij} = D_a - D_b = c_{i0} + c_{0j} - c_{ij}$$

Valori alti di risparmio per una coppia di vertici *i* e *j* indicano che è una buona scelta quella di fondere i percorsi in cui questi vertici si trovano [9].

Esistono due versioni di questo algoritmo: una variante che opera in maniera sequenziale e un'altra che opera in maniera parallela; nella prima versione, i percorsi vengono costruiti uno per volta, mentre nella seconda la costruzione dei percorsi avviene in maniera contemporanea. Si riporta di seguito lo schema che illustra i passi dell'algoritmo nella sua versione parallela.

1. calcola i risparmi $s(i,j)$ tra ogni coppia di vertici (i,j)
2. ordina le coppie (i,j) in maniera decrescente secondo i risparmi $s(i,j)$
3. crea un percorso per ogni vertice
4. considera la prossima coppia (i,j)
5. se *i* e *j* sono punti estremi all'interno dei loro percorsi e questi possono essere fusi tra loro, inserisci l'arco (i,j) in soluzione
6. se ci sono altre coppie da considerare, vai al punto 4
7. completa i percorsi con i collegamenti al deposito

2.1.1.2 Algoritmo I1

Il metodo chiamato I1 è stato sviluppato da Solomon nel 1987 [14]; esso rappresenta un metodo costruttivo applicabile per individuare una soluzione ad un qualsiasi VRP con vincoli di finestre temporali presenti sui vertici. Al contrario del metodo Clarke-Wright visto in precedenza, l'algoritmo I1 costruisce una soluzione in maniera sequenziale, ovvero espandendo un percorso per volta. Ogni itinerario viene inizializzato scegliendo un "seme", che corrisponde al primo vertice da includere nel percorso. Esistono diversi criteri per scegliere il vertice di inizializzazione di un percorso. Uno di questi potrebbe essere, ad esempio, l'individuazione del vertice più lontano dal deposito tra tutti quelli che devono ancora essere inclusi nella soluzione; un altro criterio di scelta potrebbe riguardare le finestre temporali, scegliendo, quindi, il vertice la cui finestra temporale inizia prima delle altre. Dopo avere determinato il primo vertice da includere nel percorso, il metodo I1 continua l'espansione dell'itinerario, inserendo al suo interno nodi ancora da aggiungere in soluzione, con rispetto dei vincoli di capacità dei veicoli e delle finestre temporali di consegna presenti sui vertici. Quando non è più possibile espandere un percorso e nel caso in cui rimangano altri vertici da inserire in soluzione, il metodo I1 inizia la creazione di un nuovo itinerario, scegliendo il vertice seme con cui inizializzare il percorso ed aggiungendo ad esso vertici non ancora presenti in soluzione, fino a che i vincoli presenti sul problema lo consentono. Il metodo I1 termina solamente dopo aver incluso tutti i vertici in soluzione. Nel seguito della trattazione, si analizzano in maggior dettaglio i criteri utilizzati dall'algoritmo I1 per individuare quali vertici inserire nel percorso in espansione.

Dopo avere scelto il vertice seme per inizializzare un nuovo percorso, il metodo utilizza due criteri per selezionare il prossimo vertice u da inserire nel percorso in costruzione. Si indichi con $(i_0, i_1, \ldots, i_m)$ l'itinerario in espansione, dove i_0 e i_m rappresentano il deposito. Per ogni vertice non appartenente alla soluzione, l'algoritmo individua il punto migliore di inserimento tra tutte le coppie di vertici adiacenti i e j presenti sul percorso in espansione; il costo è calcolato come

$$c_1(i(u), u, j(u)) = min_{\rho=1,\ldots,m}\, c_1(i_{\rho-1}, u, i_\rho)$$

dove il costo $c_1(i, u, j)$ è calcolato come

$$c_1(i, u, j) = \alpha_1 c_{11}(i, u, j) + \alpha_2 c_{12}(i, u, j)$$

$$\text{dove } \alpha_1 + \alpha_2 = 1,\ \alpha_1 \geq 0,\ \alpha_2 \geq 0,$$
$$c_{11}(i, u, j) = d_{iu} + d_{uj} - \mu d_{ij},\ \mu \geq 0,$$
$$c_{12}(i, u, j) = b_{ju} - b_j$$

I valori d_{iu}, d_{uj} e d_{ij} rappresentano le distanze tra i vertici i e u, u e j e i e j rispettivamente; b_{ju} indica il nuovo tempo di inizio servizio presso il vertice j, nel caso in cui il vertice u sia inserito nel percorso, e b_j rappresenta il tempo di inizio servizio presso il vertice j prima dell'inserimento del vertice u.

Dopo avere calcolato i costi $c_1(i, u, j)$, tra tutti i vertici non ancora in soluzione, viene individuato il vertice u^* da inserire nel percorso in espansione, calcolato come

$$c_2(i(u^*), u^*, j(u^*)) = max_u \{ c_2(i(u), u, j(u)) \mid u \text{ non appartenga alla soluzione e il percorso rispetti i vincoli del problema}\}$$

dove il costo $c_2(i, u, j)$ è calcolato come

$$c_2(i, u, j) = \lambda d_{0u} - c_1(i, u, j),\ \lambda \geq 0$$

Il parametro λ è utilizzato per controllare quanto il calcolo del punto migliore per l'inserimento del vertice u debba essere influenzato dalla distanza dal deposito e quanto dall'incremento della distanza totale percorsa e del tempo richiesto per visitare il vertice stesso.

2.1.2 Metodi euristici di ricerca locale

2.1.2.1 Ejection chain

Il metodo *ejection chain* (ovvero “catena di espulsione”) fu ideato da Glover [17], e rappresenta un algoritmo euristico molto utilizzato per il miglioramento di soluzioni di problemi di routing. Questo metodo basa il suo funzionamento su una serie di “cambiamenti di stato” degli elementi che compongono la soluzione del problema. I cambiamenti portano tali elementi a subire un’espulsione dal loro stato attuale verso un nuovo stato da determinare. Una ejection chain viene avviata selezionando un insieme di elementi a partire dalla soluzione corrente del problema; tali elementi vengono modificati in modo tale da poter eseguire un cambiamento di stato, che li porti dalla configurazione attuale ad una nuova configurazione. Il risultato del cambiamento di stato eseguito su tali elementi porta ad individuare, eventualmente, altri gruppi di elementi sui quali applicare gli stessi cambiamenti di stato eseguiti in precedenza. Quindi, quando viene iniziata una ejection chain, gli elementi di una soluzione subiscono cambiamenti di stato ed espulsioni dal proprio stato attuale in maniera alternata, innescando spesso un effetto domino. La catena di espulsioni può avere termine oppure non averlo; nel primo caso, la catena termina quando, dopo aver cambiato configurazione ad alcuni elementi della soluzione, non è più necessario espellere altri elementi dal loro stato attuale. In figura 2.2, viene illustrato l’effetto dell’esecuzione del metodo ejection chain su un’istanza di VRP; l’elemento della soluzione è rappresentato dal punto singolo e il cambiamento di stato di tale elemento lo porta ad essere eliminato dal percorso in cui si trova per essere inserito in un altro percorso differente. Nella figura (A) viene scelto il cliente che darà inizio alla catena (cliente rosso); nella figura (B) si nota che questo cliente è stato rimosso dal percorso in cui era inserito; nella figura (C) si osserva che l’inserimento del cliente precedentemente rimosso comporta l’espulsione di un altro cliente dallo stesso percorso di inserimento (cliente verde); infine, nella figura (D), il cliente appena rimosso viene inserito in un percorso differente, senza dover espellere alcun vertice dal percorso e riuscendo, in maniera tale, a chiudere la catena di espulsioni. Se applicato ai vertici di un singolo percorso, il metodo ejection chain

può rivelarsi utile per diminuire il numero di veicoli utilizzati in una data soluzione di un problema di routing: infatti, se le catene di espulsione vengono inizializzate con i vertici di un dato percorso e se tali catene hanno tutte un termine, il percorso risulterà infine vuoto e potrà, quindi, essere eliminato dalla soluzione del problema.

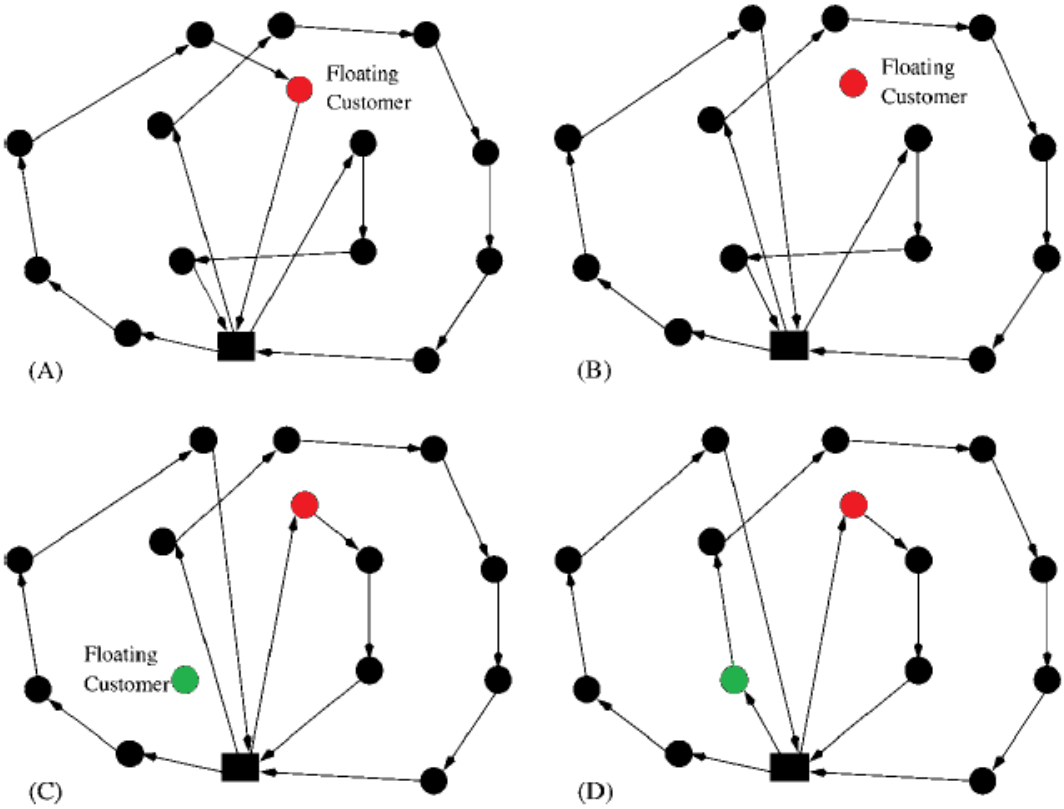

Figura 2.2 Esempio di ejection chain applicata ad un'istanza di VRP

2.1.2.2 2-opt

2-opt è uno tra gli algoritmi di ricerca locale più noti ed utilizzati nell'ambito dei problemi di routing. L'insieme di vicinanza definito dall'operatore prevede di esaminare tutte le soluzioni ottenibili, a partire da quella di partenza, con uno scambio tra una qualsiasi coppia di archi non adiacenti all'interno di un dato percorso. Scambiare una coppia di archi all'interno di un viaggio di consegna merci significa individuare e rimuovere tale coppia, spezzando così il percorso in due percorsi distinti; successivamente, è necessario riconnettere questi percorsi creando due nuovi archi ed invertendo l'ordine di visita dei clienti compresi tra gli archi

scambiati. L'operatore 2-opt ha come scopo quello di diminuire la distanza totale percorsa dell'itinerario su cui viene applicato, riducendo, di conseguenza, il valore della distanza totale globale della soluzione. In figura 2.3, viene riportato un esempio di applicazione del metodo 2-opt su un percorso; il punto di forma quadrata rappresenta il deposito, mentre tutti gli altri punti sono i clienti da servire. Notiamo che l'operatore rimuove dal viaggio di consegna gli archi (i,i+1) e (j,j+1), riconnettendo i due pezzi di percorso creati con l'inserimento degli archi (i,j) e (i+1,j+1). In questo esempio, risulta, inoltre, chiaro come l'applicazione di questo operatore sul viaggio abbia permesso di diminuire la distanza totale percorsa.

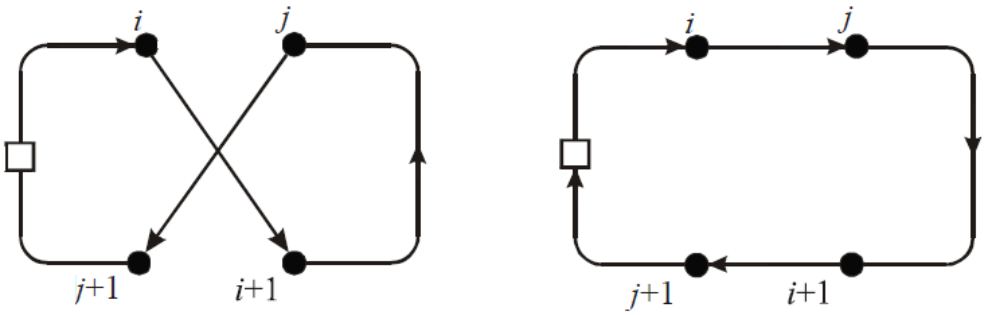

Figura 2.3 Applicazione di 2-opt su un percorso

2.1.2.3 Or-opt

L'idea alla base dell'operatore Or-opt, (introdotto da Or nel 1976 [18]), è quella di trasferire, all'interno di uno stesso percorso, una catena di m vertici consecutivi. Ciò si può realizzare rimuovendo tre archi nella soluzione di partenza e rimpiazzandoli con tre nuovi archi, senza la necessità di dover invertire l'ordine di visita dei vertici trasferiti. In figura 2.4, si visualizza l'applicazione di questo operatore. La catena di vertici trasferiti è di lunghezza 2; i vertici coinvolti nel trasferimento sono i e i+1. Il trasferimento della catena avviene rimuovendo tre archi ((i-1, i), (i+1, i+2) e (j, j+1)) e rimpiazzandoli con gli archi (i-1, i+2), (j, i) e (i+1, j+1). Come l'operatore 2-opt, anche l'obiettivo di Or-opt è quello di eseguire modifiche sulla soluzione che portino ad una diminuzione della distanza totale percorsa.

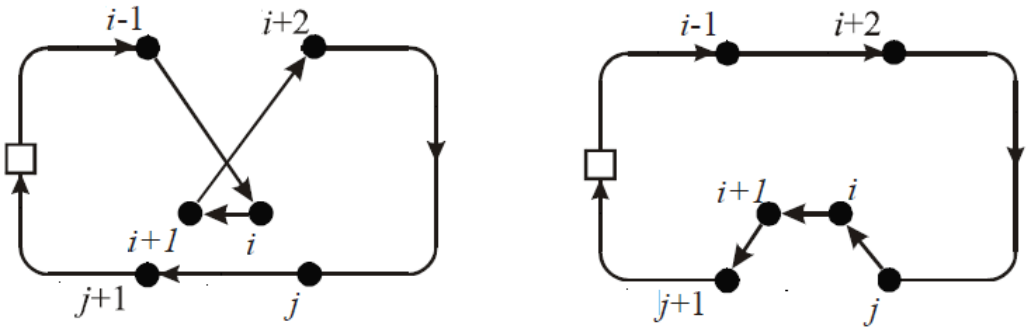

Figura 2.4 Applicazione di Or-opt su un percorso

2.1.2.4 2-opt*

Al contrario di 2-opt e Or-opt, 2-opt* opera su più di un percorso contemporaneamente; questo operatore sceglie una qualsiasi coppia di itinerari appartenenti alla soluzione corrente ed esegue tra essi una serie di scambi di archi, in maniera tale da diminuire la distanza totale percorsa. In figura 2.5, si nota che gli archi (i,i+1) e (j,j+1) vengono rimossi dalla soluzione e rimpiazzati dai nuovi archi (i,j+1) e (j,i+1); come mostra l'esempio, lo scambio di archi tra la coppia di percorsi ha portato ad una diminuzione della distanza totale coperta.

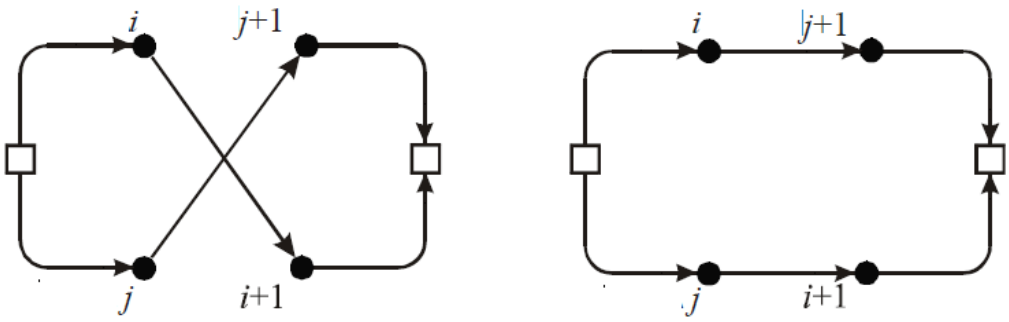

Figura 2.5 Applicazione di 2-opt* su una coppia di percorsi

2.1.2.5 Relocate 1-0

L'operatore relocate 1-0 trasferisce un vertice singolo da un percorso verso un altro, inserendolo tra due vertici consecutivi; lo scopo dell'applicazione di questo metodo è quello di diminuire la distanza totale percorsa. Come mostrato in figura 2.6, il vertice i viene trasferito dal percorso in cui attualmente è collocato in un altro

percorso; ciò viene eseguito rimuovendo gli archi (*i*-1,*i*), (*i*,*i*+1) e (*j*, *j*+1) ed inserendo gli archi (*j*,*i*), (*i*,*j*+1) e (*i*-1, *i*+1).

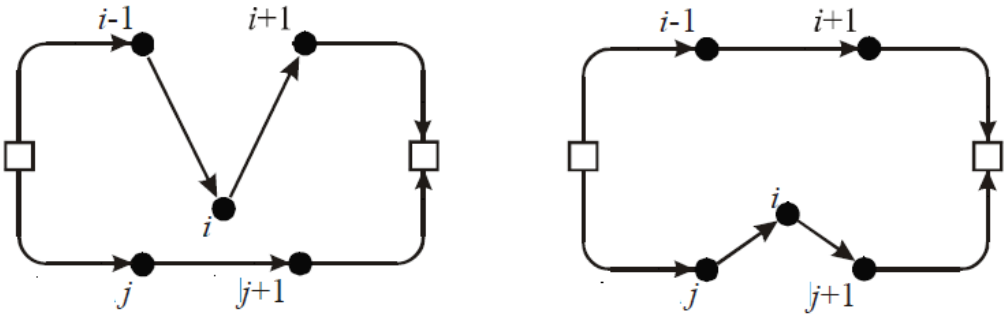

Figura 2.6 Applicazione di relocate 1-0 su un vertice della soluzione

2.1.2.6 Relocate 1-1

Il metodo relocate 1-1 esegue uno scambio tra una coppia di vertici residenti su due percorsi differenti; lo scambio da effettuarsi ha lo scopo di diminuire la distanza totale percorsa. Come mostra la figura 2.7, l'operatore effettua uno scambio tra i vertici *i* e *j*, rimuovendo, quindi, dalla soluzione i rispettivi archi adiacenti ai vertici ed inserendo i nuovi archi di collegamento ai rispettivi percorsi di inserimento.

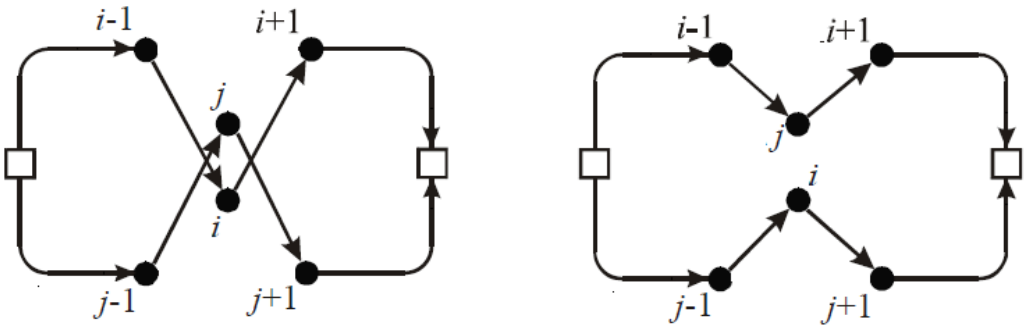

Figura 2.7 Applicazione di relocate 1-1 su una coppia di vertici della soluzione

2.1.2.7 Relocate 2-1

L'operatore relocate 2-1 esegue uno scambio tra una coppia di vertici adiacenti residenti su uno stesso percorso e un vertice appartenente ad un percorso differente. Come mostra la figura 2.8, i vertici adiacenti *i* e *k* vengono scambiati con il vertice *j*.

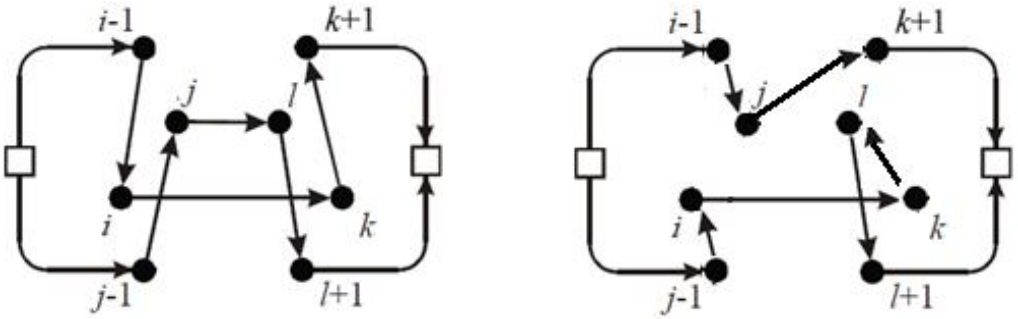

Figura 2.8 Applicazione di relocate 2-1 su una coppia di vertici della soluzione

2.1.2.8 CROSS-exchange

L'operatore CROSS-exchange fu introdotto da Taillard [19]; l'idea alla base del funzionamento di questo operatore è quella di scambiare coppie di archi tra coppie di percorsi, in maniera tale da poter diminuire la distanza totale percorsa. Come mostra la figura 2.9, le coppie di archi (i-1,i), (k,k+1), (j-1,j) e (l,l+1) vengono rimosse dai rispettivi percorsi, introducendo quattro nuovi archi (i-1,j), (l,k+1), (j-1,i) e (k,l+1).

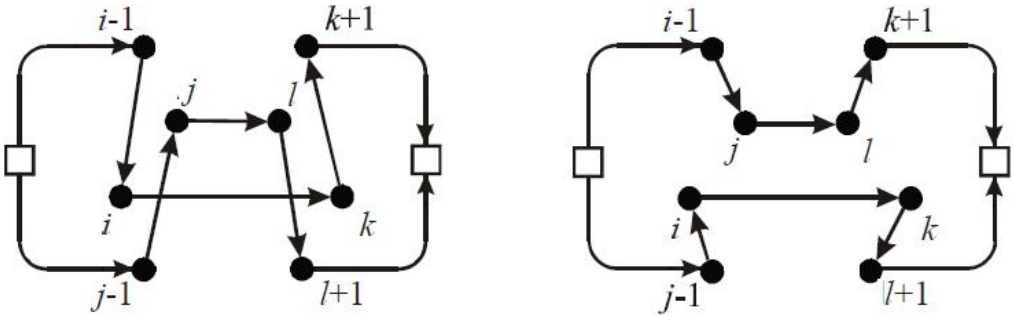

Figura 2.9 Applicazione di CROSS-exchange su una coppia di percorsi della soluzione

2.1.2.9 GENI-exchange

Il metodo GENI-exchange, (introdotto da Gendreau nel 1992 [20]), rappresenta un'estensione dell'operatore relocate 1-0: infatti, questo metodo prevede che un vertice possa essere trasferito in un altro percorso individuando, su di esso, i due vertici a lui più vicini, anche se questi non sono consecutivi. In questo modo, l'operatore GENI-exchange definisce un insieme di vicinanza più ampio rispetto a

quello individuato dall'operatore relocate 1-0. In figura 2.10 si mostra un esempio di applicazione di questo operatore: il nodo *i* viene trasferito dal percorso originale in un nuovo percorso, inserendolo tra i nodi *j* e *k*, (i nodi a lui più vicini), e modificando, di conseguenza, entrambi i percorsi.

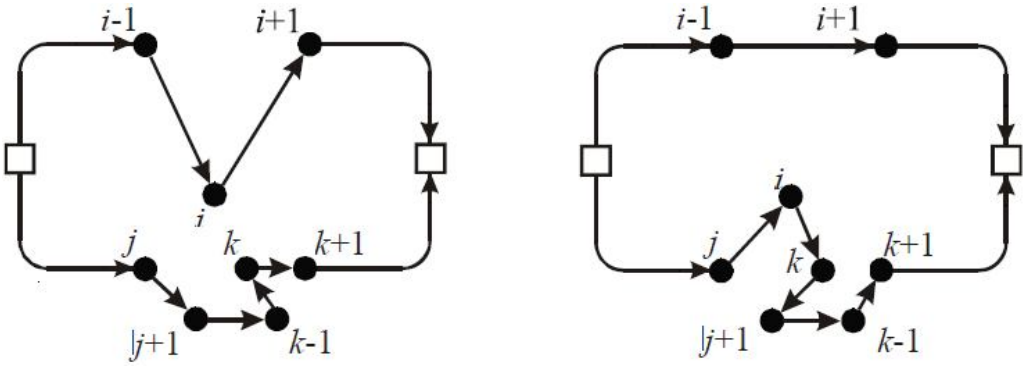

Figura 2.10 Applicazione di GENI-exchange su un vertice della soluzione

Gli algoritmi euristici presentano spesso un problema di "convergenza prematura": a causa dei criteri di ottimalità locale che guidano la loro ricerca, questi metodi individuano, nello spazio delle soluzioni, punti di ottimalità locale, che spesso sono lontani dall'ottimo globale del problema. Solitamente, un metodo di ricerca locale, dopo aver raggiunto un punto di minimo locale, blocca il processo di ricerca, restituendo questo minimo come valore di output. Dunque, i metodi euristici di per sé non sono in grado di esplorare in maniera ampia lo spazio delle soluzioni per individuare una soluzione di qualità migliore, ma si limitano a fornire la soluzione ottima nell'intorno analizzato dai diversi metodi. La figura 2.11 mostra in maniera esemplificata come i punti di ottimalità locale possano essere anche molto distanti dal punto di ottimalità globale nello spazio delle soluzioni; il grafico delinea il valore delle soluzioni di un generico problema di minimizzazione.

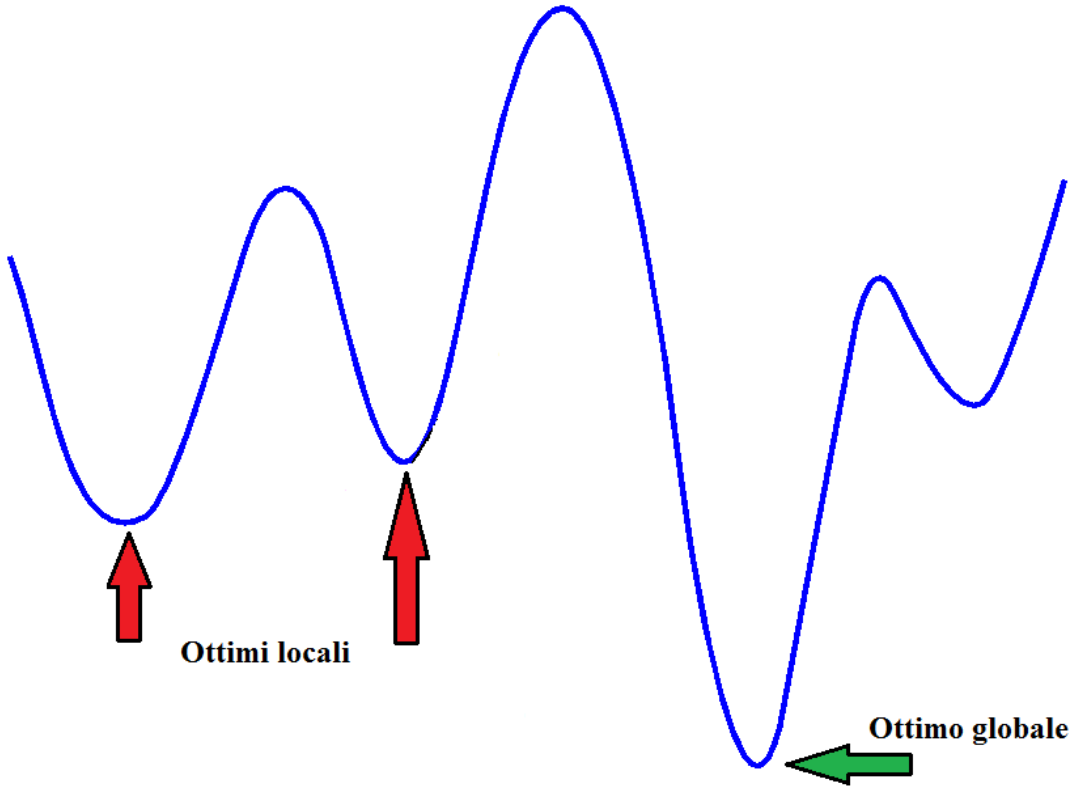

Figura 2.11 Ottimi locali e globale in uno spazio delle soluzioni

2.2 Metaeuristiche

Le metaeuristiche sono algoritmi progettati per la risoluzione di problemi di ottimizzazione complessi; questi metodi sono stati riconosciuti come uno degli approcci più pratici per risolvere molti problemi complessi del mondo reale che possiedono una natura combinatoria. L'applicabilità delle metaeuristiche come metodo preferito rispetto ad altri metodi di ottimizzazione è data dal fatto che questi algoritmi riescono a guidare la ricerca di diversi punti di ottimo locale all'interno dello spazio delle soluzioni, individuando, quindi, soluzioni di buona qualità per problemi di ottimizzazione il cui spazio delle soluzioni è caratterizzato dalla presenza di molti punti di ottimo locale. L'approccio metaeuristico per risolvere problemi di ottimizzazione consiste nel costruire una o più *soluzioni iniziali* e, quindi, nell'utilizzare le soluzioni individuate come punto di partenza per la ricerca guidata nello spazio delle soluzioni. La struttura della ricerca ha molti elementi comuni tra le differenti tipologie di metaeuristiche: ad ogni iterazione dell'algoritmo di ricerca, è

presente una soluzione o un insieme di soluzioni θ_k che rappresenta lo stato corrente della ricerca. Molte metaeuristiche come tabu search e simulated annealing sono metodi solution-to-solution, ovvero θ_k corrisponde ad una sola soluzione; altre metaeuristiche, come ad esempio gli algoritmi genetici, sono set-based, ovvero rappresentano θ_k come un insieme di soluzioni. Di seguito, si descrive la struttura di ricerca guidata comune a tutte le metaeuristiche.

Si definisce un insieme di vicinanza $N(\theta_k)$ della/e soluzione/i θ_k; si individuano e valutano una o più soluzioni candidate $\{\theta^c\} \subset N(\theta_k)$, stimandone le performance rispetto alle soluzioni correnti θ_k. In base ai risultati ottenuti, le soluzioni candidate possono essere accettate (in tal caso, si porrà $\theta_{k+1} = \theta^c$) o rifiutate (in tal caso, si porrà $\theta_{k+1} = \theta_k$) [10]. Si ottiene il seguente schema metaeuristico generale:

1. costruisci una soluzione iniziale θ_0; poni $k = 0$
2. ripeti
   - 2.1 identifica un insieme di vicinanza $N(\theta_k)$ della soluzione corrente
   - 2.2 seleziona una soluzione candidata $\{\theta^c\} \subset N(\theta_k)$ dall'insieme di vicinanza
   - 2.3 accetta la soluzione candidata ponendo $\theta_{k+1} = \theta^c$, oppure rifiutala ponendo $\theta_{k+1} = \theta_k$
   - 2.4 poni $k = k+1$
3. fino a che non si soddisfa una condizione di termine

Di seguito, si illustrano quattro note metaeuristiche utilizzate per la risoluzione di problemi complessi: simulated annealing, tabu search, large neighborhood search e adaptive large neighborhood search.

2.2.1 Simulated annealing

Simulated annealing (Kirkpatrick 1983 [21]) è uno schema metaeuristico del tipo solution-to-solution; la definizione dell'insieme di vicinanza viene stabilita dall'utente in base al problema trattato. Simulated annealing trae ispirazione dal

processo fisico di solidificazione dei metalli fusi: per ottenere la solidificazione di un metallo fuso, la temperatura viene lentamente abbassata, facendo transitare il sistema da uno stato energetico ad un altro con un grado inferiore di energia, fino a che il sistema non viene a trovarsi in uno stato di energia minima. La peculiarità di simulated annealing risiede nella capacità di scelta della soluzione corrente. Se la soluzione candidata è migliore, simulated annealing accetta tale soluzione aggiornando il valore della soluzione corrente; al contrario, se la soluzione candidata risulta peggiore rispetto a quella corrente, simulated annealing può accettare la soluzione candidata con una probabilità calcolata come

$$P[Accept\ \theta^c] = e^{-\frac{f(\theta_k)-f(\theta^c)}{T_k}},$$

dove $f:\Theta\rightarrow R$ rappresenta la funzione obiettivo del problema e T_k è un parametro chiamato *temperatura*. La temperatura viene ridotta iterativamente nel corso dell'algoritmo. Si ha quindi che, partendo con valori di temperatura alti, inizialmente soluzioni peggiori vengono accettate con una sufficiente alta probabilità. Come illustra la formula, la probabilità di accettare una soluzione è alta se la differenza in termini di valori di funzioni obiettivo è bassa e il valore della temperatura T_k è alto. Il meccanismo di abbassamento della temperatura, usato per la solidificazione dei metalli, viene impiegato nel simulated annealing per poter "uscire" dai punti di ottimalità locale dello spazio delle soluzioni ed ampliare gli orizzonti della ricerca verso altri punti di ottimalità locale non ancora individuati.

2.2.2 Tabu search

Come simulated annealing, anche tabu search (Glover 1986 [22]) è un algoritmo metaeuristico solution-to-solution, in cui le caratteristiche dell'insieme di vicinanza sono stabilite dall'utente. Anche tabu search, come simulated annealing, prevede meccanismi di accettazione di "soluzioni peggioranti" per poter uscire da punti di ottimalità locale. La peculiarità che contraddistingue tabu search dagli altri schemi

metaeuristici è data da come avviene la selezione, nell'insieme di vicinanza, delle soluzioni candidate. Il procedimento proibisce che vengano eseguite le ultime "mosse" nel cammino di ricerca, in modo che il metodo non possa tornare sui suoi passi e ricadere in un minimo locale. Per realizzare ciò, l'algoritmo aggiorna iterativamente una lista L_k di soluzioni recentemente visitate, che viene detta "lista tabu", memorizzando le soluzioni che non possono essere riesaminate. Caratteristica basilare del tabu search è, quindi, l'uso sistematico della memoria: per aumentare l'efficacia del processo di ricerca, viene tenuta traccia non solo delle informazioni locali (come il valore corrente della funzione obiettivo), ma anche di alcune informazioni relative all'itinerario percorso. Tali informazioni vengono impiegate per guidare la mossa dalla soluzione corrente alla soluzione successiva, da scegliersi all'interno dell'insieme di adiacenza.

2.2.3 Large neighborhood search

Large neighborhood search (*LNS*, Shaw 1998 [23]) è un schema metaeuristico, la cui idea base è quella di esplorare insiemi di vicinanza complessi attraverso l'utilizzo di differenti euristiche. Questo algoritmo fa uso di un insieme di operatori cosiddetti "*destroy/repair*": quando vengono applicati ad una soluzione, i metodi euristici di destroy rilassano alcuni vincoli di tale soluzione, distruggendo una parte di essa. Applicando successivamente un operatore di repair, è possibile ripristinare i vincoli precedentemente rilassati, ricostruendo, in maniera tale, una soluzione ammissibile al problema. L'applicazione alla soluzione corrente di una coppia di operatori "destroy/repair" avviene iterativamente, fino a che non si verificano le condizioni di termine dell'algoritmo metaeuristico.

2.2.4 Adaptive large neighborhood search

Adaptive large neighborhood search (*ALNS*, Ropke, Pisinger 2006 [11]) è un'estensione di LNS; costituisce un framework di ricerca locale in cui un insieme di diversi algoritmi competono per modificare la soluzione corrente. Durante ogni

iterazione, viene scelto un algoritmo per distruggere la soluzione corrente e viene scelto un algoritmo per ricostruire una nuova soluzione; tale soluzione viene accettata se essa soddisfa i criteri di accettazione definiti dalla metaeuristica. Di seguito, si mostra lo schema di ALNS.

1. costruisci una soluzione ammissibile x; poni $x^* = x$
2. ripeti
    3. scegli un operatore di destroy e un operatore di repair utilizzando un meccanismo di selezione a roulette, basato sui punteggi $\{\pi_j\}$ precedentemente ottenuti dai singoli operatori
    4. genera una nuova soluzione x' ottenuta da x utilizzando gli operatori destroy e repair scelti
    5. se x' può essere accettata, poni $x = x'$
    6. aggiorna i punteggi π_j degli operatori usati
    7. se $f(x) < f(x^*)$, poni $x^* = x$
8. fino a che non si verificano le condizioni di termine
9. restituisci x^*

In riga 1, avviene la costruzione di una soluzione iniziale, mentre le righe comprese tra 2 e 8 delimitano il ciclo del procedimento metaeuristico. Come illustrato in linea 3, durante ogni iterazione vengono scelti un operatore di destroy e un operatore di repair. Una procedura adattiva controlla con un meccanismo casuale quale operatore scegliere in base ai risultati ottenuti sulla soluzione del problema in precedenti applicazioni dell'operatore stesso. Più l'operatore ha contribuito al processo di soluzione del problema, maggiore sarà il suo punteggio π_j, e, quindi, maggiore sarà la probabilità che tale operatore sia scelto. La procedura adattiva utilizza un meccanismo di determinazione degli operatori da utilizzare basato su un'estrazione a roulette. Se il punteggio dell'i-esimo operatore è π_i e si hanno a disposizione m operatori totali, allora la probabilità di scegliere il j-esimo operatore è pari a

$$\frac{\pi_j}{\sum_{i=1}^{m} \pi_i}$$

In linea 4 si applicano in sequenza gli operatori destroy e repair scelti al passo precedente. Per ottenere un'adeguata diversificazione, può risultare vantaggioso, durante l'applicazione di tali operatori, l'utilizzo di fattori di rumore o di casualità.

In linea 6 avviene l'aggiornamento dei punteggi degli operatori. Diversi criteri possono essere usati per misurare quanto un operatore abbia contribuito al processo di soluzione: quando questo permette di individuare nuove soluzioni migliori, ovviamente, i punteggi ricevuti sono alti, ma anche quando individua soluzioni mai scoperte vengono assegnati punteggi. Punti agli operatori possono anche essere assegnati se questi hanno individuato soluzioni peggioranti che, in base ai criteri di accettazione utilizzati, sono state accettate (queste situazioni possono verificarsi nel caso in cui il framework metaeuristico utilizzi un criterio di accettazione come definito da simulated annealing). Esistono ulteriori politiche di gestione dei punteggi: alcune implementazioni di ALNS attribuiscono punteggi separati per operatori di destroy e repair, mentre altre implementazioni stabiliscono un punteggio unico per ogni coppia di operatori utilizzata.

Ogni *M* iterazioni della metaeuristica i punteggi degli operatori vengono azzerati e le probabilità di scelta degli operatori ricalcolate.

Quando le condizioni di termine della metaeuristica vengono verificate, essa restituisce la migliore soluzione individuata durante il procedimento.

2.3 Mateuristiche: ibridazione di metaeuristiche e tecniche di programmazione matematica

Le mateuristiche sono algoritmi di ottimizzazione basati sull'interazione tra metodi metaeuristici e tecniche di programmazione matematica; l'impiego di questi metodi nel campo dei problemi di ottimizzazione combinatoria ha permesso di

migliorare lo stato dell'arte su diversi problemi sia di interesse teorico sia emergenti dal mondo reale. I contributi forniti dalla ricerca in questo campo possono essere classificati sotto due aspetti: come i metodi metaeuristici possono migliorare i metodi esatti e come i metodi esatti possono migliorare gli algoritmi metaeuristici. In particolare, quest'ultima area è la più intensamente studiata, dove i potenziali benefici rispetto allo stato dell'arte sembrano essere più immediati. In quest'area, si possono identificare due direzioni in cui la ricerca si sta orientando: la prima consiste nell'utilizzo di tecniche di programmazione matematica come componenti da includere in metodi metaeuristici, la seconda nel creare algoritmi metaeuristici derivati dalle logiche di funzionamento interno dei metodi di programmazione matematica [12].

3. Definizione delle metodologie utilizzate per la risoluzione del problema

3.1 Problema analizzato e motivazioni di utilizzo del metodo di risoluzione

Il problema analizzato durante il lavoro di tesi è il VRPTWPS, ovvero Vehicle Routing Problem with Time Windows and Pairwise Synchronization. Questo problema di routing prevede la presenza di un certo numero di clienti, i quali richiedono un servizio di consegna merci; è presente un unico deposito centrale, in cui vengono conservate tutte le merci da spedire e i veicoli utilizzati per il loro trasporto. La soluzione di questo problema richiede di determinare un insieme di percorsi di distribuzione merci, che partano dal deposito centrale servendo tutti i clienti, ciascuno esattamente una volta. Ogni cliente può ricevere la consegna delle merci solamente all'interno di una propria finestra temporale predefinita; un sottoinsieme di questi clienti richiede, inoltre, che il servizio di consegna merce venga svolto da esattamente una coppia di veicoli. La presenza di quest'ultimo vincolo prevede, quindi, che una qualsiasi coppia di mezzi, indipendentemente dai viaggi di consegna merce che essi svolgono, si incontrino presso uno stesso cliente, consegnino la merce e ripartano per proseguire autonomamente i propri viaggi di consegna.

Come esposto nel Capitolo 1, esistono diversi obiettivi che è possibile perseguire durante la risoluzione dei problemi di routing, e quindi anche del VRPTWPS. Lo sviluppo del metodo di risoluzione di questo problema ha avuto come obiettivo primario quello di ottenere soluzioni che minimizzassero il numero di veicoli

utilizzati per eseguire i viaggi di distribuzione merce ai clienti; il secondo obiettivo stabilito è stato quello di rendere minima la distanza totale percorsa dai veicoli per servire tutti i clienti.

I vincoli di sincronizzazione che caratterizzano il VRPTWPS rendono questo problema difficile da risolvere utilizzando tradizionali metodi di ricerca locale. L'inserimento di un cliente in un percorso di distribuzione merci può diventare complicato e spesso non possibile, a causa del rispetto di eventuali vincoli di sincronizzazione dei clienti presenti sul percorso. La struttura dei percorsi di ogni soluzione è fortemente condizionata da questi vincoli, i quali impongono forti legami di dipendenza tra i percorsi che condividono vertici sincronizzati. L'alto numero di interconnessioni rende, dunque, il problema assai difficile da risolvere. Solitamente, si utilizzano algoritmi euristici per la risoluzione di questi problemi.

Come enunciato al termine del Capitolo 1, il VRPTWPS appartiene alla classe di problemi NP-hard. Ogni problema che fa parte di questo insieme necessita di un tempo di risoluzione esponenziale rispetto alla dimensione del problema stesso. Per ottenere una soluzione di buona qualità in maniera efficiente si utilizzano, quindi, algoritmi euristici e schemi di ricerca delle soluzioni di tipo metaeuristico; tali metodi sono progettati per fornire soluzioni approssimate del problema, che possano avere buone caratteristiche di "vicinanza" alla soluzione esatta del problema e che possano essere ricavabili in maniera relativamente veloce e semplice. Nel seguito del capitolo, verranno approfondite nel dettaglio le fasi di funzionamento dell'algoritmo progettato per la risoluzione del problema.

3.2 Algoritmo progettato per la risoluzione del problema

La risoluzione del VRPTWPS è stata affrontata progettando un algoritmo mateuristico: tale metodo integra, all'interno di uno schema metaeuristico di base, una componente esatta, che ha il compito di migliorare la qualità della soluzione ottenuta.

La soluzione iniziale, che costituisce il punto di partenza per la ricerca nello spazio delle soluzioni, è unica.

Lo schema metaeuristico adottato è Adaptive Large Neighborhood Search (ALNS, introdotto nel Capitolo 2). Questo procedimento prevede la presenza di un insieme di operatori cosiddetti destroy/repair; gli operatori destroy distruggono una parte della soluzione correntemente considerata, mentre quelli di repair, a partire dalla soluzione parziale ottenuta, cercano di ricostruire una nuova soluzione ammissibile. Ogni coppia di operatori destroy/repair esplora un diverso insieme di vicinanza di una data soluzione corrente. Diverse coppie di operatori destroy/repair vengono utilizzate ed ognuna di esse compete con tutte le altre coppie per la ricerca di soluzioni sempre migliori all'interno dello spazio delle soluzioni. Ogni coppia di operatori ha un punteggio, che ne fornisce il grado di efficacia nella ricerca di soluzioni non ancora individuate e di buona qualità rispetto alla soluzione migliore individuata dalla metaeuristica. Se l'applicazione di questi operatori consente l'individuazione di una nuova soluzione migliore, il valore della soluzione corrente viene aggiornato al valore della soluzione migliore appena scoperta; se ciò non avviene, la metaeuristica prevede comunque un meccanismo probabilistico di accettazione di una soluzione non migliorante, per poter offrire un buon grado di esplorazione dell'insieme delle soluzioni. Il criterio di accettazione definito all'interno di ALNS si basa sullo schema definito da Simulated Annealing: in questo modo, anche soluzioni peggioranti possono diventare nuovo punto di partenza per la ricerca, favorendo la diversificazione dell'analisi dell'insieme delle soluzioni del problema.

Per poter trattare in maniera specifica i vincoli di sincronizzazione presenti sul problema, è stata integrata, nel framework di ALNS, una componente esatta; lo scopo di questa componente è quello di migliorare la qualità di una data soluzione, riposizionando, all'interno dei percorsi in cui sono stati inseriti, i vertici su cui insistono vincoli di sincronizzazione. Questa componente viene applicata ogni volta che ALNS individua una soluzione migliorante.

Come illustrato nel Capitolo 1, per poter rappresentare l'esigenza di visite sincronizzate su un sottoinsieme di clienti del problema, è necessario duplicare i vertici che corrispondono a questi clienti. La gestione del vincolo di sincronizzazione

su questi vertici avviene riducendo le loro finestre temporali ad un unico istante nel tempo. Se, per esempio, la finestra temporale di un vertice da sincronizzare è [8:00, 12:00], essa può essere ridotta stabilendo l'istante di sincronizzazione in un qualsiasi instante nel tempo compreso tra le ore 8:00 e le ore 12:00, ad esempio, alle ore 10:30. Questa riduzione viene operata durante la fase di costruzione della soluzione iniziale della metaeuristica, secondo il seguente procedimento: quando il primo della coppia di vertici da sincronizzare viene inserito in soluzione, il valore delle finestre temporali di entrambe i vertici rimane invariato. Quando si considera l'inserimento in soluzione del secondo vertice della coppia di vertici da sincronizzare, si valutano tutte le possibili alternative per impostare l'istante di sincronizzazione. Le scelte ammissibili sono le seguenti:

- posticipare il tempo di visita del cliente inserito per primo per farlo coincidere col tempo di visita del cliente inserito per secondo (nel caso in cui il tempo di visita del cliente inserito come secondo sia superiore a quello del cliente inserito come primo)
- anticipare il tempo di visita del cliente inserito per primo per farlo coincidere col tempo di visita del cliente inserito per secondo (nel caso in cui il tempo di visita del cliente inserito come secondo sia inferiore a quello del cliente inserito come primo)
- posticipare il tempo di visita del cliente inserito per secondo per farlo coincidere col tempo di visita del cliente inserito per primo (nel caso in cui il tempo di visita del cliente inserito come primo sia superiore a quello del cliente inserito come secondo)
- anticipare il tempo di visita del cliente inserito per secondo per farlo coincidere col tempo di visita del cliente inserito per primo (nel caso in cui il tempo di visita del cliente inserito come primo sia inferiore a quello del cliente inserito come secondo)
- mantenere i tempi di visita inalterati (nel caso in cui i tempi di visita di entrambe i clienti siano uguali)

Il posticipo o l'anticipo degli orari di visita di uno dei clienti da sincronizzare deve permettere di mantenere l'ammissibilità della soluzione e, dunque, di rispettare i vincoli di finestre temporali degli altri clienti serviti sullo stesso percorso. Se è possibile applicare una delle alternative per l'impostazione dell'istante di sincronizzazione, le finestre temporali di entrambe i clienti vengono ridotte secondo l'istante di sincronizzazione individuato. Posticipare l'applicazione del vincolo di sincronizzazione al momento dell'inserimento in soluzione del secondo vertice da sincronizzare permette di rendere più flessibile l'impostazione dell'istante di sincronizzazione, e, di conseguenza, di aumentare la probabilità di individuare soluzioni di miglior qualità.

Durante l'esecuzione di ALNS, è possibile ristabilire l'istante di sincronizzazione: ciò può essere fatto solamente se si verificano particolari condizioni. La riduzione delle finestre temporali ad istanti nel tempo garantisce che una coppia di veicoli giunga presso uno stesso vertice nello stesso momento. Nei seguenti paragrafi, saranno approfonditi il metodo di creazione della soluzione iniziale, il funzionamento della metaeuristica e della componente esatta, indicando, in ciascuna di queste fasi dell'algoritmo, come avviene la gestione del vincolo di sincronizzazione.

3.2.1 Ottimizzazione di una soluzione

In questo paragrafo, si illustra una procedura di ottimizzazione di una data soluzione, utilizzata in diverse fasi dell'algoritmo di risoluzione del VRPTWPS, e, pertanto, illustrata separatamente. Il metodo integra gli algoritmi di ricerca locale discussi nel Capitolo 2. Lo schema della procedura è riportato di seguito.

```
ottimizzazione_soluzione(s)
1.  soluzione_migliore = true;
2.   while(soluzione_migliore)
     3.   soluzione_migliore = false;
     4.   copia = copia_soluzione(s);
     5.   ejection_chain(copia);
     6.   su ogni percorso "p" della soluzione "copia"
```

```
            7.    esegui_2_opt(p);
        8.  su ogni percorso “p” della soluzione “copia”
            9.    esegui_or_opt(p);
        10. relocate_1_0(copia);
        11. relocate_1_1(copia);
        12. relocate_2_1(copia);
        13. 2_opt*(copia);
        14. cross_exchange(copia);
        15. geni(copia);
        16. aggiorna_soluzione(copia, s, soluzione_migliore);
    17. end
```

Il metodo di ottimizzazione implementato è iterativo e viene eseguito fino a che è possibile ottenere soluzioni migliori rispetto alla soluzione di partenza. Tale algoritmo applica in maniera sequenziale diversi metodi di ricerca locale. Il suo obiettivo primario è quello di diminuire il numero di percorsi della soluzione su cui è applicato, mentre l’obiettivo secondario è quello di diminuire la distanza totale percorsa. La sequenza in cui i metodi di ricerca locale sono eseguiti rispetta un ordinamento crescente di complessità e velocità di esplorazione degli insiemi di vicinanza.

In riga 1, si imposta a true il valore di test per l’esecuzione del ciclo while, in maniera tale da eseguire almeno una volta i metodi di ricerca locale sulla soluzione corrente.

In riga 2, si verifica la condizione del ciclo while, mentre in riga 3 si imposta il valore di test a false per verificare, al termine del ciclo, se sia stata individuata o meno una soluzione migliorante.

In riga 4, l’algoritmo esegue una copia temporanea della soluzione corrente; su questa soluzione, successivamente, saranno eseguiti gli algoritmi di ricerca locale.

In riga 5, viene richiamata una procedura che utilizza l’operatore ejection chain, con lo scopo di diminuire il numero di percorsi della soluzione. Il metodo analizza e prova ad eliminare ogni percorso della soluzione, inizializzando ogni ejection chain a partire dai vertici del percorso che sta analizzando. L’ordine con cui vengono

esaminati i percorsi da eliminare è casuale; ciò permette di poter controllare un ampio sottoinsieme di soluzioni. La procedura di analisi ed eliminazione viene ripetuta su ogni percorso, fino a che è possibile eliminare almeno un percorso della soluzione.

Nelle righe 6-7, l'operatore 2-opt viene applicato su ogni percorso della soluzione. L'implementazione realizzata prevede di utilizzare questo operatore applicando lo scambio tra tutte le coppie di archi non adiacenti del percorso analizzato; l'applicazione dell'operatore di scambio tra coppie di archi viene ripetuta fino a che almeno uno scambio porta ad una diminuzione della distanza totale attraversata.

Nelle righe 8-9, l'operatore Or-opt viene applicato su ogni percorso della soluzione. L'implementazione creata applica questo operatore provando il trasferimento di tutte le catene di due vertici consecutivi all'interno di uno stesso percorso; lo spostamento delle catene di vertici consecutivi viene iterato fino a che è possibile ottenere una diminuzione della distanza totale percorsa.

In riga 10, viene richiamato il metodo che implementa l'operatore relocate 1-0. Questo metodo applica l'operatore su ogni vertice del problema, e tale procedura viene ripetuta fino a che è possibile ottenere una diminuzione della distanza totale percorsa.

In riga 11, viene richiamato il metodo che implementa l'operatore relocate 1-1. Questo metodo applica l'operatore tra ogni coppia di vertici del problema, ripetendo tale procedura fino a che è possibile ottenere una diminuzione della distanza totale percorsa.

In riga 12, viene richiamato il metodo che implementa l'operatore relocate 2-1. Questo metodo applica l'operatore su ogni coppia di vertici consecutivi della soluzione, fino a che è possibile ottenere una diminuzione della distanza totale percorsa.

In riga 13, viene invocato il metodo che realizza l'operatore 2-opt*. L'implementazione realizzata prevede di applicare questo operatore provando lo scambio tra ogni arco appartenente ad un percorso e tutti gli archi appartenenti agli altri percorsi. La procedura viene ripetuta fino a che almeno uno scambio di archi porta ad una diminuzione della distanza totale percorsa.

In riga 14, viene invocato il metodo che realizza l'operatore CROSS-exchange. L'implementazione realizzata prevede di applicare questo operatore provando lo scambio tra due qualsiasi coppie di archi su un percorso e tutte le coppie di archi presenti sugli altri percorsi. La procedura viene ripetuta fino a che almeno uno scambio di archi porta ad una diminuzione della distanza totale percorsa.

In riga 15, viene richiamato il metodo che realizza l'operatore GENI. Questo metodo applica l'operatore su ogni vertice del problema, e tale procedura viene ripetuta fino a che è possibile ottenere una diminuzione della distanza totale percorsa.

In riga 16, avviene l'eventuale aggiornamento della soluzione corrente al valore della soluzione temporanea individuata dopo l'applicazione dei metodi di ricerca locale. Il valore di test del ciclo while viene aggiornato in base al fatto che sia stata individuata o no una soluzione migliorante; nel primo caso, il suo valore è posto a true e il ciclo while può essere ulteriormente eseguito, in caso contrario il suo valore è posto a false e la procedura di ottimizzazione della soluzione può terminare.

3.2.2 Generazione della soluzione iniziale

Di seguito, si riporta lo schema della procedura che calcola la soluzione iniziale utilizzata come punto di partenza per la ricerca nello spazio delle soluzioni eseguita da ALNS.

```
calcolo_soluzione_iniziale()
1.  applica_I1(s1);
2.  copia_s1 = copia_soluzione(s1);
3.  ottimizzazione_soluzione(s1);

4.  copia_s1_ej_1 = copia_soluzione(copia_s1);
5.  applica_ejection_chain_tipo_1(copia_s1_ej_1);
6.  ottimizzazione_soluzione(copia_s1_ej_1);
7.  aggiorna_soluzione(copia_s1_ej_1, s1);

8.  copia_s1_ej_2 = copia_soluzione(copia_s1);
```

```
9.  applica_ejection_chain_tipo_2(copia_s1_ej_2);
10. ottimizzazione_soluzione(copia_s1_ej_2);
11. aggiorna_soluzione(copia_s1_ej_2, s1);


12. i = 0;
13. while (i < 100)
    14. copia_s1_ej_3 = copia_soluzione(copia_s1);
    15. applica_ejection_chain_tipo_3(copia_s1_ej_3);
    16. ottimizzazione_soluzione(copia_s1_ej_3);
    17. aggiorna_soluzione(copia_s1_ej_3, s1);
    18. i = i + 1;
19. end


20. ottimizzazione_soluzione(s1);


21. restituisci s1;
```

In riga 1, avviene la creazione della soluzione iniziale, ottenuta applicando l'algoritmo euristico costruttivo I1 (vedi Capitolo 2). Il criterio utilizzato per la scelta del seme, (vertice con cui si inizializza un nuovo percorso), è dato dalla distanza tra ogni vertice ancora non incluso in soluzione e il deposito: il cliente che viene utilizzato per iniziare la costruzione di un nuovo percorso è quello che dista maggiormente dal deposito. Si utilizza questo criterio poiché i vertici che più distano dal deposito sono, solitamente, tra i più difficili da inserire in un percorso già iniziato: la loro inclusione, infatti, può comportare un aumento nelle distanze percorse all'interno dell'itinerario in costruzione. Questo incremento può causare, in alcuni casi, la violazione di alcuni vincoli di finestre temporali sui vertici del percorso, portando ad una non ammissibilità del percorso stesso e rendendo, quindi, impossibile l'inserimento del vertice. Dopo avere determinato il seme di inizializzazione, il percorso creato viene espanso, seguendo i criteri già illustrati in precedenza nel Capitolo 2; la fase di espansione di un percorso continua fino a che è possibile inserire nuovi vertici al suo interno, continuando a mantenerne l'ammissibilità. Ogni volta che l'algoritmo I1 valuta l'inserimento di un vertice da

sincronizzare, si verifica se il vertice duplicato corrispondente sia già stato inserito in soluzione; in caso negativo, non è necessario valutare le possibili alternative per l'impostazione dell'istante di sincronizzazione; in caso contrario, il metodo stabilisce, qualora possibile, l'istante di sincronizzazione, secondo le regole enunciate in precedenza e, successivamente, riduce le finestre temporali di entrambe i vertici in base all'istante di sincronizzazione stabilito.

Quando tutti i vertici sono stati inseriti, l'algoritmo I1 termina, restituendo in "s1" la soluzione iniziale creata.

In riga 2, viene eseguita una copia della soluzione iniziale, memorizzata in "copia_s1".

In riga 3, viene eseguita la procedura di ottimizzazione della soluzione "s1" illustrata nel precedente paragrafo.

Dopo avere effettuato una copia temporanea della soluzione iniziale, in riga 5 il metodo esegue un algoritmo di ejection chain, con lo scopo di diminuire il numero di percorsi della soluzione. Il metodo analizza e prova ad eliminare un determinato percorso della soluzione, inizializzando ogni ejection chain a partire dai vertici del percorso che sta analizzando. Il criterio utilizzato per selezionare il percorso da eliminare consiste nell'individuare l'itinerario col minor numero di vertici non ancora analizzato dal metodo. La procedura di analisi ed eliminazione dei percorsi di una soluzione viene ripetuta, fino a che è possibile eliminare almeno un percorso.

In riga 6, si applica, alla soluzione ottenuta, la procedura di ottimizzazione illustrata nel paragrafo precedente; in riga 7, si aggiorna eventualmente, al valore di questa soluzione, la soluzione iniziale "s1".

Dopo avere effettuato una copia temporanea della soluzione iniziale, in riga 9, il metodo esegue nuovamente un algoritmo di ejection chain. A differenza del metodo precedente, i percorsi da eliminare vengono selezionati ed analizzati in maniera sequenziale, dal primo fino all'ultimo. La procedura di analisi ed eliminazione dei percorsi della soluzione viene ripetuta, fino a che è possibile eliminare almeno un percorso.

In riga 10, si applica, alla soluzione ottenuta, la procedura di ottimizzazione illustrata nel paragrafo precedente; in riga 11, si aggiorna eventualmente, al valore di questa soluzione, la soluzione iniziale “s1”.

Nelle righe 13-19, il metodo esegue un ciclo di 100 iterazioni, in cui, ad ogni iterazione, esegue una procedura di ejection chain su una copia della soluzione iniziale e, successivamente, la procedura di ottimizzazione illustrata in precedenza. La procedura di ejection chain utilizza un ordine di analisi dei percorsi casuale. La procedura di analisi ed eliminazione dei percorsi della soluzione viene ripetuta, fino a che è possibile eliminare almeno un percorso. Al termine di ogni iterazione del ciclo, avviene l’eventuale aggiornamento della soluzione iniziale “s1” al valore della soluzione ottenuta.

Infine, in riga 20, il metodo applica nuovamente l’algoritmo di ottimizzazione alla soluzione iniziale, restituendo, al termine, la soluzione ottenuta.

3.2.3 Applicazione della metaeuristica

La metaeuristica utilizzata per la risoluzione del problema analizzato è ALNS; di seguito si riporta la struttura generale dell’algoritmo.

ALNS

```
1.  genera una soluzione iniziale s
2.  poni s_best = s
3.  ripeti
    4.  scegli una coppia di operatori destroy/repair (d,
        r) utilizzando i pesi adattivi
    5.  applica (d, r) alla soluzione s, ottenendo la
        soluzione s'
    6.  se s' è migliore rispetto a s_best
        7.  poni s_best = s'
        8.  poni s = s'
    9.  altrimenti se s' risponde al criterio di
        accettazione
```

```
            10. poni s = s'
        11. aggiorna i punteggi e i pesi degli operatori
    12. fino a che non viene verificato il criterio di arresto
    13. restituisci s_best
```

Il metodo per la creazione della soluzione iniziale è stato discusso nei precedenti paragrafi. Di seguito, verranno illustrati gli operatori destroy e repair utilizzati, il meccanismo di gestione ed aggiornamento dei pesi degli operatori e il criterio di accettazione utilizzato.

3.2.3.1 Tecnica destroy/repair: principio di utilizzo ed operatori implementati

Il funzionamento della metaeuristica ALNS è basato sull'utilizzo di un framework di ricerca locale, in cui un insieme di diversi algoritmi competono per modificare una data soluzione corrente. In letteratura, questi algoritmi sono noti con l'appellativo *ruin and recreate*, oppure *destroy and repair*. L'idea fondamentale che conduce all'utilizzo di questi algoritmi è quella di individuare soluzioni miglioranti, a partire da una data soluzione, applicando su di essa prima un metodo di distruzione, e successivamente un metodo di ricostruzione, che porti alla determinazione di una nuova soluzione. Nell'ambito dei problemi di routing, generalmente, distruggere una data soluzione significa rimuovere da essa un certo numero di vertici, per poi ricostruire, tramite un operatore di repair, una nuova soluzione, cercando di inserire al suo interno i vertici eliminati, seguendo un criterio prestabilito. In figura 3.1, viene illustrata l'applicazione di una coppia di operatori destroy/repair su un dato percorso di visita ad un insieme di clienti; in figura (a), il cerchio rosso localizza l'insieme di clienti che saranno rimossi dalla soluzione, in figura (b), i clienti vengono effettivamente rimossi, ed infine in figura (c), i clienti sono inseriti nel percorso, permettendo così di individuare e costruire una nuova soluzione.

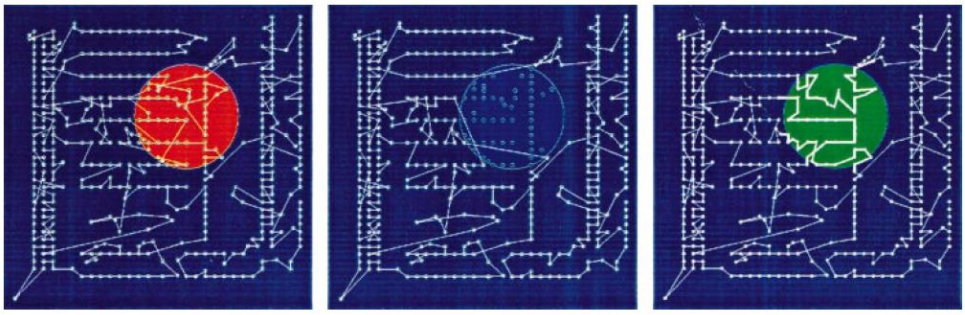

Figura 3.1 Applicazione di una coppia di operatori destroy/repair su un dato percorso di visita a clienti

Diversi metodi destroy/repair sono stati implementati per il funzionamento della metaeuristica ALNS. Il numero di vertici che devono essere rimossi da una data soluzione viene stabilito prima dell'applicazione degli operatori destroy in maniera casuale, scegliendolo all'interno di un intervallo numerico prestabilito. Nel seguito della trattazione, vengono illustrati gli operatori utilizzati.

3.2.3.1.1 Random removal

L'operatore random removal rimuove da una soluzione un dato numero di clienti scelti in maniera casuale.

3.2.3.1.2 Worst removal

L'operatore worst removal elimina i clienti la cui rimozione dalla soluzione deteriora il meno possibile il valore della funzione obiettivo. Se si indica con $f(s)$ il valore della funzione obiettivo di una data soluzione s e si indica con $f_{-i}(s)$ il valore della funzione obiettivo della soluzione s senza il cliente i, il valore

$$d(i, s) = f_{-i}(s) - f(s)$$

fornisce la differenza in termini di valore della funzione obiettivo per rimuovere il cliente i dalla soluzione s.

Il metodo di rimozione calcola, per ogni cliente appartenente alla soluzione, il valore $d(i,s)$ e costruisce una lista di nodi L ordinata in modo crescente rispetto a questo valore. Un numero casuale y viene scelto nell'intervallo [0,1); quindi, il

cliente $i = L[y^{p\_worst}|L|]$ viene rimosso dalla soluzione e i valori *d(i,s)* degli altri clienti vengono aggiornati. Questo procedimento viene ripetuto fino a che non è stato eliminato il numero di clienti stabilito. Il valore *p_worst* costituisce un parametro prestabilito per la metaeuristica.

3.2.3.1.3 Related removal

L'idea centrale alla base del funzionamento dell'operatore related removal è quella di rimuovere da una data soluzione i clienti che sono simili tra loro: è probabile che sia più semplice scambiare tra loro le posizioni di clienti dalle caratteristiche similari piuttosto che clienti molto dissimili. Si definisce, pertanto, una funzione di similarità tra due clienti generici *i* e *j*,

$$R(i,j) = \delta c_{ij} + \phi|B_i - B_j|,$$

dove δ e ϕ corrispondono ai pesi utilizzati per indicare il grado di similarità in termini rispettivamente di costi di viaggio (basati sulla distanza tra i clienti) e di tempi di arrivo presso i clienti. Maggiormente il valore *R(i,j)* è piccolo, maggiormente simili risulteranno essere i clienti *i* e *j*. Il metodo related removal seleziona il primo cliente da rimuovere in maniera casuale; questo cliente viene inserito in una lista di clienti rimossi. Successivamente, ad ogni iterazione del metodo related removal, un cliente casuale *i** viene scelto all'interno della lista dei clienti rimossi; utilizzando *i**, viene costruita una lista di valori di similarità *R(i*,j)* per ogni cliente *j* appartenente alla soluzione. La lista viene ordinata secondo un ordinamento crescente di valori di similarità. Successivamente, viene scelto un numero casuale *y* nell'intervallo [0,1) e viene rimosso dalla soluzione il cliente $i = L[y^{p\_related}|L|]$. Questo procedimento viene ripetuto fino a che non è stato eliminato il numero di clienti stabilito. Il valore *p_related* costituisce un parametro prestabilito per la metaeuristica.

3.2.3.1.4 Cluster removal

Il funzionamento dell'operatore cluster removal si basa sul meccanismo definito dall'operatore related removal. Mentre quest'ultimo operatore si focalizza sul calcolo

delle similarità tra vertici senza considerare se questi appartengano o meno allo stesso percorso, l'operatore cluster removal seleziona vertici appartenenti allo stesso itinerario. Inizialmente, cluster removal individua un percorso *k*, che possieda almeno tre vertici, in maniera casuale; selezionato tale percorso, i suoi vertici vengono separati in due gruppi, detti cluster. Per formare i due raggruppamenti, si costruisce l'albero di copertura minimo sui vertici del percorso selezionato (utilizzando l'algoritmo di Kruskal), e si elimina dall'albero di copertura l'arco più lungo; in questo modo, si ottengono due raggruppamenti di vertici. L'operatore seleziona uno dei due gruppi in maniera casuale; tutti i vertici appartenenti al cluster selezionato vengono eliminati dalla soluzione. Successivamente, l'operatore seleziona un nuovo percorso *k*; per individuare l'itinerario, un vertice *i**, appartenente all'insieme di vertici precedentemente eliminati, viene selezionato in maniera casuale. A questo punto, l'operatore individua il vertice in soluzione più vicino a *i**; il percorso di appartenenza del cliente deve possedere almeno tre vertici e deve essere diverso rispetto a quello a cui apparteneva il vertice *i**. Si indica il vertice individuato con *i*. L'itinerario del cliente *i* diventa il nuovo percorso su cui individuare una suddivisione in due cluster. L'operatore ripete la frammentazione e l'eliminazione dei cluster sui percorsi fino a che non è stato eliminato il numero di vertici prestabilito, oppure fino a che esistono percorsi con almeno tre vertici su cui eliminare un cluster.

3.2.3.1.5 Historical request-pair removal

L'operatore historical request-pair removal basa il suo funzionamento sull'utilizzo di informazioni storiche, riguardanti le soluzioni individuate dalla metaeuristica. In particolare, tale operatore utilizza le informazioni di successo storico derivanti dal collocare su uno stesso percorso coppie di vertici.

Si indica con $h_{(a,b)}$ il peso di una coppia di vertici (a, b) $\in \{1, \ldots, n\} \times \{1, \ldots, n\}$. Il valore $h_{(a,b)}$ indica il numero di volte in cui i clienti *a* e *b* sono stati serviti lungo lo stesso percorso considerando tutte le *B* soluzioni migliori individuate nel corso dell'esecuzione della metaeuristica. Prima che ALNS abbia inizio, tutti i pesi $h_{(a,b)}$ sono posti uguali a zero; ogni volta che ALNS individua una nuova soluzione che rientra nelle *B* soluzioni migliori osservate, i pesi delle coppie di vertici su uno stesso

percorso vengono incrementati. Nel caso in cui le *B* soluzioni migliori siano state tutte individuate, i pesi delle coppie di vertici su uno stesso percorso relativo alla *B*-esima soluzione migliore vengono decrementati, poiché essa non rientrerà più tra le soluzioni migliori considerate.

I pesi $h_{(a,b)}$ definiscono una misura di correlazione tra vertici: maggiore è il valore del peso, maggiore è il grado di correlazione tra i vertici coinvolti. L'operatore historical request-pair removal inizia il processo di rimozione dei clienti da una soluzione selezionando per l'eliminazione un cliente casuale, e aggiungendo tale cliente ad una lista di vertici rimossi. A questo punto, il metodo individua ed elimina dalla soluzione il vertice col massimo grado di correlazione rispetto al cliente appena eliminato; tale vertice viene identificato in base ai valori dei pesi $h_{(a,b)}$ precedentemente descritti. La procedura di individuazione ed eliminazione del vertice maggiormente correlato all'ultimo vertice rimosso dalla soluzione viene ripetuta fino a che non è stato rimosso dalla soluzione il numero di vertici prestabilito.

3.2.3.1.6 Greedy heuristic

Il metodo euristico greedy è una tecnica utilizzata per l'inserimento dei vertici precedentemente eliminati dalla soluzione corrente tramite l'applicazione di un operatore destroy. Questo metodo è iterativo; ad ogni iterazione, calcola il costo minimo di inserimento per ogni vertice da includere in soluzione, determinando, quindi, il percorso e la posizione di inserimento di ogni vertice; successivamente, inserisce in soluzione il vertice per cui il costo di inserimento è minimo.

Si indica con V^0 l'insieme dei vertici precedentemente eliminati dalla soluzione e con $f_\Delta(i, k)$ la variazione del valore della funzione obiettivo dovuta all'inserimento in soluzione del cliente *i* sul percorso *k*; se si ha che $f_\Delta(i, k) = \infty$, significa che il cliente *i* non può essere inserito lungo il percorso *k*. Durante ogni iterazione, il metodo greedy heuristic sceglie il cliente *i** e l'itinerario *k** per cui si verifica che

$$(i^*, k^*) = min_{i \in V^0, k \in K} f_\Delta(i, k)$$

Il metodo greedy heuristic continua l'inserimento dei vertici eliminati dalla soluzione fino a che questi non siano stati inclusi tutti, oppure fino a che non sia possibile inserire altri vertici mantenendo l'ammissibilità della soluzione in costruzione.

3.2.3.1.7 Sequential insertion heuristic

Il metodo sequential insertion esamina in maniera sequenziale ogni percorso della soluzione, cercando di inserire al suo interno i clienti precedentemente rimossi. Il procedimento basa il suo funzionamento sul metodo euristico I1 (vedi Capitolo 2); i criteri definiti dal metodo I1 vengono utilizzati dall'algoritmo sequential insertion per determinare quale cliente, tra tutti quelli eliminati, inserire in un determinato percorso della soluzione. A questo scopo, si utilizzano due criteri $c_1(i, h, j)$ e $c_2(i, h, j)$ per individuare quale vertice includere in soluzione e il punto di inserimento migliore; h corrisponde al cliente da inserire, i e j individuano una particolare posizione di inserimento, ovvero indicano rispettivamente il diretto predecessore e successore del vertice h dopo il suo inserimento in soluzione. I criteri sono definiti secondo le seguenti formule

$$c_1(i, h, j) = \alpha(c_{ih} + c_{hj} - c_{ij}) + (1 - \alpha)(B_{j(new)} - B_j)$$

$$c_2(i, h, j) = 2c_{0h} - c_1(i, h, j)$$

dove $0 \leq \alpha \leq 1$, i valori c_{ih}, c_{hj} e c_{ij} corrispondono alle distanze rispettivamente tra i vertici i e h, h e j e i e j, mentre i valori $B_{j(new)}$ e B_j indicano rispettivamente il tempo di inizio servizio presso il cliente j dopo e prima dell'inserimento di h. Come si può notare, il calcolo della funzione di costo $c_1(i, h, j)$ può essere ricondotto al caso più generale del calcolo del termine $c_1(i, u, j)$ enunciato durante l'algoritmo I1: infatti, per ottenere la formulazione di $c_1(i, h, j)$ è sufficiente porre la condizione $\mu = 1$.

Si ha, quindi, che il metodo sequential insertion determina, per ogni cliente da includere in soluzione, il valore minimo della funzione di costo $c_1(i, h, j)$, calcolata su ogni coppia di vertici adiacenti di un determinato percorso; in questo modo, il metodo individua il miglior punto di inserimento per ogni cliente sul percorso

considerato. Successivamente, il metodo sceglie quale tra tutti i clienti inserire in soluzione, basandosi sul valore del costo $c_2(i, h, j)$; questo termine indica il vantaggio che deriva dal servire il cliente *h* nel percorso attualmente analizzato, piuttosto che in un percorso formato da un unico cliente. Come si può notare, anche questo termine di costo può essere ricavato a partire dalla formulazione più generale utilizzata nel metodo I1 per il calcolo dell'espressione $c_2(i, u, j)$: infatti, è sufficiente porre il parametro $\lambda = 2$ per ottenere la formulazione del costo $c_2(i, h, j)$ utilizzata nell'algoritmo sequential insertion. A questo punto, il metodo inserisce in soluzione il cliente che massimizza il valore della funzione di costo $c_2(i, h, j)$.

Il metodo sequential insertion analizza ogni percorso della soluzione sequenzialmente e termina quando non è più possibile inserire clienti all'interno dei percorsi, oppure quando tutti i clienti sono stati inclusi in soluzione.

3.2.3.1.8 Regret heuristic

Il metodo euristico regret utilizza lo stesso principio alla base del funzionamento dell'operatore di inserimento greedy visto in precedenza; inoltre, tale metodo integra al proprio interno una componente di look ahead, chiamata "regret". L'utilizzo di questa componente permette di individuare, per ogni cliente, il punto di inserimento migliore su ogni percorso della soluzione, e di fornire un ordinamento crescente dei percorsi calcolato in base ai costi di inserimento del cliente al loro interno.

Si indichi con $f_\Delta(i, k)$ il cambiamento del valore della funzione obiettivo, derivato dall'inserimento del cliente *i* nella posizione più conveniente all'interno del *k*-esimo percorso migliore di inserimento. La versione base dell'operatore regret, ovvero regret-2, inserisce in soluzione il cliente per cui la differenza tra l'inserimento nel percorso più vantaggioso nella posizione migliore e l'inserimento nel secondo percorso più vantaggioso nella posizione migliore è massima. In questo modo, la nozione di regret può essere estesa a casi più generali di regret-*q*, dove *q* indica il numero di percorsi da considerare. Ad ogni iterazione, il metodo regret individua il prossimo cliente da inserire in soluzione *i** come

$$i^* = max_{i \in V^0} \left\{ \sum_{k=2}^{\min(q,m)} \left(f_\Delta(i,k) - f_\Delta(i,1) \right) \right\}$$

dove il valore del parametro *m* corrisponde al numero di percorsi presenti nella soluzione e *q* dipende da quale metodo di regret è stato implementato. Il metodo termina quando tutti i clienti sono stati inseriti in soluzione, oppure quando non è più possibile inserire altri clienti mantenendo l'ammissibilità della soluzione. L'algoritmo metaeuristico progettato utilizza al suo interno l'operatore regret-2.

3.2.3.2 Criteri di scelta degli operatori destroy/repair da applicare

Come illustrato precedentemente, ALNS utilizza un dato insieme di operatori destroy/repair in maniera competitiva, con lo scopo di individuare soluzioni miglioranti all'interno dello spazio delle soluzioni. Nel seguito, si illustrano le politiche utilizzate dalla metaeuristica per stabilire quale coppia di operatori utilizzare durante ogni sua iterazione.

L'implementazione di ALNS prevede l'utilizzo di ogni abbinamento di operatori destroy {random removal, worst removal, related removal, cluster removal, historical request-pair removal} con operatori di repair {greedy heuristic, sequential insertion, regret heuristic}. Ogni abbinamento di operatori possiede un proprio "punteggio", un proprio "peso" e una propria probabilità di utilizzo; il punteggio è rappresentativo dell'efficacia che una data coppia di operatori possiede nella ricerca di soluzioni miglioranti, mentre il peso definisce l'importanza della coppia di operatori in relazione a tutte le altre coppie e viene calcolato in base al punteggio ottenuto dalla coppia stessa. La probabilità di scegliere una determinata coppia di operatori è proporzionale al peso ρ_{dr} per ogni coppia (d, r) di operatori destroy/repair. Si indicano con n_d e n_r rispettivamente il numero di operatori destroy e di operatori repair utilizzati dalla metaeuristica; la probabilità φ_{dr} di scegliere una data coppia di operatori (d, r) è calcolata come

$$\varphi_{dr} = \rho_{dr} / \left(\sum_{d'=1}^{n_d} \sum_{r'=1}^{n_r} \rho_{d'r'}\right)$$

La scelta dell'operatore da utilizzare viene eseguita usando un meccanismo di selezione a roulette. Si collochino le probabilità di scelta di ogni operatore su un segmento; si ha che la somma di tutte le probabilità, ovvero la lunghezza del segmento, risulta essere pari a 1. A questo punto, il meccanismo di selezione a roulette sceglie un numero casuale nell'intervallo [0, 1]; il procedimento determina, quindi, la coppia di operatori destroy/repair da applicare in base all'intervallo in cui è compreso il numero estratto. La figura 3.2 mostra il funzionamento del procedimento appena descritto; gli operatori A, B, C, D, E, F, e G sono disposti lungo un segmento di lunghezza 1. Un numero *r* viene estratto casualmente; poiché il suo valore ricade nell'intervallo coperto dall'operatore C, proprio questo operatore viene scelto dalla procedura di selezione.

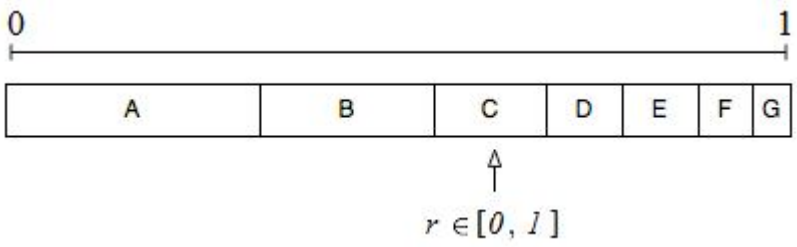

Figura 3.2 Rappresentazione del meccanismo di scelta degli operatori destroy/repair

I pesi degli operatori vengono aggiornati in base al valore dei punteggi corrispondenti. Prima dell'esecuzione di ALNS, i pesi ρ_{dr} di ogni coppia di operatori valgono 1 e i loro punteggi ψ_{dr} valgono 0; al termine di ogni iterazione della metaeuristica, i punteggi ψ_{dr} di ogni coppia di operatori (*d*, *r*) vengono aggiornati secondo le seguenti casistiche:

$$\psi_{dr} = \begin{cases} \psi_{dr} + \sigma_1, & \text{se la coppia di operatori destroy/repair ha individuato una soluzione migliore rispetto alla soluzione migliore fino a quel momento identificata dalla metaeuristica} \\ \psi_{dr} + \sigma_2, & \text{se la coppia di operatori destroy/repair ha individuato una soluzione mai identificata fino a quel momento e che risulta migliore rispetto alla soluzione corrente} \\ \psi_{dr} + \sigma_3, & \text{se la coppia di operatori destroy/repair ha individuato una soluzione mai identificata fino a quel momento e che è stata accettata come nuova soluzione corrente, nonostante fosse di qualità peggiore} \\ \psi_{dr}, & \text{altrimenti} \end{cases}$$

Figura 3.3 Modalità di aggiornamento dei punteggi per gli operatori destroy/repair

σ_1, σ_2 e σ_3 sono parametri il cui valore è predefinito. Per mantenere uno storico delle soluzioni individuate dalla metaeuristica, per ciascuna di esse si mantiene traccia del numero di percorsi e della loro lunghezza totale. Due soluzioni possono essere considerate uguali se la differenza tra il valore delle loro funzioni obiettivo è inferiore o uguale a 10^{-4}.

I pesi ρ_{dr} vengono aggiornati al termine di un certo segmento temporale prestabilito; durante questo intervallo, le coppie di operatori destroy/repair accumulano un proprio punteggio secondo i criteri discussi in precedenza. Al termine del segmento temporale di valutazione, i pesi di tutte le coppie di operatori vengono aggiornati in base ai punteggi ottenuti, secondo la seguente formula

$$\rho_{dr} = (1 - p_{react})\, \rho_{dr} + p_{react}\, (\psi_{dr} / \max(1, \theta_{dr}))$$

dove Θ_{dr} indica il numero di iterazioni in cui la coppia di operatori destroy/repair (d, r) è stata utilizzata durante l'ultimo segmento temporale; p_{react} è un parametro dal valore prestabilito. All'inizio di ogni nuovo segmento temporale, i valori di punteggio delle coppie di operatori vengono azzerati.

3.2.3.3 Componente di diversificazione

Per poter introdurre un grado di diversificazione nella ricerca all'interno dello spazio delle soluzioni del problema, è stato utilizzato un elemento di rumore sui dati; questa componente interviene durante l'esecuzione delle procedure di repair (greedy insertion, sequential insertion, regret-2) per introdurre rumore sul valore della funzione obiettivo utilizzata dagli operatori per la scelta del prossimo cliente da inserire in soluzione. Quando una procedura di repair calcola il costo di inserimento C di un cliente all'interno della soluzione, viene generato un numero casuale p_{noise} nell'intervallo [- η $max_{i,j \in V}$ (c_{ij}), η $max_{i,j \in V}$ (c_{ij})]; tale valore viene aggiunto al costo C, ottenendo un costo C'. L'utilizzo della componente di diversificazione viene deciso utilizzando un meccanismo adattivo analogo a quello descritto ed utilizzato per la scelta degli operatori destroy/repair: l'utilizzo e il non utilizzo della componente di diversificazione possiedono un punteggio, un peso e una probabilità di utilizzo. Durante un segmento temporale di lunghezza prefissata, si mantiene traccia di quanto spesso l'utilizzo della componente di diversificazione e il suo non utilizzo abbiano permesso di individuare soluzioni miglioranti. Al termine di ogni segmento temporale, i pesi e le probabilità di utilizzo e non utilizzo della componente di diversificazione vengono aggiornati secondo i punteggi ottenuti.

3.2.3.4 Criteri di accettazione

ALNS è integrata all'interno di un framework di Simulated Annealing. Una soluzione *s'* è sempre accettata se risulta migliore di *s*, ovvero se $f(s') < f(s)$. Nel caso in cui *s'* sia peggiore rispetto a *s*, *s'* sostituisce *s* con una probabilità pari a

$$e^{-\frac{f(s\prime)-f(s)}{T}}$$

dove il parametro T indica la temperatura corrente; il valore di temperatura iniziale viene impostato utilizzando il valore di costo della soluzione iniziale. Indicando tale valore con $f_c(s)$, si ha che la temperatura iniziale vale

$$T = -\frac{\omega_T}{\ln 0.5} f_c(s)$$

dove ω_T è un parametro predefinito; la temperatura T viene decrementata durante ogni iterazione della metaeuristica utilizzando un parametro dal valore prefissato.

3.2.3.5 Gestione dei vincoli di sincronizzazione durante ALNS

Per poter gestire i vincoli di sincronizzazione che caratterizzano alcuni vertici del problema, durante l'esecuzione di ALNS, sono state realizzate alcune routine di controllo, integrate all'interno degli operatori destroy/repair implementati. Il compito di questi procedimenti è quello di permettere la rinegoziazione dell'istante di sincronizzazione per alcuni clienti, permettendo così di migliorare la qualità delle soluzioni ottenute durante l'esecuzione della metaeuristica.

Ogni volta che un vertice su cui è presente un vincolo di sincronizzazione viene rimosso dalla soluzione corrente, la sua finestra temporale, attualmente impostata in base all'istante di sincronizzazione per lui stabilito, viene riaperta e ripristinata ai suoi valori originali. Ricordando che, per rappresentare il vincolo di sincronizzazione, i vertici che richiedono questo tipo di visita vengono modellati come due vertici distinti, si ha che ogni operatore destroy possiede una routine di verifica, che controlla se almeno uno di questi due vertici è stato rimosso dalla soluzione; in caso affermativo, è possibile impostare ai loro valori originali le finestre temporali di entrambe i vertici (indipendentemente dal fatto che entrambe o solo uno dei due vertici sia stato rimosso dalla soluzione). La routine di controllo sui vertici rimossi dalla soluzione viene eseguita al termine delle operazioni di rimozione per ogni operatore destroy utilizzato. Quando uno dei due vertici da sincronizzare viene valutato per l'inserimento in soluzione, si verifica se l'altro vertice corrispondente appartenga o meno alla soluzione; in caso affermativo, è necessario impostare l'istante di sincronizzazione (valutando tutte le possibili alternative di inserimento) e, successivamente, qualora un inserimento sia ammissibile, ridurre l'ampiezza delle finestre temporali in base al valore dell'istante di sincronizzazione stabilito; in caso negativo, l'inserimento del vertice può essere effettuato come

l’inserimento di un qualsiasi altro vertice su cui non insistono vincoli di sincronizzazione.

La rinegoziazione dell’istante di sincronizzazione durante l’esecuzione di ALNS riveste una grande importanza per il processo di individuazione di soluzioni miglioranti. Nel caso in cui almeno un vertice di una coppia di vertici da sincronizzare venga rimosso dalla soluzione, i valori delle finestre temporali di entrambe i vertici vengono ripristinati ai valori originali. Posticipare l’applicazione del vincolo di sincronizzazione al momento dell’inserimento in soluzione del secondo vertice da sincronizzare permette di rendere più flessibile l’impostazione dell’istante di sincronizzazione; questo consente la valutazione di migliori alternative per l’impostazione dell’istante di sincronizzazione; inoltre, ciò rende potenzialmente più agevole anche l’inserimento degli altri vertici eliminati dalla soluzione, la cui inclusione non sarà condizionata dal vincolo di sincronizzazione presente, fino a che questo non venga stabilito con la riduzione delle finestre temporali.

3.2.4 Definizione della componente esatta e suo utilizzo

L’algoritmo metaeuristico progettato comprende al suo interno l’utilizzo di una componente esatta. L’utilizzo di tale componente ha come scopo quello di trattare in maniera più efficace i vincoli di sincronizzazione che caratterizzano il problema; in particolare, il suo compito è quello di migliorare il posizionamento, all’interno dei percorsi in cui sono inseriti, dei vertici da sincronizzare. Come spiegato in precedenza, le finestre temporali dei vertici su cui è presente un vincolo di sincronizzazione vengono rappresentate come istanti nel tempo; la componente esatta realizzata prevede di migliorare, qualora possibile, il settaggio dell’istante di sincronizzazione.

L’algoritmo progettato prevede di utilizzare tale componente all’interno della metaeuristica ALNS ogni volta che viene individuata una nuova soluzione migliore. Lo schema generale di gestione della sincronizzazione è il seguente: per ogni vertice su cui insiste un vincolo di sincronizzazione, si individuano i due percorsi che lo

raggiungono. Successivamente, la finestra temporale del vertice, ridotta ora ad un istante nel tempo, viene ripristinata al suo valore originale; ciò viene fatto allo scopo di individuare l'istante di sincronizzazione migliore per il vertice. Analizzando i due percorsi che si occupano di servire il vertice sincronizzato, la componente esatta individua, qualora non sia già stato determinato, l'istante di sincronizzazione migliore per il vertice all'interno della coppia di percorsi in cui è inserito. Successivamente, la finestra temporale del vertice viene nuovamente ridotta ad un istante nel tempo, in accordo con il nuovo posizionamento individuato; si provvede, quindi, all'aggiornamento della soluzione migliore corrente e l'esecuzione della metaeuristica ALNS viene ripristinata a partire dalla soluzione appena individuata. Nel seguito del capitolo, verrà spiegato il funzionamento della componente esatta utilizzata.

3.2.4.1 Implementazione della componente esatta

La componente esatta, utilizzata per migliorare il posizionamento dei vertici sincronizzati nei percorsi di inserimento, è stata implementata utilizzando un approccio che si basa su tecniche di programmazione dinamica; queste metodologie vengono spesso utilizzate per la risoluzione di problemi complessi attraverso un procedimento di scomposizione in sotto problemi più semplici.

Per individuare in maniera esatta il posizionamento migliore di un vertice sincronizzato all'interno della coppia di percorsi da cui è servito, è necessario definire lo spazio degli stati del problema, ovvero individuare tutti i possibili collocamenti del vertice sincronizzato all'interno dei percorsi. Per fare ciò, bisogna individuare tutti i possibili stadi di avanzamento delle visite ai vertici su entrambe i percorsi, e prevedere, in ogni stadio di avanzamento opportuno, il collocamento della visita al cliente sincronizzato. Per rappresentare gli stadi di avanzamento e, quindi, lo spazio degli stati, viene definito un grafo di programmazione dinamica, detto G_{DP}; i nodi del grafo individuano tutti i possibili stadi di avanzamento delle visite su entrambe i percorsi, mentre gli archi indicano lo spostamento da uno stadio ad un altro.

Si supponga di esaminare due percorsi r_1 ed r_2, all'interno dei quali è presente un vertice sincronizzato *s*; si assuma che i percorsi r_1 ed r_2 siano rispettivamente composti dai seguenti vertici *a-b-c* e *d-e-f-g*. Si definisce il grafo che individua lo spazio degli stati del problema come quello in figura 3.4.

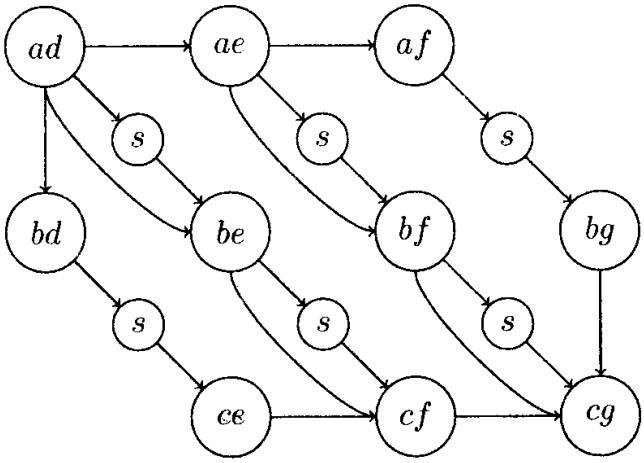

Figura 3.4 Grafo dello spazio degli stati

Ogni nodo del grafo corrisponde ad una coppia di vertici, uno appartenente al percorso r_1 e un altro ad r_2. Gli archi orizzontali del grafo individuano gli spostamenti verso il vertice successivo a quello attualmente visitato sul percorso r_2 rimanendo nello stesso vertice sul percorso r_1; gli archi verticali del grafo individuano gli spostamenti verso il vertice successivo a quello attuale sul percorso r_1 rimanendo nello stesso vertice sul percorso r_2. Gli archi diagonali individuano spostamenti su entrambe i percorsi. I nodi *s* posti sugli archi diagonali individuano l'inclusione nei percorsi del vertice sincronizzato. Si ha, quindi, che ogni nodo del grafo dello spazio degli stati è connesso con al massimo quattro nodi: il nodo successivo in senso orizzontale, il nodo successivo in senso verticale, il nodo diagonale e il nodo che individua l'inclusione del vertice sincronizzato. Come si può notare, tuttavia, non tutti i nodi presentano quattro archi. Il nodo iniziale, che individua lo stato di partenza dei percorsi dal deposito, prevede collegamenti verso tutti i quattro nodi descritti, poiché è possibile avanzare verso il vertice successivo su uno o su entrambe i percorsi; inoltre, è possibile includere subito la visita al vertice sincronizzato. Tutti i nodi del grafo che comprendono un vertice di origine di un

percorso ed un vertice qualsiasi sull'altro percorso non possiedono i quattro archi: a partire da questi nodi è possibile spostarsi verso uno stato che prevede un avanzamento su entrambe i percorsi, verso uno stato in cui l'avanzamento riguarda solo il percorso a cui non appartiene il vertice di origine e verso uno stato in cui avviene l'inclusione del vertice sincronizzato. Esistono due eccezioni a questa regola di costruzione del grafo. I due nodi che rappresentano la combinazione di un vertice di origine su un percorso e di un vertice di destinazione sull'altro non sono connessi a nessun nodo del grafo e non sono raggiungibili da nessun nodo del grafo: se, per assurdo, lo stato di avanzamento delle visite su entrambe i percorsi giungesse a uno di questi due nodi, ciò renderebbe impossibile l'inserimento del vertice sincronizzato e, quindi, porterebbe ad ottenere una soluzione non ammissibile al problema del collocamento di tale vertice all'interno dei percorsi considerati. Un'altra eccezione è individuabile considerando i due stati che comprendono un vertice di origine di un percorso e il penultimo vertice sull'altro percorso; i nodi corrispondenti a questi stati sono connessi solo al nodo di inclusione del vertice sincronizzato. Infine, tutti gli altri nodi del grafo sono connessi al successivo nodo in diagonale e al nodo che prevede l'inclusione del vertice sincronizzato.

In ogni nodo del grafo G_{DP} vengono aggiornati i valori di costi e tempi di visita calcolati per il corrispondente stadio di avanzamento di visita su entrambe i percorsi; quando si giunge al nodo di inclusione di un vertice sincronizzato, avviene il settaggio dell'istante di sincronizzazione. Individuare il miglior collocamento del vertice sincronizzato all'interno di una coppia di percorsi significa individuare il miglior stato di inclusione del vertice sincronizzato tra tutti quelli possibili; questo problema equivale alla risoluzione di un problema di cammino minimo sul grafo dello spazio degli stati, che può essere risolto utilizzando un algoritmo di settaggio delle etichette. Ogni nodo possiede un'etichetta, la quale memorizza diverse informazioni:

- il nodo v a cui fa riferimento
- il tempo trascorso sul primo percorso fino al vertice corrente, t_{r1}
- il tempo trascorso sul secondo percorso fino al vertice corrente, t_{r2}
- i costi di visita del primo percorso fino al vertice corrente, c_{r1}

- i costi di visita del secondo percorso fino al vertice corrente c_{r2}
- un flag che indica se il vertice sincronizzato è già stato visitato
- un puntatore all'etichetta padre

Si assuma di generare una nuova etichetta η' su un generico nodo j del grafo a partire dall'etichetta η relativa ad un generico nodo i; si indichi con $r_1(i)$ il vertice del primo percorso individuato al nodo i del grafo e si indichi con $r_2(i)$ il vertice del secondo percorso individuato al nodo i. La generazione di una nuova etichetta avviene seguendo le seguenti regole:

$$v(\eta') = j$$

$$t_{r1}(\eta') = \begin{cases} max\,(e'_{r1(j)},\ t_{r1}(\eta) + \tau_{r1(i),r1(j)},\ t_{r2}(\eta) + \tau_{r2(i),r2(j)}), \\ \text{se } j \text{ corrisponde al nodo di inclusione del vertice sincronizzato} \\ max\,(e'_{r1(j)},\ t_{r1}(\eta) + \tau_{r1(i),r1(j)}), \text{ altrimenti} \end{cases}$$

$$t_{r2}(\eta') = \begin{cases} t_{r1}(\eta'), \text{ se } j \text{ corrisponde al nodo di inclusione del vertice sincronizzato} \\ max\,(e'_{r2(j)},\ t_{r2}(\eta) + \tau_{r2(i),r2(j)}), \text{ altrimenti} \end{cases}$$

$$c_{r1}(\eta') = c_{r1}(\eta) + c_{r1(i),r1(j)}$$

$$c_{r2}(\eta') = c_{r2}(\eta) + c_{r2(i),r2(j)}$$

$$sync(\eta') = \begin{cases} 1, \text{ se } j \text{ corrisponde al nodo di inclusione del vertice sincronizzato} \\ sync(\eta), \text{ altrimenti} \end{cases}$$

$$padre(\eta') = \text{puntatore a } \eta$$

dove $e'_{r1(j)}$ ed $e'_{r2(j)}$ corrispondono rispettivamente ai tempi di apertura delle finestre temporali del vertice *r1(j)* (vertice del primo percorso del nodo *j* del grafo) e del vertice *r2(j)* (vertice del secondo percorso del nodo *j* del grafo).

Il termine $\tau_{rk(i),rk(j)}$ indica il tempo trascorso su ogni arco (*i*, *j*) del grafo, e viene calcolato nel seguente modo:

$$\tau_{rk(i),rk(j)} = \begin{cases} s_{rk(i)} + t_{rk(i),rk(j)}, & \text{se } r_{k(i)} \neq r_{k(j)} \quad \forall\, k \in \{1,2\} \\ 0, \text{ altrimenti} \end{cases}$$

Nel nodo iniziale del grafo, si ha che $t_{r1} = e'_{r1(0)}$, $t_{r2} = e'_{r2(0)}$, $c_{r1} = c_{r2} = 0$, $sync = 0$. L'estensione di un'etichetta η lungo l'arco (i,j) è possibile se si verificano le seguenti condizioni:

$$t_{r1}(\eta) + \tau_{r1(i),r1(j)} \leq l'_{r1(j)}$$
$$t_{r2}(\eta) + \tau_{r2(i),r2(j)} \leq l'_{r2(j)}$$

4. Configurazione dei parametri del metodo di risoluzione

4.1 Il problema della configurazione dei parametri di un algoritmo

Molti metodi, che mirano ad offrire risultati altamente competitivi in maniera efficiente, necessitano di una fase preliminare di definizione ed impostazione di un insieme di parametri, la cui configurazione controlla aspetti molto importanti del loro comportamento, influendo spesso in maniera incisiva sui risultati che gli stessi metodi producono [13]. Ciò è particolarmente vero quando si analizzano le procedure euristiche utilizzate per risolvere problemi computazionalmente difficili.

La configurazione di un insieme di parametri per un algoritmo costituisce, di fatto, un problema. Tale problema può essere espresso in maniera informale nel modo seguente: dati un algoritmo, un insieme di parametri per l'algoritmo e un insieme di problemi in input, individuare i valori dei parametri per cui l'algoritmo raggiunge i migliori risultati possibili sui dati in input. La figura 4.1 mostra lo scenario di interesse: è presente un *configuratore*, ovvero un algoritmo progettato ad hoc per risolvere il problema della configurazione dei parametri di esecuzione dell'algoritmo di interesse. Il configuratore riceve in input i domini di appartenenza dei valori assunti da ogni parametro; quindi, invoca l'algoritmo con diverse combinazioni di valori di parametri. L'algoritmo viene eseguito su un insieme di istanze di problemi da risolvere, detto *training set*. Questo gruppo costituisce un sottoinsieme dell'intero set di problemi che l'algoritmo deve risolvere, e viene utilizzato dal configuratore per *addestrare* l'algoritmo al procedimento di risoluzione. L'algoritmo, dunque, viene addestrato, risolvendo, su ogni problema del training set, il problema dell'individuazione della configurazione di parametri migliore. Al termine del ciclo

di addestramento, l'algoritmo individua, per ciascun problema del training set, la configurazione di parametri migliore; per verificare la qualità dei risultati ottenuti, il configuratore applica tali configurazioni sull'intero set di problemi, valutando, in questo modo, le performance delle configurazioni identificate in casi di applicazione reale. Questo processo di verifica è chiamato *validazione*. Al termine della validazione, utilizzando opportune misure statistiche per la valutazione dei risultati, si individua la combinazione migliore di parametri tra tutte quelle analizzate in fase di validazione.

In maniera iterativa, il configuratore può eseguire diversi cicli di training; ogni ciclo sarà caratterizzato da una diversa scelta del training set utilizzato per addestrare l'algoritmo di risoluzione.

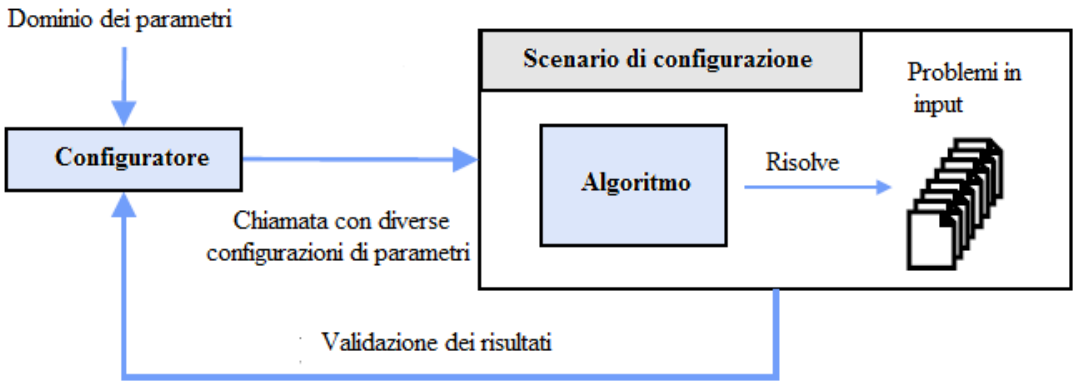

Figura 4.1 Schema di riassunto del processo di configurazione dei parametri di un algoritmo

In maniera più formale, si consideri un algoritmo A e si indichino con $p_1, \ldots, p_k$ i parametri di A; si indichi il dominio di possibili valori di ogni parametro p_i col termine Θ_i; sia $\Theta \subseteq \Theta_1 \times \ldots \times \Theta_k$ lo spazio di tutte le configurazioni di parametri ammissibili e $A(\theta)$ rappresenti l'esecuzione dell'algoritmo A con la configurazione di parametri θ. Per misurare le performance di un algoritmo, eseguito su un insieme di problemi, in alcuni casi non è sufficiente considerare semplicemente le soluzioni individuate, valutandone il costo rispetto alla funzione obiettivo definita dal problema: spesso, infatti, i metodi utilizzati per la risoluzione di problemi complessi possono incorporare alcune componenti di casualità, il cui scopo è quello di aumentare il raggio di analisi dello spazio delle soluzioni che caratterizza il problema

da risolvere. L'uso di questi componenti fa sì che lo stesso algoritmo, eseguito con la stessa configurazione di parametri e su uno stesso problema, non produca sempre le stesse soluzioni. Si rende necessaria, dunque, la scelta di una misura statistica per valutare le performance dell'algoritmo (ad esempio, si possono valutare le prestazioni ottenute dall'algoritmo con una particolare configurazione di parametri utilizzando un valore di media); si indichi con $c(\theta)$ il costo della soluzione individuata calcolato secondo tale misura statistica. Si denoti con D una distribuzione di probabilità sullo spazio Π delle istanze di problemi in input.

Si definisce un'istanza del problema di configurazione di un algoritmo come la seguente sestupla $<A, \Theta, D, k_{max}, o, m>$, dove:

- A è l'algoritmo di interesse
- Θ è lo spazio delle configurazioni di A
- D è una distribuzione di probabilità sullo spazio Π delle istanze di problemi da risolvere
- k_{max} stabilisce la lunghezza del lasso temporale dedicato all'esecuzione dell'algoritmo A
- o è una funzione che misura il costo della soluzione ottenuta eseguendo $A(\theta)$ su un generico problema $\pi \in \Pi$ utilizzando un determinato lasso temporale $k \in \mathrm{R}$
- m indica la misura statistica utilizzata per la valutazione delle prestazioni dell'algoritmo

Ogni configurazione di parametri θ rappresenta una soluzione ammissibile al problema della configurazione dei parametri di un algoritmo. Per ogni configurazione θ, si indica con O_θ la distribuzione dei costi indotta dalla funzione o; θ è applicata su istanze π tratte dalla distribuzione D in multiple esecuzioni, utilizzando un lasso temporale pari a k_{max}. Il costo di θ è definito come

$$c(\theta) = m(O_\theta)$$

Una soluzione ottima θ^* minimizza $c(\theta)$, ovvero

$$\theta^* \in arg\,min_{\theta \in \Theta}\ c(\theta)$$

Come si può notare, il numero di tutte le configurazioni ammissibili per i parametri di un algoritmo cresce in maniera esponenziale rispetto al numero di parametri da configurare; il problema della configurazione di un insieme di parametri può possedere, quindi, uno spazio delle soluzioni di elevata dimensione. Per poter risolvere questo problema in maniera efficiente, è necessario progettare metodi ed algoritmi ad hoc, che permettano, in maniera automatica, di eseguire ampie ricerche nello spazio delle configurazioni, con lo scopo di individuare la miglior configurazione possibile per i parametri dell'algoritmo.

4.2 Configurazione dei parametri per il metodo di risoluzione

Nel capitolo 3, è stato discusso il metodo mateuristico realizzato per la risoluzione del VRPTWPS, illustrando il funzionamento delle più importanti componenti che ne costituiscono la struttura:

- il metodo adottato per la creazione della soluzione iniziale dell'algoritmo
- la metaeuristica ALNS, gli operatori destroy/repair utilizzati e la loro gestione
- la componente esatta, che permette di migliorare una data soluzione del problema

Numerose sono le variabili decisionali che caratterizzano il funzionamento dei metodi utilizzati per la risoluzione del problema. Precisamente, è necessario stabilire il valore dei seguenti parametri e condizioni:

- i parametri α_1, α_2, μ, e λ per il metodo I1, utilizzato per costruire la soluzione iniziale del problema
- il criterio di arresto della metaeuristica
- l'intervallo numerico all'interno del quale scegliere il numero di vertici da eliminare da una data soluzione con un operatore destroy
- il parametro *p_worst*, (utilizzato dall'operatore worst removal)
- i parametri *p_related*, δ e φ (utilizzati dall'operatore related removal)
- *B*, ovvero il valore che indica il numero di soluzioni migliori da considerare per il calcolo dei valori di correlazione $h_{(a,b)}$ tra coppie di vertici, (utilizzati dall'operatore historical request-pair removal)
- i valori σ_1, σ_2 e σ_3, usati per aggiornare i punteggi degli operatori destroy/repair
- la lunghezza del segmento temporale considerato per il monitoraggio delle prestazioni degli operatori destroy/repair e l'aggiornamento dei loro pesi
- il valore *p_react*, impiegato per aggiornare i pesi degli operatori destroy/repair
- il valore ω_T, utilizzato per impostare la temperatura iniziale della metaeuristica e il parametro di decremento del valore di temperatura ad ogni iterazione dell'algoritmo
- il valore η, che permette di stabilire l'entità della componente di rumore presente sui costi di inserimento dei vertici durante l'applicazione degli operatori repair

Per affrontare il problema della configurazione di ciascuno di questi parametri, si è pensato di suddividere l'insieme dei parametri in due sottoinsiemi principali, in base al loro utilizzo nelle diverse fasi dell'algoritmo:

- parametri utilizzati per la costruzione della soluzione iniziale
- parametri utilizzati durante l'esecuzione della metaeuristica

Al primo insieme di parametri appartengono α_1, α_2, μ, e λ, mentre al secondo tutti i restanti. Sulla base di questa suddivisione, sono state adottate diverse scelte di configurazione dei parametri.

4.2.1 Insieme di problemi per la valutazione delle performance dell'algoritmo

Nel contesto dei VRPTW (Vehicle Routing Problems with Time Windows), la modalità più comune per comparare i risultati ottenuti dai metodi euristici è quella di utilizzare, come insieme di valutazione delle performance, il set di problemi definiti da Solomon ([14]). Questo insieme comprende 56 problemi di riferimento; ogni problema possiede un centinaio di clienti e un deposito centrale; stabilisce vincoli di capacità sulla quantità di merce trasportabile dai veicoli, vincoli di tempo massimo di durata totale di ogni percorso di distribuzione, vincoli di tempi di consegna delle merci presso ogni cliente e vincoli di quantità di merce richiesta da ciascuno di essi. I problemi sono suddivisi in sei classi: C1, C2, R1, R2, RC1 ed RC2. Ogni classe di problemi presenta caratteristiche ben precise. I problemi delle classi C1 e C2 possiedono clienti raggruppabili in cluster; le classi R1 ed R2 presentano clienti distribuiti in maniera uniforme nello spazio, mentre i problemi di classe RC1 ed RC2 rappresentano una situazione intermedia, poiché comprendono un sottoinsieme di clienti raggruppati in cluster e un altro sottoinsieme di clienti disposti in maniera uniforme. Ogni classe contiene da 8 a 12 istanze di problemi. Tutti i problemi appartenenti ad una stessa classe comprendono lo stesso insieme di clienti e gli stessi vincoli su ciascuno di essi; solamente le finestre temporali dei clienti differiscono in problemi diversi. I problemi di classe C1, R1 ed RC1 hanno un orizzonte di pianificazione breve; ciò significa che il trasporto delle merci ai clienti viene eseguito utilizzando veicoli con ridotte capacità, che devono terminare i propri viaggi di consegna avendo a disposizione tempi stringenti e che, di conseguenza, non riusciranno a visitare un ampio numero di clienti. La soluzione di questi problemi può richiedere l'impiego di un numero di veicoli a partire da 9 fino a 19. I problemi di classe C2, R2 ed RC2 sono maggiormente rappresentativi delle forniture più a

lungo raggio, con un orizzonte di pianificazione più lungo ed un minor numero di veicoli impiegati per il servizio presso i clienti (genericamente, tra i 2 ed i 4 veicoli).

Poiché i problemi definiti da Solomon non prevedono vincoli di sincronizzazione sui clienti, tali vincoli sono stati introdotti come illustrato in [3]: si assume che, ogni dieci clienti, ci sia una richiesta di visita sincronizzata. Ciò significa che, se il vertice 0 indica il deposito, i vertici {10,20,…,100} richiedono una visita sincronizzata. Si assume che il carico di merce, richiesto per eseguire una visita sincronizzata, sia trasportato da uno solo dei due veicoli richiesti per portare a termine la visita; quindi, se indichiamo con *s* il cliente che domanda una visita sincronizzata, (modellato come due clienti distinti s_1 ed s_2), si ha che la quantità richiesta da s_1 è pari alla quantità richiesta da *s*, mentre s_2 richiede una quantità di merce pari a 0.

4.2.2 Algoritmo di configurazione dei parametri di costruzione della soluzione iniziale

I valori dei parametri α_1, α_2, μ, e λ, utilizzati dal metodo I1 per la costruzione della soluzione iniziale, sono stati configurati progettando un apposito algoritmo. Lo schema del metodo è riportato di seguito:

```
1. ricevi in input il dominio di valori assunti dai
   parametri
2. stabilisci il numero di cicli di training che il
   configuratore dei parametri dovrà eseguire
3. stabilisci la dimensione del training set utilizzato
   durante ogni ciclo di addestramento

4. ripeti
   5. scegli i problemi che comporranno il training set

   6. per ogni problema p' del training set

      7. ripeti
```

8. scegli una configurazione casuale di valori di parametri c
9. esegui l'algoritmo da addestrare sul problema p' utilizzando la configurazione c
10. aggiorna il valore della configurazione $c_{p'}$ che ha prodotto il risultato migliore su p'
11. fino a che non si verifica la condizione di arresto

12. per ogni problema p dell'intero insieme di problemi

13. per ogni configurazione $c_{p'}$ individuata sul problema di training p'

14. esegui l'algoritmo da addestrare sul problema p utilizzando la configurazione $c_{p'}$

15. determina la configurazione $c_{p'*}$, individuata durante l'addestramento, che fornisce le migliori performance, valutando i risultati ottenuti in fase di validazione sull'intero insieme di problemi
16. aggiorna eventualmente il valore della migliore configurazione di parametri $C*$, individuata durante i diversi cicli di training, al valore della configurazione $c_{p'*}$

17. fino a che non sono terminati i cicli di training

Al passo 1, l'algoritmo riceve come input il dominio di valori di ogni parametro da configurare; utilizzando questi intervalli, il configuratore potrà stabilire quali valori poter assegnare a ciascuno di essi durante il procedimento di addestramento

dell'algoritmo di risoluzione. Gli intervalli di valori considerati per ogni parametro sono i seguenti:

- $0 \leq \alpha_1 \leq 1$
- $\alpha_2 = 1 - \alpha_1$
- $0{,}5 \leq \mu \leq 1{,}5$
- $0{,}5 \leq \lambda \leq 2{,}5$

Al passo 2, il configuratore stabilisce il numero di cicli di training da eseguire; questo valore viene stabilito in maniera casuale, scegliendo un numero di cicli compreso tra 5 e 10.

Il configuratore individua la dimensione del training set, scegliendo tale valore in maniera casuale; sono stati considerati training set di dimensione compresa tra 5 e 10.

I passi compresi tra 4 e 17 individuano le operazioni eseguite durante un ciclo di training; la prima operazione attuata consiste nella scelta dei problemi che comporranno il training set del ciclo di addestramento (passo 5). Il configuratore seleziona questi problemi utilizzando un meccanismo di scelta casuale, che non prevede la possibilità di selezionare uno stesso problema più di una volta. I passi compresi tra 6 e 11 individuano le operazioni di addestramento dell'algoritmo di risoluzione su un particolare problema del training set; il procedimento di addestramento è iterativo e termina quando sono state completate 1000 iterazioni. Durante ogni iterazione, il configuratore addestra l'algoritmo con configurazioni di parametri sempre diverse. La configurazione iniziale viene stabilita scegliendo un valore casuale per ogni parametro; durante ogni iterazione, la configurazione corrente viene stabilita, a partire dalla configurazione utilizzata nella precedente iterazione, cambiando il valore di un solo parametro, scelto in maniera casuale. Quando ha invocato l'algoritmo da addestrare applicando la configurazione scelta, il configuratore valuta la qualità della soluzione ottenuta, mantenendo traccia della configurazione che ha permesso di ottenere la soluzione di miglior qualità. Il primo criterio per confrontare due soluzioni è dato dal numero di veicoli impiegati per

servire l'insieme dei clienti: la soluzione che viene considerata migliore è quella che impiega un numero di veicoli inferiore per servire tutti i clienti. A parità di veicoli utilizzati, la soluzione considerata migliore è quella in cui la distanza totale percorsa dai veicoli risulta inferiore.

Si ha, quindi, che, al termine del ciclo di training, il configuratore ha individuato, per ogni problema del training set, la configurazione che ha permesso di ottenere su quel problema la soluzione migliore. I passi compresi tra 12 e 14 includono le operazioni eseguite per validare i risultati, ottenuti durante la fase di addestramento, sull'intero insieme di problemi.

Al passo 15, il configuratore determina quale configurazione, tra tutte quelle individuate in fase di addestramento, permette di ottenere i risultati migliori sull'intero insieme di problemi. Per valutare la qualità delle soluzioni ottenute, si utilizzano valori di lower bound. Per ogni configurazione di parametri:

1. si calcolano sull'intero set di problemi le differenze totali, (in termini di numero di veicoli e di distanze), tra le soluzioni ottenute in fase di validazione e i valori di lower bound di ogni problema
2. si calcolano i valori di differenza medi a partire dalle differenze totali

Il configuratore mantiene traccia della configurazione che permette di minimizzare i valori medi di differenza. Come misura di lower bound, sono stati utilizzati i valori delle migliori soluzioni, (numero di veicoli e distanza totale), individuate sui problemi definiti da Solomon.

Dopo avere determinato la migliore configurazione del ciclo di training terminato, al passo 16, il configuratore aggiorna, eventualmente, al valore della configurazione appena determinata, la configurazione migliore individuata durante tutti i cicli di training.

4.2.3 Configurazione dei parametri utilizzati dalla metaeuristica

Il criterio utilizzato per definire la condizione di arresto per l'esecuzione della metaeuristica ALNS si basa sul numero di iterazioni concluse: come stabilito in [3], la metaeuristica termina la propria esecuzione dopo avere completato 25000 iterazioni.

Come in [15], il numero di clienti da rimuovere da una data soluzione utilizzando un operatore destroy viene stabilito scegliendo in maniera casuale un valore compreso nell'intervallo $[min\{0.1|V|, 30\}, min\{0.4|V|, 60\}]$, dove V indica l'insieme dei vertici del problema.

I valori dei parametri *p_worst* (worst removal), *p_related*, δ, φ (related removal), σ_1, σ_2, σ_3 (valori di aggiornamento dei punteggi degli operatori destroy/repair), *p_react* (valore utilizzato per l'aggiornamento dei pesi degli operatori destroy/repair), ω_T (valore di impostazione della temperatura iniziale), decremento del valore di temperatura, η (entità della componente di rumore presente sui dati) sono stati impostati secondo [15]:

Parametro	δ	φ	*p_related*	*p_worst*	σ_1	σ_2	σ_3	*p_react*	η	ω_T	Dec. temp.
Valore	9	3	6	3	33	9	13	0.1	0.025	0.05	0.99975

Tabella 4.1 Valori di configurazione per i parametri della metaeuristica

La lunghezza del segmento temporale utilizzato per l'aggiornamento dei pesi degli operatori destroy/repair e della componente esatta è pari a 100 iterazioni.

Il valore *B*, che indica il numero di soluzioni migliori da considerare per il calcolo dei valori di correlazione $h_{(a,b)}$ tra coppie di vertici, utilizzato dall'operatore historical request-pair removal, è stato stabilito pari a 100, secondo [11].

5. Sperimentazioni del metodo di risoluzione

5.1 Istanze di problemi utilizzate

Al termine della fase implementativa, sono state realizzate diverse sperimentazioni, per potere valutare le performance dell'algoritmo realizzato; le verifiche delle prestazioni ottenute sono state eseguite su due tipologie di problemi:

- istanze di problemi Solomon
- istanza di un problema reale

Il primo insieme di problemi è stato introdotto durante la trattazione del precedente capitolo.

L'istanza del problema reale utilizzata è relativa ad un problema di assistenza tecnica: una squadra di tecnici deve fornire un servizio di assistenza presso il domicilio dei propri clienti. Ogni tecnico effettua le visite presso i clienti in maniera autonoma ed indipendente rispetto agli altri tecnici; ciascuno di loro inizia il proprio itinerario di visite partendo da un deposito centrale. I clienti possono ricevere visite dai tecnici solamente all'interno di una finestra temporale prestabilita; un sottoinsieme di essi richiede, inoltre, che il servizio di assistenza venga effettuato da esattamente una coppia di tecnici. I clienti che compongono l'istanza del problema studiato sono 86, di cui 32 richiedenti una visita sincronizzata; essi sono dislocati principalmente nella zona della Romagna (province di Ravenna, Forlì-Cesena, Rimini); altri clienti risiedono invece in provincia di Arezzo e Pesaro-Urbino. In figura 5.1 si mostrano la rete stradale e i clienti dell'istanza del problema; i punti rossi individuano i clienti.

Figura 5.1 Insieme di clienti dell'istanza reale

5.2 Tecnologie e strumenti utilizzati per la realizzazione dell’algoritmo

L’algoritmo di risoluzione del VRPTWPS è stato implementato utilizzando il linguaggio di programmazione C# all’interno dell’ambiente di sviluppo Microsoft Visual Studio 2010. Affinché l’algoritmo possa essere facilmente utilizzabile per risolvere i problemi di trasporto trattati, è stata realizzata una Windows Forms Application, che, tramite interfaccia grafica, permette all’utente dell’applicazione di svolgere diverse azioni. Come mostra la figura 5.2, l’interfaccia grafica si compone di quattro bottoni, due pannelli principali e una barra di menù. I bottoni presenti sono i seguenti:

- Carica Istanza Solomon; permette all’utente, tramite apposita finestra di dialogo, di selezionare il file corrispondente ad uno tra i problemi Solomon

da risolvere e di caricarne i dati nell'applicazione; dopo avere terminato il caricamento dei dati, nel pannello "Soluzione Iniziale", è possibile visualizzare l'insieme di vertici del problema caricato

- ALNS; esegue l'algoritmo di risoluzione realizzato per risolvere il problema Solomon caricato nell'applicazione
- Esporta Soluzione; al termine dell'esecuzione dell'algoritmo, permette l'esportazione su file di testo della soluzione del problema, individuata dall'algoritmo
- Risolvi Istanza Reale; permette di caricare nell'applicazione i dati di un'istanza di VRPTWPS reale e di utilizzare il metodo di risoluzione creato per risolvere tale istanza

L'interfaccia è caratterizzata dalla presenza di due pannelli:

- Soluzione Iniziale; consente di visualizzare l'insieme di vertici del problema e di raffigurare la soluzione di partenza calcolata dall'algoritmo di risoluzione
- Soluzione Finale; visualizza la soluzione determinata dall'algoritmo

La barra di menù contiene un elemento "File", il cui menù consente all'utente dell'applicazione di selezionare e caricare i dati dei problemi Solomon e di esportare su file la soluzione individuata per il problema.

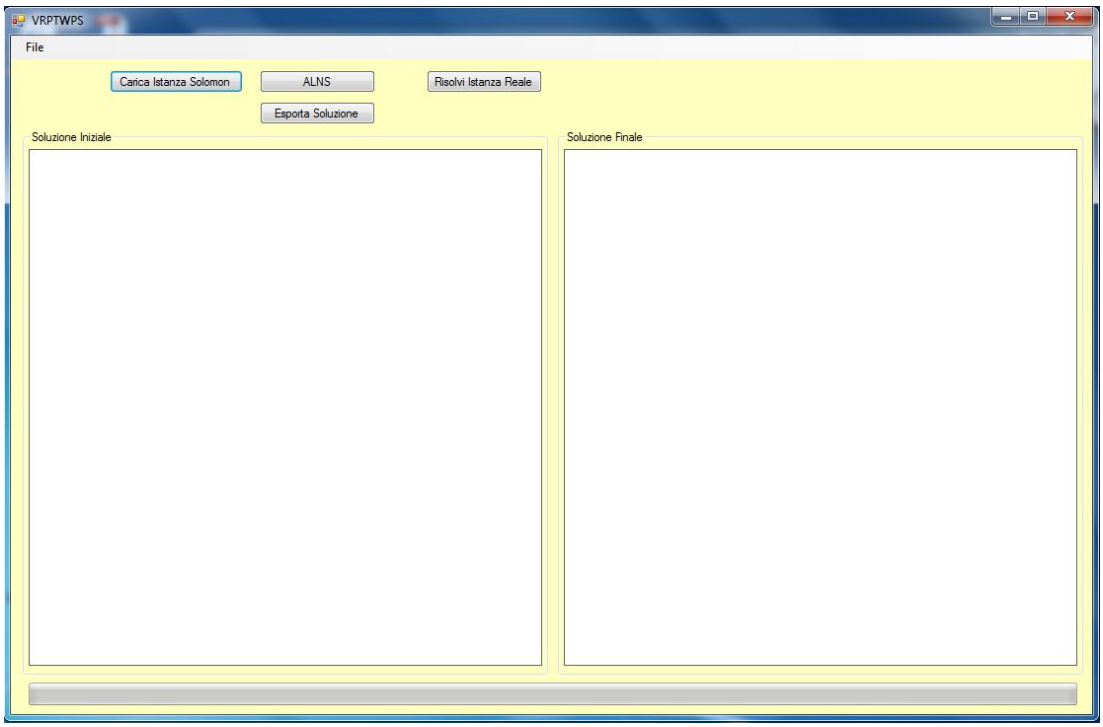

Figura 5.2 Interfaccia grafica dell'applicazione realizzata

5.3 Formato dei file di input e output

5.3.1 File di input per i problemi Solomon

I dati dei problemi del dataset di Solomon sono memorizzati su file di testo, col seguente formato:

<nome problema>

<capacity>

<cust no.> <xcoord.> <ycoord.> <demand> <ready time> <due date> <service time>

. . .

. . .

<cust no.> <xcoord.> <ycoord.> <demand> <ready time> <due date> <service time>

Il campo “capacity” indica la capacità dei veicoli utilizzati per il trasporto. A seguire, il file specifica su ogni riga i dati di un singolo cliente: il campo “cust no.” determina il numero identificativo del cliente; “xcoord” e “ycoord” individuano, rispettivamente, le coordinate x ed y del cliente; “demand” determina la quantità di merce che il cliente richiede; “ready time” e “due date” indicano, rispettivamente, l’istante nel tempo in cui la finestra temporale del cliente ha inizio e fine; “service time” determina la durata nel tempo del servizio di consegna merce svolto presso il cliente. In figura 5.3, si mostra il contenuto del file in riferimento ai primi 5 vertici del problema C101; il vertice con identificativo 0 corrisponde al deposito.

```
C101

CAPACITY
200

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
```

Figura 5.3 Dati dei primi 5 vertici del problema C101

5.3.2 File di input per l’istanza di problema reale

I dati relativi al problema reale analizzato sono memorizzati all’interno di un file con estensione .csv. Ogni riga del file individua in maniera distinta i dati di un cliente, memorizzando, nell’ordine, le informazioni relative a:

- identificativo
- coordinate geografiche
- numero di tecnici necessari per il servizio
- orari di apertura e chiusura delle finestre temporali (espressi in numero di minuti a partire da mezzanotte)
- durata del servizio (espressa in minuti)

In figura 5.4 si mostra il dettaglio dei dati relativi a 4 vertici del problema reale.

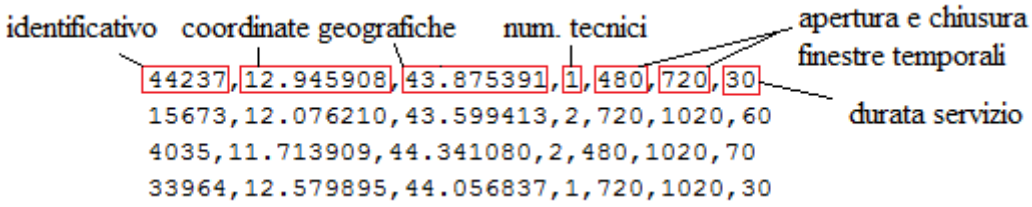

Figura 5.4 Dati di 4 vertici del problema reale

5.3.3 File di output della soluzione

Al termine dell'esecuzione dell'algoritmo, l'applicazione permette di esportare su file di testo i dati relativi alla soluzione determinata. Le informazioni che vengono riportate su file riguardano la composizione dei percorsi della soluzione, indicando, per ciascuno di essi, la lunghezza totale, la quantità di merce trasportata e l'elenco delle visite ai clienti. Ogni visita è dettagliata con riferimento a:

- identificativo del cliente visitato
- tempo di arrivo del veicolo presso il cliente
- valori di apertura e chiusura delle finestre temporali del cliente (coincidenti nel caso in cui il cliente richieda una visita sincronizzata)
- valori originali di apertura e chiusura delle finestre temporali del cliente
- quantità di merce richiesta dal cliente

La figura 5.5 riporta il dettaglio di un percorso della soluzione al problema R107. Dopo avere specificato la quantità di merce trasportata lungo il percorso e la sua lunghezza, si riporta l'elenco dei clienti visitati lungo l'itinerario, secondo l'ordine di visita; il dettaglio di visita di ogni cliente è racchiuso tra parentesi quadre. Il primo valore in parentesi indica l'identificativo del cliente; il secondo, (preceduto dalla lettera B), indica il tempo di arrivo del veicolo presso il cliente; la prima coppia di

valori, (preceduta dalla sigla TW), indica rispettivamente i valori originali di apertura e di chiusura della finestra temporale del cliente; la successiva coppia di valori indica i reali istanti di apertura e di chiusura della finestra temporale; infine, il valore preceduto dalla lettera L indica la quantità di merce richiesta dal cliente. Si noti che il percorso in figura comprende due clienti che richiedono una visita sincronizzata: il cliente 40 e il cliente 100. Per questi clienti, si ha che il valore reale di finestra temporale è rappresentato da un istante nel tempo (75 e 165 rispettivamente); per tutti gli altri clienti, al contrario, si ha che il valore reale di finestra temporale e quello originale coincidono.

```
SOLUZIONE AL PROBLEMA r107



ROUTE N° 1

QUANTITà TRASPORTATA LUNGO IL PERCORSO: 109

LUNGHEZZA PERCORSO: 64,12

ELENCO NODI
[0 B:219.572322750397; TW:0,230; TW:0,230; L:0]
[40 B:75; TW:75,105; TW:75,75; L:9]
[2 B:94.4339811320566; TW:0,202; TW:0,202; L:7]
[87 B:111.505048943922; TW:83,113; TW:83,113; L:26]
[97 B:125.747689631041; TW:0,202; TW:0,202; L:12]
[100 B:165; TW:165,195; TW:165,165; L:17]
[98 B:178.162277660168; TW:0,198; TW:0,198; L:10]
[59 B:191.767828935632; TW:0,202; TW:0,202; L:28]
```

Figura 5.5 Dettaglio di un percorso di visita per il problema R107

5.4 Test sui problemi Solomon

L'algoritmo mateuristico realizzato è stato testato sull'insieme di problemi definiti da Solomon, opportunamente modificati per gestire i vincoli di sincronizzazione (come già introdotto nel Capitolo 4); questo set di problemi rappresenta un punto di riferimento per la valutazione delle performance degli algoritmi euristici di

risoluzione dei problemi di routing. I test svolti sui problemi Solomon hanno avuto diversi scopi di valutazione, permettendo di misurare le performance dell'algoritmo sotto diversi punti di vista, in particolar modo:

- controllando la qualità della soluzione ottenuta su ogni problema
- individuando, per ogni coppia di operatori destroy/repair, il grado di successo che questa ha avuto nel processo di identificazione di una nuova soluzione migliorante del problema

Poiché i VRPs caratterizzati da vincoli di sincronizzazione rappresentano una tipologia di problemi di analisi recente, il confronto e la valutazione dei risultati ottenuti hanno utilizzato come misura di confronto i risultati ottenuti sullo stesso set di problemi riportati in [3]. Poiché, inoltre, l'algoritmo realizzato presenta l'utilizzo di numerose componenti di diversificazione impiegate per ampliare la ricerca nello spazio delle soluzioni, (ad esempio, la scelta casuale del numero di vertici da rimuovere da una soluzione, l'utilizzo della componente di diversificazione che introduce rumore sui dati, etc.), per valutare i risultati ottenuti sui problemi Solomon, ogni sessione di test eseguita ha compreso 5 run di valutazione dei risultati raggiunti per ogni problema. Di seguito, in tabella 5.1, si riportano i dettagli delle soluzioni ottenute dall'algoritmo realizzato applicato su ogni problema Solomon. La prima colonna identifica il nome del problema risolto; la seconda e la terza colonna individuano i dettagli della soluzione determinata, sul problema corrispondente, dall'algoritmo realizzato in questo lavoro di tesi (rispettivamente numero di veicoli utilizzati per eseguire il trasporto delle merci e valore di distanza totale); la quarta e la quinta colonna indicano, rispettivamente per ogni problema, il numero di veicoli e il valore di distanza totale riportati in [3]; la sesta colonna indica la differenza tra il numero di veicoli della soluzione individuata dall'algoritmo e il numero di veicoli della soluzione indicata in [3]; la settima colonna individua lo scarto percentuale tra il valore di distanza totale della soluzione determinata dall'algoritmo e il valore di distanza totale della soluzione indicata in [3]; l'ottava colonna indica il tempo (misurato in secondi) che l'algoritmo impiega per risolvere il problema, calcolato

come media dei tempi ottenuti eseguendo 5 run dell'algoritmo su ogni problema. I test sono stati eseguiti su una macchina con le seguenti caratteristiche:

- Intel® Core™ 2 Duo CPU E6550 @ 2.33 GHz
- 4 GB RAM

	Numero Veicoli	Distanze Totali	Numero Veicoli Rif.	Distanze Tot. Rif.	Diff. Numero Veicoli	Scarto Distanze (%)	Tempo esec. medio (s)
C101	12	1231,33	14	1160,80	-2	6,08	440,93
C102	11	1258,36	14	1150,27	-3	9,40	485,06
C103	11	1211,66	14	1147,50	-3	5,59	819,88
C104	10	1169,65	11	1075,12	-1	8,79	3206,39
C105	12	1180,57	14	1160,80	-2	1,70	722,57
C106	12	1153,7	13	1139,73	-1	1,23	999,94
C107	12	1152,47	13	1146,78	-1	0,50	1468,79
C108	12	1123,7	13	1143,35	-1	-1,72	1869,94
C109	11	1131,58	-	-	-	-	2538,97
C201	4	939,15	6	878,84	-2	6,86	1296,89
C202	4	931,24	5	877,27	-1	6,15	4034,34
C203	4	926,08	6	873,28	-2	6,05	1802,89
C204	4	940,25	5	872,78	-1	7,73	2496,81
C205	4	912,79	6	877,54	-2	4,02	3155,39
C206	4	911,68	6	876,91	-2	3,96	4563,70
C207	4	911,37	5	878,03	-1	3,80	4724,76
C208	4	907,76	5	876,66	-1	3,55	5249,54
R101	21	1798,05	22	1791,74	-1	0,35	418,64
R102	19	1656,66	20	1612,63	-1	2,73	531,70
R103	15	1448,86	16	1368,82	-1	5,85	657,26
R104	11	1143,9	12	1115,01	-1	2,59	1588,63
R105	16	1566,97	18	1535,68	-2	2,04	639,48
R106	14	1396,67	15	1406,15	-1	-0,67	706,73

R107	12	1262,86	13	1223,76	-1	3,20	1243,34
R108	11	1106,83	11	1099,42	0	0,67	1648,93
R109	13	1311,73	15	1316,08	-2	-0,33	1037,54
R110	12	1361,21	14	1254,69	-2	8,49	1419,54
R111	12	1273,04	13	1202,80	-1	5,84	1397,85
R112	12	1151,73	12	1119,03	0	2,92	1928,23
R201	5	1440,04	9	1288,13	-4	11,79	1117,28
R202	4	1344,16	8	1172,57	-4	14,63	2009,09
R203	3	1443,32	7	1012,92	-4	42,49	2789,09
R204	3	968,46	5	885,06	-2	9,42	3180,01
R205	4	1232,69	7	1118,02	-3	10,26	2891,50
R206	3	1224,78	7	1041,57	-4	17,59	6416,37
R207	3	1108,39	6	955,32	-3	16,02	9663,22
R208	3	896,11	5	840,69	-2	6,59	11216,67
R209	3	1247,94	6	1002,92	-3	24,43	2949,48
R210	4	1147,38	7	1063,58	-3	7,88	2182,26
R211	3	1079,24	6	892,09	-3	20,98	2853,97
RC101	17	1823,36	18	1799,52	-1	1,32	530,55
RC102	15	1639,87	17	1604,64	-2	2,20	566,10
RC103	12	1442,23	13	1429,57	-1	0,89	998,21
RC104	11	1332,29	12	1319,00	-1	1,01	1255,59
RC105	15	1717,54	17	1683,68	-2	2,01	571,92
RC106	14	1548,8	15	1521,64	-1	1,78	664,64
RC107	13	1430,54	14	1442,04	-1	-0,80	1197,09
RC108	12	1341,93	13	1322,27	-1	1,49	1277,42
RC201	5	1686,33	10	1501,90	-5	12,28	1364,43
RC202	4	1608,45	8	1291,24	-4	24,57	2326,78
RC203	4	1328,42	6	1123,80	-2	18,21	2860,54
RC204	3	1207,8	5	1002,17	-2	20,52	3038,79
RC205	5	1560,36	8	1340,54	-3	16,40	1460,91
RC206	4	1443,62	8	1257,32	-4	14,82	3494,50
RC207	4	1364	7	1222,56	-3	11,57	2540,02

RC208	4	1068,49	6	998,63	-2	7,00	3114,43

Tabella 5.1 Risultati dei test eseguiti sui problemi Solomon

Come menzionato nel corso del Capitolo 3, lo scopo principale che l'algoritmo propone è quello di fornire una soluzione che riesca a minimizzare il numero di veicoli utilizzati per svolgere servizio presso i clienti; osservando i risultati ottenuti sul dataset dei problemi Solomon e comparandoli con i risultati riportati in [3], è possibile notare come, nella maggioranza dei casi, l'algoritmo realizzato nel lavoro di tesi abbia permesso di migliorare i risultati ottenuti in [3], minimizzando il numero di veicoli utilizzati per eseguire servizio presso i clienti. Solamente considerando le soluzioni di 2 problemi su 56 (R108 ed R112), si può notare che il numero di veicoli della soluzione finale calcolata secondo l'algoritmo realizzato risulta essere pari a quello della soluzione in [3]. Analizzando ulteriormente i risultati conseguiti, si può osservare come nel caso di 4 problemi (C108, R106, R109, RC107) l'algoritmo realizzato abbia individuato soluzioni migliori rispetto a quelle determinate in [3] non solamente in rapporto al numero di veicoli ma anche considerando il valore di distanza totale percorsa, consentendo, in maniera tale, di determinare una soluzione di migliore qualità rispetto a quella determinata in [3].

I test eseguiti sui problemi Solomon hanno avuto come ulteriore scopo quello di studiare il grado di successo che ogni coppia di operatori destroy/repair ha avuto nel processo di identificazione di una nuova soluzione migliorante durante l'esecuzione della metaeuristica. La tabella 5.2 riporta i risultati ottenuti. Ogni colonna della tabella individua un abbinamento di operatori destroy/repair; le abbreviazioni indicano rispettivamente:

- ra. random removal
- w. worst removal
- c. cluster removal
- re. related removal
- h. historical request-pair removal
- g. greedy heuristic
- s. sequential insertion heuristic

- t. regret-2 heuristic

Ogni riga di tabella riporta il numero di iterazioni della metaeuristica in cui l'applicazione di una particolare coppia di operatori destroy/repair ha permesso di individuare una nuova soluzione migliorante; i valori di ogni riga di tabella si riferiscono ad una singola run dell'algoritmo su tutti i problemi Solomon; in ultima riga, si riportano i totali per ogni coppia di operatori.

	ra.g.	ra.s.	ra.t.	w.g.	w.s.	w.t.	c.g.	c.s.	c.t.	re.g.	re.s.	re.t.	h.g.	h.s.	h.t.
Run 1	0	83	107	0	61	101	0	28	100	0	40	151	0	11	55
Run 2	1	100	112	0	76	87	2	21	76	0	27	162	0	15	56
Run 3	0	83	119	0	74	129	1	31	93	0	29	125	0	17	53
Run 4	0	76	128	0	79	91	0	20	81	0	23	153	0	23	55
Run 5	1	94	107	0	80	98	0	25	95	0	19	132	0	12	63
Totale	2	436	573	0	370	506	3	125	445	0	138	723	0	78	282

Tabella 5.2 Performance degli operatori destroy/repair

Analizzando i risultati riportati in tabella 5.2, è possibile rilevare un primo dato di interesse: si nota, infatti, che tutti gli abbinamenti dell'operatore di repair greedy heuristic con un qualsiasi operatore destroy risultano avere un'efficacia di individuazione di soluzioni miglioranti estremamente bassa. Ciò può essere giustificato se si considera il criterio che l'operatore greedy heuristic utilizza per inserire in soluzione i vertici rimossi: questo metodo, infatti, calcola il costo minimo di inserimento per ogni vertice da includere, inserendo iterativamente in soluzione il vertice per cui il costo di inserimento è minimo (dove il costo di inserimento viene calcolato in base all'incremento della lunghezza del tour in cui il vertice dovrà essere inserito). Il principio adottato da questo metodo di repair risulta, quindi, considerare una condizione di inserimento che rispetta un criterio di ottimalità locale, e che, in quanto tale, può portare ad operare sequenze di inserimenti di vertici che non permettono di individuare una nuova soluzione migliorante. Se si osserva quali coppie di operatori hanno permesso di individuare il maggior numero di soluzioni

miglioranti, è possibile ricavare un altro dato di interesse: l'applicazione dell'operatore di repair regret-2 heuristic in abbinamento con la maggior parte degli operatori destroy utilizzati è risultata avere un'alta efficacia nell'individuazione di soluzioni miglioranti. Le buone performance di questo metodo di repair della soluzione possono essere giustificabili se si considera il principio di inserimento adottato dal metodo: infatti, esso inserisce in soluzione il vertice per cui la differenza tra l'inserimento all'interno del percorso più vantaggioso nella posizione migliore e l'inserimento nel secondo percorso più vantaggioso nella posizione migliore è massima. Ciò significa che il metodo favorisce l'inserimento in soluzione dei vertici che risultano particolarmente difficili da includere (in particolare, favorisce l'inserimento in soluzione dei vertici che hanno una sola scelta possibile per l'inclusione). Questo elemento di look ahead nella costruzione della soluzione permette di operare scelte di inserimento più "lungimiranti", e quindi di aumentare la probabilità di individuare soluzioni miglioranti.

5.5 Test sull'istanza di problema reale

L'istanza del problema reale utilizzata è relativa ad un problema di assistenza tecnica: come menzionato in precedenza, il problema si compone di una squadra di tecnici, che deve fornire un servizio di assistenza presso il domicilio dei propri clienti. I clienti che compongono l'istanza del problema studiato sono 86, di cui 32 richiedenti una visita sincronizzata; essi sono dislocati principalmente nella zona della Romagna (province di Ravenna, Forlì-Cesena, Rimini); altri clienti risiedono invece in provincia di Arezzo e Pesaro-Urbino.

L'algoritmo realizzato è stato utilizzato per risolvere questa istanza di problema reale; in figura 5.6, viene visualizzata la soluzione che l'algoritmo ha individuato per questo problema. La soluzione prevede che il piano di visite presso i clienti venga svolto seguendo 23 percorsi di visita.

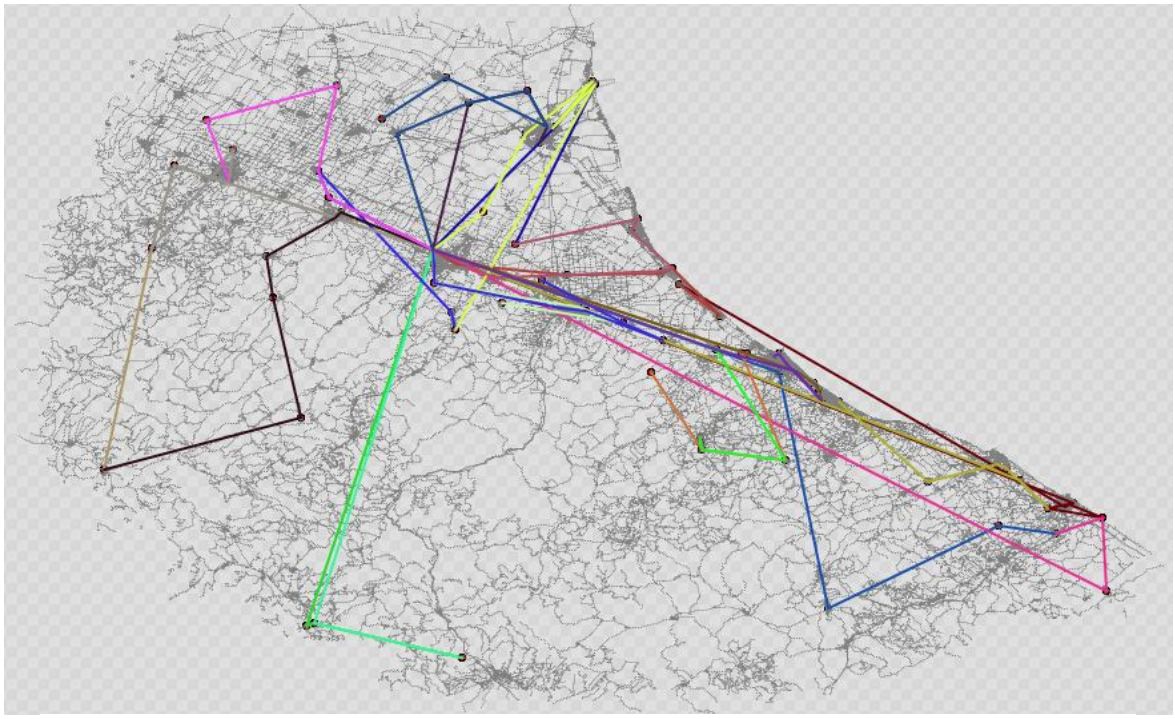

Figura 5.6 Soluzione dell'istanza di problema reale

Conclusioni

L’obiettivo del lavoro di tesi è stato quello di progettare e sviluppare un algoritmo per la pianificazione ottimizzata della distribuzione con viaggi sincronizzati; il metodo realizzato si occupa, dunque, di risolvere in maniera efficiente problemi di routing, caratterizzati dalla presenza di vincoli di sincronizzazione su un sottoinsieme di clienti.

La metodologia scelta per risolvere problemi di questa tipologia è stata quella degli algoritmi mateuristici. I metodi mateuristici sono algoritmi di ottimizzazione, basati sull’interazione tra metodi metaeuristici e tecniche di programmazione matematica. Lo scopo che i metodi mateuristici si prefiggono è quello di coniugare i pregi che caratterizzano gli algoritmi esatti e le metaeuristiche: grazie all’utilizzo di metodi esatti, è possibile trattare in maniera più specifica alcuni vincoli dei problemi studiati, mentre, grazie alle metaeuristiche, è possibile esplorare ampie zone dello spazio delle soluzioni dei problemi in un tempo accettabile. Il metodo esatto utilizzato nell’ambito del VRPTWPS permette di poter trattare in maniera specifica i vincoli di sincronizzazione, migliorando, qualora possibile, il posizionamento dei vertici sincronizzati all’interno dei percorsi in cui sono inseriti; grazie all’utilizzo di ALNS ed alla logica adattiva impiegata per sfruttare diversi operatori di vicinanza, è possibile esplorare diverse zone dello spazio delle soluzioni del problema.

Lo sviluppo del metodo di risoluzione del VRPTWPS ha avuto come obiettivo primario quello di ottenere soluzioni che minimizzassero il numero di veicoli utilizzati per eseguire i viaggi di distribuzione merce ai clienti. Come è possibile notare dal confronto con i risultati dei test eseguiti sui problemi Solomon e i risultati riportati in [3], si può affermare che tale obiettivo è stato raggiunto; l’algoritmo si è rivelato efficace nell’individuare, per 4 dei 56 problemi totali, soluzioni miglioranti anche dal punto di vista della distanza totale.

I vincoli di sincronizzazione che caratterizzano il VRPTWPS rendono questo problema difficile da risolvere utilizzando tradizionali metodi di ricerca locale. La

struttura dei percorsi di ogni soluzione è fortemente condizionata da questi vincoli, i quali impongono forti legami di dipendenza tra i percorsi che condividono vertici sincronizzati. L'alto numero di interconnessioni rende, dunque, il problema assai difficile da risolvere. Potere gestire in maniera efficace e allo stesso tempo efficiente i vincoli di sincronizzazione può permettere di ottenere soluzioni sempre più performanti. In questo lavoro di tesi, il vincolo di sincronizzazione è stato gestito dall'algoritmo tramite la riduzione della finestra temporale dei clienti che lo richiedono. Ridurre la finestra temporale di un vertice sincronizzato ad un istante nel tempo impone ai percorsi in cui è presente quel vertice un vincolo molto stringente, che può rendere potenzialmente più difficoltoso e a volte impedire l'inserimento di altri vertici al loro interno, ostacolando, di fatto, il processo di individuazione di soluzioni miglioranti. In questo lavoro di tesi, la gestione del vincolo di sincronizzazione ha avuto come scopo quello di ritardare il più possibile la riduzione delle finestre temporali, effettuando tale operazione al momento dell'inserimento in soluzione del secondo vertice sincronizzato.

Per evitare la riduzione delle finestre temporali durante il processo di individuazione di soluzioni miglioranti e permettere ugualmente che il vincolo di sincronizzazione sia rispettato per tutti i vertici che lo richiedono, si può prevedere la creazione di procedure efficienti, che controllino, per ogni inserimento di vertice in soluzione, che i vincoli di sincronizzazione siano rispettati su tutti i vertici interessati.

Bibliografia

[1] Olli Braysy, Michel Gendreau, “Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms”, *Transportation Science*, 2005, 39(1)

[2] Paolo Toth, Daniele Vigo, *The Vehicle Routing Problem*, SIAM, Philadelphia, 2002

[3] Sophie N. Parragh, Karl F. Doerner, “Hybridization strategies for routing problems with pairwise synchronization constraints”, Research Report

[4] David Bredstrom, Mikael Ronnqvist, “Combined vehicle routing and scheduling with temporal precedence and synchronization constraints”, *European Journal of Operational Research*, 2008, 191(1), pp. 19-31

[5] Louis-Martin Rousseau, Michel Gendreau, Gilles Pesant, “The Synchronized Vehicle Dispatching Problem”, in *Odysseus 2003*

[6] Yanzhi Li, Andrew Lim, Brian Rodrigues, “Manpower Allocation with Time Windows and Job-Teaming Constraints”, *Naval Reseach Logistics*, 2005, 52(4), pp. 302-311

[7] Michael Drexl, “Synchronization in Vehicle Routing – A Survey of VRPs with Multiple Synchronization Constraints”, *Transportation Science*, 2012

[8] Natallia Kokash, “An introduction to heuristic algorithms”, in *Research Methodology course*, Trento, June 2005

[9] Jens Lysgaard, “Clarke & Wright’s Savings Algorithm”, 1997

[10] S.Olafsson, “Metaheuristics”, in *Handbook on Simulation*, Nelson and Henderson (eds.), Elsevier, 2006, Handbooks in Operation Research and Management Science VII, 633-654

[11] David Pisinger, Stefan Ropke, “A general heuristic for vehicle routing problems”, *Computers & Operation Research*, 2007, 34(8), pp. 2403-2435

[12] Marco A. Boschetti, Vittorio Maniezzo, “Combining Exact Methods and Heuristics”, *Wiley Enciclopedia of Operations Research and Management Science*, 2011

[13] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, Thomas Stützle, “ParamILS: An Automatic Algorithm Configuration Framework”, *Journal of Artificial Intelligence Research (JAIR)*, 2009, 36, pp. 267-306

[14] M. M. Solomon, “Algorithms for the vehicle routing and scheduling problems with time window constraints”, *Operations Research*, 1987, 35, pp. 254-265

[15] Attila A. Kovacs, Sophie N. Parragh, Karl F. Doerner, “Adaptive large neighborhood search for service technician routing and scheduling problems”, *Journal of scheduling*, 2011

[16] Clarke, G. & Wright, J.W., “Scheduling of Vehicles from a Central Depot to a number of Delivery Points”, *Operations Research*, 1964, 12, pp.568-581

[17] F. Glover, “Ejection chains, reference structures and alternating path methods for Traveling Salesman Problems”, *Discrete Applied Mathematics*, 1996, 65, pp.223-253

[18] Or I., Traveling Salesman-Type Combinatorial Problems and their Relation to the Logistics of Regional Blood Banking, Ph.D. thesis, Northwestern University, Evanston, Illinois

[19] Taillard E., P. Badeau, M. Gendreau, F. Guertin and J.-Y. Potvin, “A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows”, *Transportation Science*, 1997, 31, pp.170-186

[20] Gendreau M., A. Hertz, G. Laporte, “A New Insertion and Postoptimization Procedure for the Traveling Salesman Problem”, *Operations Research*, 1992, 40, pp.1086-1093

[21] Kirkpatrick S., C. D. Gelatt Jr, M. P. Vecchi, “Optimization by Simulated Annealing”, *Science*, 1983, 220(5498), pp.671-680

[22] F. Glover, “Future Paths for Integer Programming and Links to Artificial Intelligence”, *Computers and Operations Research*, 1986, 13(5), pp.533-549

[23] P. Shaw, “Using Constraint Programming and Local Search Methods to solve Vehicle Routing Problems”, 1998, *Lecture Notes in Computer Science*, 1520, pp.417-431