Indice

Introduzione

Il *Vehicle Routing Problem* è un problema di ottimizzazione di notevole interesse pratico, con importanti applicazioni nell'ambito dei servizi di raccolta e distribuzione in genere.

Con questo lavoro si propone un nuovo algoritmo euristico, denominato *SERR*, per il *Capacitated Vehicle Routing Problem*, la variante più comune e più studiata del problema. Gli obiettivi conseguiti sono:

- la definizione di un nuovo *neighborhood* di cardinalità esponenziale e di un nuovo algoritmo basato su di esso;
- l'implementazione dell'algoritmo in linguaggio C++, sfruttando l'interfaccia ILOG Concert Technology dell'ottimizzatore ILOG Cplex;
- la sperimentazione dell'algoritmo su un insieme di istanze di diffuso impiego nella valutazione dei metodi per il VRP, con risultati che proiettano *SERR* fra i migliori euristici proposti in letteratura per questo problema;
- l'individuazione di nuove migliori soluzioni in assoluto per due dei pochissimi problemi di *benchmark* non ancora risolti all'ottimo.

Nel Capitolo 1 si illustra il *Vehicle Routing Problem* in tutti i suoi aspetti fondamentali, introducendone simbologia e nomenclatura e presentandone le varianti più comuni. Dopo aver introdotto i principali modelli di programmazione lineare intera, nel Capitolo 2 si presenta un'ampia rassegna di algoritmi euristici e metaeuristici, contenente i contributi più noti, più interessanti e più originali. Il Capitolo 3 è completamente dedicato alla presentazione e alla descrizione dell'algoritmo *SERR*, illustrato in tutti i suoi aspetti. La parte conclusiva del capitolo è riservata agli esperimenti computazionali, e contiene le illustrazioni e i dettagli di alcune delle soluzioni più interessanti ricavate grazie al nuovo algoritmo.

Ringraziamenti

Ringraziamenti particolari vanno al relatore, Prof. Matteo Fischetti, e al Prof. Paolo Toth dell'Università di Bologna per la loro supervisione; una dedica speciale va ad Alessandra per la pazienza, il supporto psicologico ed il prezioso aiuto concreto.

Capitolo 1

VRP e le sue varianti

Il problema noto come VRP — *Vehicle Routing Problem* — è stato proposto nel 1959 da Dantzig e Ramser [13]. In letteratura VRP è il nome generico con cui ci si riferisce ad un'intera classe di problemi inerenti alla visita di "clienti" da parte di "veicoli". Questo tipo di problemi ha notevoli implicazioni pratiche sia nel caso del trasporto effettivo di merce, sia in molti altri settori quali:

- raccolta di posta da cassette postali;
- servizio scuolabus;
- visite mediche a domicilio;
- visite di manutenzione preventiva;
- raccolta rifiuti.

Di seguito verrà descritta la versione più comune, CVRP — *Capacitated Vehicle Routing Problem* — e, successivamente, saranno prese in considerazione alcune tra le varianti più interessanti. Prima, tuttavia, è necessario introdurre gli attori coinvolti in questo genere di problemi.

1.1 Concetti generali

La distribuzione di merce riguarda il servizio di un insieme di clienti attuato mediante un flotta di *veicoli*, localizzati in uno o più *depositi* e affidati ad *autisti*, che si muovono su una *rete stradale*. La soluzione di un VRP consiste nella determinazione di un insieme di circuiti (*route*), ognuno percorso

da un singolo veicolo che parte e arriva ad un deposito (non necessariamente lo stesso), tali da soddisfare i requisiti di clienti e distributore e, contemporaneamente, da minimizzare il costo globale del trasporto.

La rete stradale è generalmente descritta tramite un grafo, che può essere orientato o meno; i suoi vertici corrispondono alle posizioni dei clienti e del deposito, mentre gli archi rappresentano i collegamenti stradali. Ogni arco è associato ad un *costo,* che di solito è pari alla lunghezza del collegamento stradale, ed eventualmente ad un tempo di percorrenza che può dipendere dal tipo di veicolo o dal periodo di tempo durante il quale l'arco è attraversato.

Ogni cliente è tipicamente associato a:

- quantità di merce (*domanda*), di uno o più tipi, che deve essere recapitata o raccolta;
- periodo del giorno (*time window*) durante il quale può o deve avvenire il servizio, per esempio legato a orari di apertura particolari o ad esigenze di disponibilità;
- tempo necessario per consegnare o raccogliere la merce, eventualmente in dipendenza dal tipo di veicolo;
- sottoinsieme dei veicoli che possono essere usati per servirlo, ristretti, ad esempio, a causa di problemi logistici o di accessibilità;
- eventuale *priorità* nel caso in cui non sia possibile servire tutti i clienti, ed eventuale *penale* associata alla parziale o totale mancanza di servizio.

I *route* percorsi hanno origine e terminano presso uno dei depositi, situati sui vertici del grafo. Ogni deposito è caratterizzato dal numero e dal tipo di veicoli associati ad esso e dall'ammontare di merce, di uno o più tipi, di cui dispone. In alcuni casi, i clienti sono già assegnati preventivamente ai depositi e i veicoli devono ritornare al deposito di partenza alla fine di ogni *route*: in questi casi, il problema può essere scomposto in sottoproblemi indipendenti, ognuno associato ad un singolo deposito.

Il trasporto delle merci è affidato ad una flotta di veicoli, la cui composizione e dimensione può essere prefissata o variare a seconda delle esigenze. Caratteristiche tipiche dei veicoli sono le seguenti:

- deposito di partenza, al quale il veicolo può essere obbligato o meno a ritornare al termine del *route*;
- *capacità* del veicolo espressa in volume, peso o numero di colli trasportabili;
- eventuale suddivisione in *scompartimenti*, ognuno caratterizzato dalla sua capacità e dal tipo di merce che può contenere — si pensi, per esempio, alla presenza di celle frigorifere assieme a vani non refrigerati;
- sottoinsieme dei collegamenti della rete stradale attraversabili dal veicolo;
- *costo* associato all'utilizzo del veicolo, fisso, per unità di distanza e/o per unità di tempo.

I veicoli sono condotti da autisti; possono essere presenti vincoli sulle modalità di lavoro riguardanti orari, numero e durata delle pause durante il servizio, straordinari, ecc. Comunemente, questi vincoli vengono associati direttamente ai veicoli.

Ogni *route* deve soddisfare determinati vincoli, che dipendono dalla natura della merce trasportata, dal livello di qualità del servizio e dalle caratteristiche di clienti e veicoli. Alcuni tipici vincoli sono i seguenti:

- la richiesta totale dei clienti posti lungo il *route* non può superare la capacità del veicolo ad esso assegnato;
- i clienti serviti possono richiedere solo la consegna di merce, solo la raccolta o ambedue le cose;
- i clienti possono essere serviti solo nei loro specifici intervalli temporali (*time windows*) e durante i periodi di lavoro degli autisti;
- devono essere rispettati eventuali vincoli di precedenza definiti tra i clienti — si pensi, ad esempio, al caso in cui parte della merce da consegnare ad un cliente debba essere prima ritirata da altri (*pickup and delivery problem*); in questo caso, inoltre, interi gruppi di clienti devono essere serviti dallo stesso veicolo. Un'altra situazione di questo tipo si ha nel cosiddetto VRP con *Backhauls*, in cui i veicoli possono effettuare raccolta e distribuzione, a condizione che quest'ultima attività avvenga per prima.

Diversi obiettivi, spesso contrastanti, possono essere considerati per i problemi di *vehicle routing*. Tipici obiettivi sono:

- minimizzazione del costo globale di trasporto, dipendente dalla distanza totale percorsa, dal tempo totale impiegato e dai costi fissi associati ai veicoli e agli autisti;
- minimizzazione del numero dei veicoli o autisti necessari per servire tutti i clienti;
- bilanciamento dei *route* in termini di tempo di percorrenza e/o carico dei veicoli;
- minimizzazione delle penali associate al servizio solo parziale dei clienti.

Talvolta viene richiesto di minimizzare una funzione di costo che corrisponde ad una media pesata di due o più delle precedenti.

In alcune applicazioni particolari i veicoli possono percorrere più di un *route* oppure i *route* possono richiedere tempi di percorrenza superiori ad un giorno. Inoltre, è talvolta necessario considerare versioni stocastiche o tempodipendenti del problema: ad esempio, è possibile che ci sia una conoscenza solo parziale o incompleta dei clienti, dei costi o dei tempi di percorrenza.

1.2 Descrizione dei problemi

Dopo questa breve introduzione, si presenterà una definizione formale, sotto forma di modello su grafo, dei problemi di base della classe VRP. Per ognuno di questi problemi, diverse varianti minori sono state proposte ed esaminate in letteratura e, talvolta, sono stati attribuiti gli stessi nomi a problemi diversi. Per questo, ogni problema verrà dapprima descritto nella sua forma base, a cui ci si riferirà mediante il corrispondente acronimo, e successivamente se ne considereranno le eventuali varianti.

In questa sezione, inoltre, si introdurranno una notazione e una terminologia di base.

1.2.1 CVRP — VRP con vincoli di capacità

Il CVRP — *Capacitated Vehicle Routing Problem* — è la versione più comune di questa famiglia di problemi e le sue peculiarità sono le seguenti:

il servizio è di semplice consegna senza raccolta; le richieste dei clienti sono note a priori e deterministiche e devono essere soddisfatte da un solo veicolo; i veicoli sono identici e basati su un singolo deposito centrale; gli unici vincoli imposti riguardano le loro capacità. L'obiettivo è minimizzare il costo totale di servizio, che può essere una funzione del numero di *route*, della loro lunghezza o del tempo di viaggio.

Il CVRP può essere descritto in termini di problema su grafo, nel seguente modo. Sia $G = (V, A)$ un grafo completo, dove $V = \{0, \ldots, n\}$ è l'insieme dei vertici e A quello degli archi. I vertici $i = 1, \ldots, n$ corrispondono ai clienti, mentre il vertice 0 corrisponde al deposito. Ad ogni arco $(i, j) \in A$ è associato un costo non negativo c_{ij}, che rappresenta il costo di trasferimento dal vertice i al vertice j; in genere, l'uso di *loop* (i, i) non è consentito e ciò è imposto definendo $c_{ii} = +\infty$ per tutti gli $i \in V$. Se G è un grafo diretto, la matrice dei costi c è asimmetrica e il corrispondente problema è detto ACVRP (*Asymmetric CVRP*); altrimenti, si ha $c_{ij} = c_{ji} \ \forall (i, j) \in A$ e il problema è chiamato SCVRP (*Symmetric CVRP*). In quest'ultimo caso, l'insieme degli archi A è generalmente sostituito da un insieme di lati E. Dato un lato $e \in E$, $\alpha(e)$ e $\beta(e)$ denotano i suoi estremi. Nel seguito si indicherà l'insieme dei lati di un grafo non diretto G con A quando i lati sono indicati a mezzo dei loro estremi (i, j), e con E quando essi sono indicati con un singolo simbolo e.

Il grafo G deve essere fortemente connesso ed è generalmente completo. Dato un vertice i, si indica con $\Delta^+(i)$ il *forward star*, ossia l'insieme di tutti i vertici j tali che $(i, j) \in A$; analogamente, $\Delta^-(i)$ indica il *backward star*, che corrisponde all'insieme dei vertici l tali che $(l, i) \in A$. Dato un insieme di vertici $S \subseteq V$, $\delta(S)$ denota l'insieme dei lati $e \in E$ con un solo estremo in S mentre $E(S)$ indica il sottoinsieme dei lati con entrambi gli estremi in S. Con un leggero abuso di linguaggio, quando si considera un solo vertice si utilizza la forma $\delta(i)$ in luogo di $\delta(\{i\})$.

In molti casi di interesse pratico la matrice dei costi soddisfa la *disuguaglianza triangolare*:

$$c_{ik} + c_{kj} \geq c_{ij} \quad \forall i, j, k \in V.$$

Questa proprietà viene talvolta richiesta esplicitamente da alcuni algoritmi; quando non è presente, si può ovviare alla sua mancanza aggiungendo una quantità positiva M al costo di ognuno degli archi. Questa procedura, tuttavia, può creare problemi nella misura in cui introduce una pesante

distorsione della metrica di costo delle soluzioni.

In alcune istanze, i vertici sono associati a punti del piano di cui sono specificate le coordinate; in tali casi il generico costo c_{ij} relativo all'arco $(i, j) \in A$ è definito come la distanza euclidea tra i due punti corrispondenti ai vertici i e j. La matrice dei costi è allora simmetrica, e soddisfa anche la disuguaglianza triangolare; il problema risultante è denominato *Euclidean SCVRP*. È bene tuttavia precisare che esistono diverse convenzioni sull'arrotondamento dei costi: non è infrequente trovare istanze in cui sia richiesto di arrotondare ogni costo all'intero più vicino o all'intero immediatamente superiore. Nel primo caso, la proprietà triangolare può venire meno a seguito dell'arrotondamento.

Ogni cliente $i = 1, \ldots, n$ è associato ad una richiesta non negativa di merce d_i, mentre il deposito ha una domanda fittizia $d_0 = 0$. Dato un insieme $S \subseteq V$, $d(S)$ denota la richiesta complessiva dei clienti in S: $d(S) = \sum_{s \in S} d_s$. Indicato con r un *route*, $d(r)$ denota la richiesta complessiva dei vertici da esso visitati.

Un insieme di K veicoli, identici tra loro ed ognuno con capacità C, è disponibile nel deposito; una semplice condizione di ammissibilità del problema richiede che $d_i \leq C$ per ogni cliente $i = 1, \ldots, n$. Ogni veicolo può percorrere al più un *route*, e si assume che K sia non minore di K_{min}, pari al minimo numero di veicoli necessari per servire tutti i clienti. Il valore di K_{min} può essere determinato risolvendo il *Bin Packing Problem* associato all'istanza di CVRP: esso richiede di determinare il minimo numero di contenitori, ognuno dei quali con capacità C, necessari per caricare un insieme di n oggetti, ognuno con un peso d_i per $i = 1, \ldots, n$. Il BPP è un problema NP-difficile, ma esistono algoritmi risolutivi molto efficienti anche per istanze con centinaia di elementi.

Dato un insieme di clienti $S \subseteq V$, si indica con $r(S)$ il numero minimo di veicoli necessari a servire tutti i vertici in S; vale $r(V \backslash \{0\}) = K_{min}$. Una grossolana stima di $r(S)$ può essere fornita dal *lower bound* $\lceil d(S)/C \rceil$ per il *Bin Packing Problem*.

Il CVRP richiede la determinazione di un insieme di esattamente K circuiti semplici, ognuno corrispondente al percorso di un veicolo, in modo che il costo totale di trasporto, definito come somma dei costi degli archi che appartengono ai circuiti, sia minimo. I vincoli del problema sono i seguenti:

- ogni veicolo, e dunque ogni circuito, deve transitare per il deposito;

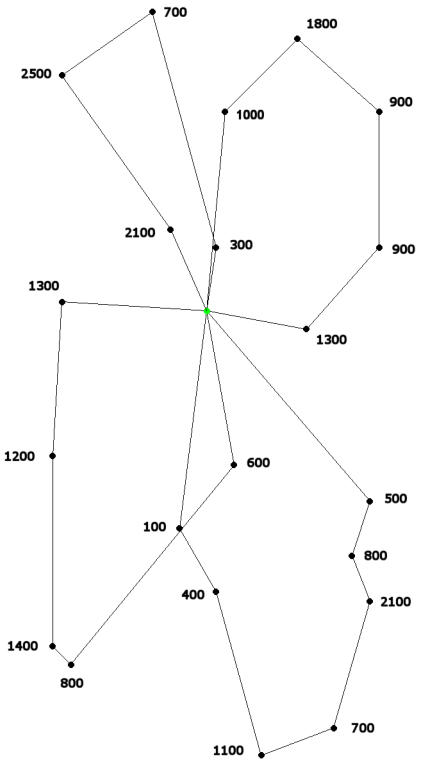


Figura 1.1: Una soluzione dell'istanza eil22, con 22 clienti e capacità dei veicoli pari a 6000.

- ogni cliente è visitato da uno ed un solo circuito;
- la somma delle richieste di merce dei vertici visitati da ogni circuito non può eccedere la capacità C dei veicoli.

In figura 1.1 è rappresentata una soluzione di un'istanza di CVRP con 22 vertici e capacità dei veicoli pari a 6000 (l'istanza è nota in letteratura come eil22).

La letteratura propone molte varianti di questa semplice versione di CVRP. In una di esse, quando il numero K di veicoli disponibili è superiore a K_{min}, è consentito lasciare inattivi alcuni di essi. In questo caso, ad ogni mezzo viene spesso associato un costo fisso, e il costo della soluzione viene valutato anche in base a questo aspetto. Un'altra variante piuttosto frequente considera la presenza di veicoli diversi tra loro, ognuno con una propria capacità C_k con $k = 1, \ldots, K$ ed, eventualmente, con un proprio costo fisso. In altri casi possono invece essere vietati *route* che visitino un solo cliente.

Il CVRP è un problema NP-difficile che generalizza il ben noto *Traveling Salesman Problem* (*TSP*), che richiede di determinare un circuito semplice di costo minimo che visiti tutti i vertici di G. Un'istanza di CVRP con parametri $C \geq d(V)$ e $K = 1$ si riduce proprio proprio ad un'istanza di TSP.

1.2.2 VRP con vincoli di lunghezza dei *route*

Una variante del CVRP è il DVRP — *Distance-Constrained VRP*. In questo problema i vincoli di capacità riguardanti ognuno dei *route* sono sostituiti da vincoli di lunghezza o di tempo massimi; in particolare, una *lunghezza* non negativa t_{ij} viene associata a ciascun arco o lato (i, j), e la lunghezza totale degli archi di ogni *route* non può superare un valore massimo, pari a T. I veicoli possono avere lunghezze massime di tragitto diverse o possono disporre di tempi diversi per completare il proprio percorso. Quando i parametri t_{ij} rappresentano tempi di viaggio, ad ogni vertice i può essere assegnato un *tempo di servizio* s_i, pari al tempo necessario ad un veicolo per compiere il proprio servizio presso il cliente. In alcuni casi i tempi di servizio possono essere inclusi nei costi temporali dei lati, ponendo per ogni arco (i, j) $t_{ij} = t'_{ij} + s_i/2 + s_j/2$, dove t'_{ij} è il costo temporale per la sola percorrenza dell'arco (i, j).

In una seconda variante, il DCVRP — *Distance-Constrained CVRP* — sono imposte entrambe le famiglie di vincoli: ogni *route* ha una distanza od un tempo di percorrenza massimo, e nel contempo valgono le limitazioni di capacità dei veicoli.

In genere le matrici dei costi e delle distanze coincidono: vale cioè l'uguaglianza $c_{ij} = t_{ij}$ per tutti gli archi $(i, j) \in A$. L'obiettivo del problema corrisponde allora a minimizzare la lunghezza totale dei route oppure, se il tempo di servizio è incluso nei costi temporali degli archi, la loro durata.

1.2.3 VRP con *Time Window*

Il VRP con *Time Window* è un'altra estensione del CVRP in cui ad ogni cliente i è associato un intervallo di tempo $[a_i, b_i]$ detto, appunto, *time window*. Il servizio di ogni cliente deve iniziare in un istante t_i contenuto nel *time window*; in caso di arrivo anticipato al vertice i, il veicolo deve attendere l'istante a_i prima di poter effettuare il servizio. Ognuno dei clienti è anche associato ad un tempo di servizio, s_i, che rappresenta la durata dell'intervallo di tempo durante il quale il veicolo che effettua il servizio rimane

fermo presso il cliente stesso. Altri dati del problema sono la matrice dei tempi di viaggio, il cui elemento generico t_{ij} è pari al tempo di percorrenza dell'arco $(i, j) \in A$, e t_0, l'istante di tempo nel quale i veicoli lasciano il deposito.

Di norma la matrice dei costi e la matrice dei tempi di transito coincidono, e i *time window* sono definiti assumendo che tutti i veicoli partano dal deposito all'istante $t_0 = 0$. VRPTW è di norma rappresentato come problema asimmetrico, in quanto i valori dei *time window* inducono implicitamente un orientamento dei *route*.

Riassumendo, la risoluzione di un VRPTW si attua determinando un insieme di K circuiti semplici di costo minimo e tali che:

- ogni circuito visiti il deposito;
- ogni cliente sia visitato da esattamente un circuito;
- la somma delle richieste dei vertici visitati da un circuito non ecceda la capacità C del veicolo;
- per ogni cliente i, il servizio abbia inizio in un istante compreso nel *time window* $[a_i, b_i]$ e il veicolo rimanga occupato per un tempo pari ad s_i.

Un'istanza caratterizzata dai parametri $a_i = 0$ e $b_i = +\infty$ per ogni $i = 1, \ldots n$ si riduce di fatto ad un'istanza di CVRP, problema quindi generalizzato da VRPTW che risulta NP-difficile in senso stretto.

1.2.4 VRP con *Pickup and Delivery*

Nella versione di base del VRP con *Pickup* e *Delivery* (VRPPD), ogni cliente i è associato a due quantità non negative d_i e p_i, rappresentanti la richiesta di merce e la quantità della stessa da ritirare, rispettivamente. Talvolta, un unico parametro pari a $d_i - p_i$, la differenza netta, eventualmente negativa, tra merce da consegnare e merce da ritirare, viene assegnato per ogni cliente i. Per ogni vertice i sono presenti altri due parametri, O_i e D_i, che caratterizzano questa variante del problema. La merce richiesta dal vertice i deve essere preventivamente raccolta dal veicolo presso il cliente O_i; allo stesso modo, la merce ritirata presso il cliente i è destinata al cliente D_i, che deve quindi essere visitato successivamente. In corrispondenza alla sede di ciascun cliente, si assume sempre che lo scarico della merce avvenga prima del caricamento.

Il VRPPD richiede di determinare esattamente K *route* di costo minimo e tali che:

- ogni circuito visiti il deposito;
- ogni cliente sia visitato da esattamente un circuito;
- il carico dei veicoli, in qualsiasi punto dei circuiti, sia sempre non negativo e non superi la capacità totale C;
- per ogni cliente i il vertice O_i, se diverso dal deposito, venga visitato nello stesso circuito e prima di i;
- per ogni cliente i il cliente D_i, se diverso dal deposito, venga visitato nello stesso circuito e dopo la visita di i.

Spesso l'origine O_i o la destinazione D_i della merce sono comuni per tutti i vertici — per esempio, possono coincidere con il deposito — e non sono quindi indicati esplicitamente. In questi casi, il problema è noto come VRP con *Pickup* e *Delivery* simultaneo (VRPSPD). Un'altra variante nota in letteratura è il presenta contemporaneamente *Pickup*, *Delivery* e *Time Window* (VRPPDTW).

VRPPD e VRPSPD generalizzano entrambi CVRP, e sono quindi NP-difficili in senso stretto.

1.2.5 VRP con *Backhaul*

Il VRP con *Backhaul* (VRPB) è un'estensione del CVRP in cui l'insieme dei clienti $V\backslash\{0\}$ è partizionato in due sottoinsiemi. Il primo, L, contiene n clienti *Linehaul*, ognuno richiedente la consegna di una certa quantità di prodotto; il secondo insieme, B, contiene m clienti *Backhaul*, dai quali una data quantità di prodotto deve essere prelevata. I vertici sono numerati in modo che $L = \{1, \ldots, n\}$ e $B = \{n+1, \ldots, n+m\}$.

Nel VRPB è sottointeso un vincolo di precedenza tra i clienti in L e i clienti in B: se un *route* serve clienti di entrambi i tipi, tutti i vertici in L devono essere serviti prima di ciascuno di quelli in B. Ad ogni vertice i è associato un parametro non negativo d_i, rappresentante la domanda o la richiesta di merce a seconda della tipologia del vertice stesso; al deposito è assegnato un valore fittizio $d_0 = 0$. Quando la matrice dei costi è asimmetrica, il problema è detto VRP cono *Backhaul* asimmetrico (AVRPB). In

letteratura è stato studiato anche il VRPBTW, una versione di VRPB con *Time Window*.

VRPB e AVRPB richiedono di individuare un insieme di K circuiti semplici di costo minimo tali che:

- ogni circuito visiti il deposito;
- ogni vertice sia visitato da uno ed un solo circuito;
- le richieste totali dei clienti in L e di quelli in B non superino, separatamente, la capacità C dei veicoli;
- in ogni circuito, tutti i clienti *linehaul* vengano visitati prima di eventuali clienti *backhaul*.

Circuiti contenenti solo clienti *backhaul* non sono in genere ammessi.

Denotati con K_L e K_B il minimo numero di veicoli necessari a servire tutti i clienti in L e tutti quelli in B, rispettivamente, si deve assumere per l'ammissibilità di un'istanza che $K \geq max\{K_L, K_B\}$. K_L e K_B possono essere ricavati risolvendo l'istanza di *Bin Packing Problem* associata ai rispettivi sottoinsiemi di vertici.

VRPB e AVRPB generalizzano rispettivamente SCVRP e ACVRP quando $B = \varnothing$, e sono quindi problemi NP-difficili in senso stretto.

Capitolo 2

Algoritmi euristici e metaeuristici

In questo capitolo verranno presentati alcuni dei principali e più interessanti algoritmi euristici e metaeuristici proposti per il *Vehicle Routing Problem*. Nella prima parte saranno inoltre illustrati alcuni modelli matematici di interesse per la risoluzione esatta tramite algoritmi *Branch-and-Bound* e *Branch-and-Cut*; per una trattazione specifica di questi metodi si vedano i contributi di Toth e Vigo [59] e Naddef e Rinaldi[46].

In tutti i casi ci si riferirà a formulazioni adatte per la variante classica del CVRP; spesso, comunque, i metodi si prestano ad adattamenti che consentono di affrontare altre varianti del problema.

2.1 Modelli matematici

I modelli matematici di programmazione lineare proposti in letteratura si possono catalogare in tre categorie:

- formulazioni di tipo *vehicle flow*;
- formulazioni di tipo *commodity flow*;
- formulazioni di tipo *set-partitioning*.

2.1.1 Modelli *vehicle flow*

Questi modelli fanno uso di variabili intere, associate ai lati del grafo, che esprimono il numero di volte in cui ogni singolo lato viene attraversato

da un veicolo. La maggioranza dei modelli proposti per le varianti più semplici del VRP appartiene a questa categoria, che non contiene invece modelli adatti ad affrontare versioni più interessanti. In questi modelli il rilassamento continuo tende ad essere piuttosto lasco in presenza di vincoli rigidi.

A questa famiglia appartiene il cosiddetto *modello a due indici*, adatto anche per istanze di tipo asimmetrico. Indicato con $G = (V, A)$ il grafo orientato che descrive il problema, sono impiegate $O(n^2)$ variabili binarie con il seguente significato:

$$x_{ij} = \begin{cases} 1 & \text{sse } (i,j) \in A \text{ appartiene alla soluzione ottima} \\ 0 & \text{altrimenti} \end{cases}$$

Il modello è il seguente:

$$\min \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij} \tag{2.1}$$

con i vincoli:

$$\sum_{i \in V} x_{ij} = 1 \quad \forall j \in V \backslash \{0\} \tag{2.2}$$

$$\sum_{j \in V} x_{ij} = 1 \quad \forall i \in V \backslash \{0\} \tag{2.3}$$

$$\sum_{i \in V} x_{i0} = K \tag{2.4}$$

$$\sum_{i \in V} x_{0j} = K \tag{2.5}$$

$$\sum_{i \notin S} \sum_{j \in S} x_{ij} \geq r(S) \quad \forall S \subseteq V \backslash \{0\},\ S \neq \emptyset \tag{2.6}$$

$$x_{ij} \in \{0, 1\} \quad \forall i, j \in V \tag{2.7}$$

I vincoli (2.2) e (2.3) impongono che esattamente un arco entri ed esca da ogni vertice; (2.4) e (2.5) riguardano il numero di archi selezionati entranti ed uscenti dal deposito, che devono essere pari a K. I vincoli (2.6), infine, impongono che, nella soluzione, ogni taglio $(V \backslash S, S)$ indotto da un insieme S di vertici sia attraversato da un numero di archi almeno pari ad $r(S)$,

valore questo che rappresenta il numero minimo di veicoli necessari per servire i vertici dell'insieme S.

Questo modello a due indici, come tutte le sue possibili varianti, ha il limite di non essere applicabile a problemi in cui il costo (o l'ammissibilità della soluzione) dipende dalla sequenza di visita dei vertici o dal tipo di veicolo che percorre ogni *route*. La risoluzione del modello, infatti, non dà informazione riguardo a quali veicoli debbano transitare sugli archi prescelti.

Una possibile soluzione è quella di considerare esplicitamente i veicoli che attraversano gli archi: si ottiene così la cosiddetta *formulazione a tre indici*, che usa $O(n^2K)$ variabili binarie, ognuna associata ad un arco e ad un veicolo; in aggiunta, vi sono altre $O(nK)$ variabili binarie, associate ai vertici ed ai veicoli, che specificano da quali veicoli debbano essere serviti i clienti. I modelli a tre indici si adattano anche a problemi più vincolati rispetto al CVRP, come per esempio il VRPTW; il prezzo pagato è l'incremento del numero di variabili.

2.1.2 Modelli *commodity flow*

Questi modelli sono caratterizzati dalla presenza di variabili continue, associate agli archi del grafo, che rappresentano l'ammontare di merce trasportata dai veicoli che percorrono gli archi stessi. Queste formulazioni considerano esclusivamente, senza alcuna perdita di generalità, grafi orientati. Un approccio esatto a SCVRP che fa uso di un modello di questo tipo, proposto da Baldacci, Mingozzi e Hadjiconstantinou [3], richiede preliminarmente di generare un grafo $G' = (V', A')$ aggiungendo a $G = (V, A)$ il nodo $n + 1$, corrispondente ad una copia del deposito; ogni *route* viene allora considerato come un cammino dal nodo 0 al nodo $n + 1$. Ad ogni arco (i, j) di A' sono associate due variabili di flusso, non negative, y_{ij} e y_{ji}, il cui significato è il seguente: se un veicolo passa dal vertice i al vertice j, y_{ij} indica al carico del veicolo mentre y_{ji} la capacità residua. Alla luce di quanto detto, per ogni arco (i, j) vale l'uguaglianza $y_{ij} + y_{ji} = C$. Per ogni possibile *route* le variabili di flusso definiscono due cammini orientati: uno va dal vertice 0 al vertice $n + 1$, per cui le variabili rappresentano il carico del veicolo, e l'altro nel verso contrario da $n + 1$ a 0, con le variabili che rappresentano la capacità residua. Poste queste premesse il modello, che fa uso delle variabili decisionali x_{ij} con il solito significato, è il seguente:

$$\min \sum_{(i,j)\in A'} c_{ij}x_{ij} \tag{2.8}$$

con i vincoli:

$$\sum_{j\in V'} (y_{ji} - y_{ij}) = 2d_i \quad \forall i \in V'\backslash\{0, n+1\} \tag{2.9}$$

$$\sum_{j\in V'\backslash\{0,n+1\}} y_{0j} = d(V\backslash\{0, n+1\}) \tag{2.10}$$

$$\sum_{j\in V'\backslash\{0,n+1\}} y_{j0} = KC - d(V\backslash\{0, n+1\}) \tag{2.11}$$

$$\sum_{j\in V'\backslash\{0,n+1\}} y_{n+1\,j} = KC \tag{2.12}$$

$$y_{ij} + y_{ji} = Cx_{ij} \quad \forall (i,j) \in A' \tag{2.13}$$

$$\sum_{j\in V'} (x_{ij} + x_{ji}) = 2 \quad \forall i \in V'\backslash\{0, n+1\} \tag{2.14}$$

$$y_{ij} \geq 0 \quad \forall (i,j) \in A' \tag{2.15}$$

$$x_{ij} \in \{0, 1\} \quad \forall (i,j) \in A' \tag{2.16}$$

I vincoli (2.9) impongono che la somma delle differenze delle variabili di *commodity flow* associate agli archi entranti e uscenti da ogni vertice i sia uguale al doppio della richiesta di i; i vincoli (2.10)–(2.12) determinano il valore corretto delle variabili di flusso di merce per archi adiacenti ai vertici-deposito 0 ed $n+1$; i vincoli (2.13) e (2.14) impongono la relazione tra il grado dei vertici e, rispettivamente, le variabili di flusso di merce e di flusso di veicoli.

2.1.3 Modelli *set-partitioning*

Proposta per la prima volta nel 1964 da Balinski e Quandt [4], la formulazione *set-partitioning* del VRP impiega un'ampia collezione di *route* ammissibili, ognuno associato ad una variabile decisionale binaria. Si considera un insieme $\mathcal{H} = \{H_1, \ldots, H_q\}$ di circuiti ammissibili, l'i-esimo avente

costo c_i, ed una serie di coefficienti binari a_{ij}, pari ad 1 se e solo se il vertice i è visitato dal *route* H_j. La generica variabile decisionale binaria x_j, con $j = 1, \ldots, q$, è pari a 1 se e solo se il circuito H_j compare nella soluzione ottima. Il semplice modello è il seguente:

$$\min \sum_{j=1}^{q} c_j x_j \tag{2.17}$$

con i vincoli:

$$\sum_{j=1}^{q} a_{ij} x_{ij} = 1 \quad \forall i \in V \backslash \{0\} \tag{2.18}$$

$$\sum_{j=1}^{q} x_j = K \tag{2.19}$$

$$x_j \in \{0, 1\} \quad \forall j = 1, \ldots, q \tag{2.20}$$

I vincoli (2.17) impongono che ogni vertice sia coperto da uno ed uno solo dei *route* scelti, mentre il vincolo (2.19) determina il numero di circuiti che formano la soluzione.

Quando la matrice dei costi soddisfa la disuguaglianza triangolare[1] i vincoli (2.18) possono essere sostituiti dai seguenti:

$$\sum_{j=1}^{q} a_{ij} x_{ij} \geq 1 \quad \forall i \in V \backslash \{0\} \tag{2.21}$$

e il modello diventa del tipo *set-covering*, che risulta essere più vantaggioso in quanto consente di limitare le variabili decisionali rispetto al *set-partitioning*.

Questo tipo di modello è molto generale e si presta ad essere adattato a molte varianti del VRP: i vincoli non contengono in sé i requisiti di ammissibilità dei circuiti, i quali vengono generati a parte. Lo svantaggio risiede nell'elevatissimo numero di variabili decisionali che si rendono necessarie; per questo motivo, spesso nell'implementazione di questi metodi si adotta un approccio a generazione di colonne. Alcune possibilità verranno illustrate nella sezione 2.2.2, in cui si accennerà ad un algoritmo euristico basato su un modello *set-partitioning*.

[1] Ciò accade quasi sempre nel caso di istanze di tipo geografico, anche se talvolta l'adozione di talune convenzioni di arrotondamento dei costi può far perdere questa proprietà.

2.2 Algoritmi euristici classici

Lo scopo degli algoritmi euristici è quello di fornire una soluzione di buona qualità con poco tempo di calcolo. Nella maggior parte dei casi, infatti, non si dispone del tempo necessario per applicare metodi esatti basati su modelli matematici come, ad esempio, quelli esposti nella sezione 2.1. In tali casi ci si accontenta di soluzioni di buona qualità, purchè queste siano ottenute in tempi accettabili.

Secondo la classificazione di Laporte e Semet [41], è opportuno distinguere tra euristici classici e metaeuristici: mentre i primi attuano una esplorazione relativamente limitata dello spazio delle soluzioni e producono solitamente buone soluzioni in poco tempo di calcolo, i metaeuristici operano una ricerca approfondita nelle regioni più promettenti dello spazio delle soluzioni. Questi metodi implementano sofisticate regole di ricerca e di ricombinazione di soluzioni, producendo soluzioni di migliore qualità rispetto agli euristici classici richiedendo però un tempo di calcolo solitamente più elevato. Rinviando la trattazione di questo tipo di algoritmi alla sezione 2.3, si focalizzerà qui l'attenzione sugli euristici di tipo classico.

A grandi linee è possibile classificare gli euristici classici proposti per VRP in tre famiglie:

- euristici costruttivi;
- euristici a due fasi;
- euristici migliorativi.

I metodi della prima categoria operano costruendo gradualmente una soluzione ammissibile, cercando di contenere il costo durante il procedimento. I metodi a due fasi, invece, scompongono il problema nelle operazioni di suddivisione dei vertici in gruppi (*cluster*) e di costruzione di *route* ammissibili. Questi metodi sono suddivisibili in due classi: *cluster-first, route-second* e *route-first, cluster-second*. Nel primo caso i vertici sono inizialmente raggruppati in *cluster* e in un secondo momento viene costruito un *route* per ogni *cluster*; nel secondo, invece, un *route* viene costruito su tutti i vertici per poi essere suddiviso. I metodi migliorativi, infine, si applicano ad una soluzione preesistente (in alcuni casi anche non ammissibile) con l'intento di migliorarla e operano tipicamente mediante scambi di archi o vertici tra diversi *route*. Va detto che molti algoritmi costruttivi implementano an-

che fasi migliorative, e questo rende la classificazione complessa e, in parte, arbitraria.

In questa sezione si presenterà una breve introduzione ad alcuni dei più conosciuti o più interessanti algoritmi euristici per il VRP. In tutti i casi si tratta di metodi che si applicano direttamente alla variante CVRP; normalmente gli algoritmi sono adattabili anche ad altre varianti più vincolate, come per esempio DCVRP. Molti di questi metodi non richiedono in ingresso il numero K di veicoli da coinvolgere, anche se sono presenti eccezioni.

2.2.1 Metodi costruttivi

Gli algoritmi costruttivi utilizzano principalmente due tecniche: una consiste nell'inserire gradualmente i vertici nei *route* in base ai costi di inserzione, mentre l'altra si attua fondendo più *route* in base al cosiddetto criterio di *saving* (risparmio): dati due *route* $(0, \ldots, i, 0)$ e $(0, j, \ldots, 0)$, se essi possono essere fusi in un singolo *route* ammissibile $(0, \ldots, i, j, \ldots, 0)$ si ha un risparmio di costo pari a $c_{i0} + c_{0j} - c_{ij}$.

I metodi costruttivi possono essere suddivisi in *sequenziali* e *paralleli*: nei primi viene costruito un *route* alla volta fino all'esaurimento dei vertici; in nessun caso si sceglie tra più *route* in cui è possibile inserire un vertice, perchè ad ogni passo c'è sempre uno ed un solo *route* in via di costruzione. Nel caso parallelo, più *route* possono essere costruiti contemporaneamente; il numero può essere fissato a priori oppure può risultare dalla fusione ripetuta tra *route* più piccoli preesistenti.

Algoritmo di Clarke e Wright [11]

Si tratta probabilmente del più noto algoritmo euristico per il VRP, e si applica in maniera naturale a problemi per i quali il numero di veicoli non è predeterminato. Il metodo ha due versioni, parallela e sequenziale.

(1) Per $i, j = 1, \ldots, n,\ i \neq j$ vengono calcolati i *saving* $s_{ij} = c_{i0} + c_{0j} - c_{ij}$. Vengono creati n *route* del tipo $(0, i, 0)$ per $i = 1, \ldots, n$, e i *saving* vengono ordinati in modo decrescente.

(2') (*Versione sequenziale*). Si considera a turno ogni generico *route* $(0, i, \ldots, j, 0)$, e si determina il primo *saving* s_{ki} o s_{jl} che consenta di fonderlo con un altro *route* contenente l'arco $(k, 0)$ o l'arco $(0, l)$, per dare luogo ad un nuovo *route* ammissibile. Se

Istanza	C/W sequenziale	C/W parallelo	Miglior soluzione nota
E051-05e	625.56	578.64	524.61
E076-10e	1005.25	900.26	835.26
E101-08e	982.48	886.83	826.14
E101-10c	670.01	618.40	555.43
E121-07c	989.42	975.46	909.68
E151-12c	1054.70	973.94	865.94
E200-17c	1383.87	1287.64	1162.55
D051-06c	1671.29	1538.66	1395.85
D076-11c	1646.60	1596.72	1541.14
D101-09c	952.53	875.75	866.37
D101-11c	1708.00	1395.74	1291.45
D121-11c	1299.39	1133.43	1028.42
D151-14c	939.99	833.51	819.56
D200-18c	1291.33	1071.07	1042.11

Tabella 2.1: Confronto tra due implementazioni dell'algoritmo di Clarke e Wright.

tale possibilità esiste la fusione viene effettuata, mentre in caso contrario si riapplica il passo 2' al prossimo *route*. L'algoritmo si arresta quando non è più possibile effettuare alcuna fusione.

(2") (*Versione parallela*). Considerando in ordine i *saving* della lista, si procede come segue. Dato il *saving* s_{ij}, si determina se esistono due *route*, uno contenente l'arco $(0, j)$ e l'altro contenente l'arco $(i, 0)$, che possono essere fusi dando origine ad un *route* ammissibile. Se sì, si opera la fusione.

Prove pratiche dimostrano che la versione parallela è molto più efficace rispetto a quella sequenziale, anche se i risultati ottenuti rimangono comunque lontani dal valore ottimo (tabella 2.1).

Una caratteristica di questi metodi è quella di produrre solitamente *route* validi all'inizio ma poi, via via, sempre meno interessanti; a volte vengono prodotti *route* molto estesi geograficamente. Per ovviare a questi inconvenienti sono stati proposti (Gaskell [26] e Yellow [67]) *saving* generalizzati della forma $s_{ij} = c_{io} + c_{0j} - \lambda c_{ij}$, con λ parametro di forma dei *route*: per valori di λ maggiori si dà più enfasi alla distanza tra i vertici da connettere.

Algoritmi basati su *saving* e *matching*

Interessanti modifiche all'algoritmo standard basato su *saving* sono state proposte da Desrochers e Verhoog [15] e Altinkemer e Gavish [1]. Ad ogni iterazione il *saving* s_{pq}, ottenuto tramite la fusione dei *route* p e q, è calcolato come $s_{pq} = t(S_p) + t(S_q) - t(S_p \cup S_q)$, dove S_k è l'insieme di vertici del *route* k e $t(S_k)$ è la lunghezza della soluzione ottima del problema del commesso viaggiatore (TSP) sui vertici di S_k. Un problema di *matching* a peso massimo è quindi risolto sugli insiemi S_k, impiegando come pesi i valori di *saving* s_{pq}: i *route* corrispondenti al matching ottimale sono, se possibile, combinati tra loro. Una variante di questo algoritmo può stimare il valore di $t(S_k)$, anzichè calcolarlo.

Un altro algoritmo basato su *matching* è stato proposto da Wark e Holt [63]: il *matching* consente di selezionare dei *cluster* (definiti come insiemi ordinati di vertici) da fondere in passi successivi. I pesi possono consistere nei valori di *saving*, ma possono essere anche modificati per favorire l'unione di *cluster* la cui richiesta complessiva è minore della capacità dei veicoli, o la cui lunghezza è significativamente più bassa rispetto alla lunghezza massima eventualmente fissata per i *route*.

Dal punto di vista pratico, gli algoritmi basati su *matching* si rivelano decisamente superiori rispetto al metodo di Clarke e Wright; come spesso accade, però, a risentirne pesantemente è il tempo di calcolo. Tra gli algoritmi citati il metodo di Wark e Holt spicca per qualità dei risultati (tabella 2.2).

Algoritmo di inserzione di Mole e Jameson [45]

Questo algoritmo fa uso di due parametri, λ e μ, per scegliere un vertice che servirà ad espandere un *route* in fase di costruzione. Dati i vertici i, k, j si considerano le due funzioni:

$$\alpha(i,k,j) = c_{ik} + c_{kj} - \lambda c_{ij}$$

$$\beta(i,k,j) = \mu c_{0k} - \alpha(i,k,j)$$

Con queste premesse, i passi dell'algoritmo sono i seguenti.

(i) In presenza di vertici non ancora appartenenti ad un *route*, si costruisce un nuovo *route* emergente $(0, v, 0)$, dove v è uno qual-

Istanza	C/W parallelo	D/V	A/G	W/H	Miglior sol. nota
E051-05e	578.64	586	556	**524.6**	524.61
E076-10e	900.26	885	855	835.8	835.26
E101-08e	886.83	889	860	830.7	826.14
E101-10c	618.40	593	577	**555.4**	555.43
E121-07c	975.46	963	939	911.8	909.68
E151-12c	973.94	914	913	878.0	865.94
E200-17c	1287.64	1292	1210	1176.5	1162.55
D051-06c	1538.66	1559	1464	1418.3	1395.85
D076-11c	1596.72	1562	1551	1548.3	1541.14
D101-09c	875.75	882	874	**866.4**	866.37
D101-11c	1395.74	1424	1351	1321.3	1291.45
D121-11c	1133.43	1133	1085	1038.5	1028.42
D151-14c	833.51	828	834	**819.6**	819.56
D200-18c	1071.07	1058	1047	1043.4	1042.11

Tabella 2.2: Confronto tra euristici basati su *matching*.

siasi dei vertici liberi. In caso contrario, il procedimento ha termine.

(ii) Per ogni vertice libero k si calcola il minimo tra i costi di inserzione ammissibile $\alpha^*(i_k, k, j_k) = min\{\alpha(r, k, s)\}$ per tutti i vertici r ed s adiacenti al *route* emergente: i_k e j_k sono dunque i due vertici che danno il costo α^*. Se nessuna inserzione è ammissibile, si torna al punto 1; altrimenti si seleziona il vertice k^* che rende $\beta^*(i_{k^*}, k^*, j_{k^*}) = max\{\beta(i_k, k, j_k)\}$ tra tutti i vertici k che possono essere inseriti, e lo si inserisce tra i_{k^*} e j_{k^*}.

(iii) Si ottimizza il *route* corrente mediante una procedura *3-opt* [42] e si torna al punto (ii).

I parametri λ e μ consentono di variare la regola di inserzione. Alcune combinazioni dei due parametri corrispondono a regole classiche: ad esempio, per $\lambda = 1$ e $\mu = 0$ l'algoritmo inserisce il vertice che dà origine alla minima distanza supplementare, mentre per $\mu = \infty$ e $\lambda > 0$ il vertice scelto è quello più lontano dal deposito.

Algoritmo di inserzione sequenziale di Christofides, Mingozzi e Toth [10]

Si tratta di un metodo più sofisticato che fa ancora uso di due parametri λ e μ e che si attua in due fasi.

Fase 1 (sequenziale)

(i) Si inizializza un indice di *route* k pari ad 1.

(ii) Si seleziona uno qualsiasi tra i vertici liberi i_k per inizializzare il *route* emergente k. Per ogni vertice libero i, si calcola $\delta_i = c_{0i} + \lambda c_{ii_k}$.

(iii) Sia $\delta_{i^*} = min_{i \in S_k}\{\delta_i\}$, dove S_k è l'insieme di vertici liberi che possono essere inseriti nel *route* k in maniera ammissibile. Si inserisce il vertice i^* nel route k, ottimizzandolo poi mediante una procedura *3-opt*. Si ripete il passo (iii) finchè nessun altro vertice può essere inserito nel *route* k.

(iv) Se tutti i vertici sono stati inseriti il procedimento ha termine; in caso contrario si pone $k := k + 1$ e si torna al passo (ii).

Fase 2 (parallela)

(v) Si inizializzano k *route* $R_t = (0, i_t, 0)$, con $t = 1, \ldots, k$, dove k è il numero di *route* ottenuti al termine della Fase 1. Sia $J = \{R_1, \ldots, R_k\}$ l'insieme dei *route* così inizializzati.

(vi) Per ogni vertice i non associato ad un *route* e per ogni *route* $R_t \in J$ si calcola $\epsilon_{ti} = c_{0i} + \mu c_{ii_t}$ ed $\epsilon_{t^*i} = min\{\epsilon_{ti}\}$. Si associa il vertice i al *route* R_{t^*} e si ripete il passo (vi) fino a che tutti i vertici sono stati assegnati ad un *route*.

(vii) Preso un qualsiasi *route* $R_t \in J$, si pone $J = J \backslash \{R_t\}$. Per ogni vertice i associato al *route* R_t scelto, si calcolano $\epsilon_{t'i} = min_{R_t \in J}\{\epsilon_{ti}\}$ e $\tau_i = \epsilon_{t'i} - \epsilon_{ti}$.

(viii) Si inserisce nel *route* R_t in vertice i^* che soddisfi l'uguaglianza $\tau_{i^*} = max_{i \in S_t}\{\tau_i\}$, dove S_t è l'insieme dei vertici liberi associati al *route* R_t e che vi possono essere inseriti in maniera ammissibile. Il *route* R_t viene ottimizzato mediante una procedura

Istanza	M/J	C/M/T	Miglior soluzione nota
E051-05e	575	547	524.61
E076-10e	910	883	835.26
E101-08e	882	851	826.14
E101-10c	599	565	555.43
E121-07c	969	969	909.68
E151-12c	999	915	865.94
E200-17c	1289	1245	1162.55
D051-06c	1770	1508	1395.85
D076-11c	1590	1612	1541.14
D101-09c	883	876	866.37
D101-11c	1545	1418	1291.45
D121-11c	1259	1093	1028.42
D151-14c	879	827	819.56
D200-18c	1100	1066	1042.11

Tabella 2.3: Confronto tra due euristici ad inserzione sequenziale.

3-opt, e il passo (viii) viene ripetuto finchè nessun altro vertice può essere inserito nel *route* R_t.

(ix) Se $|J| \neq 0$ si va al passo (vi). Se J è vuoto e tutti i vertici sono assegnati l'algoritmo termina; in caso contrario vengono creati nuovi *route* a partire dal passo (i) della Fase 1.

Questo algoritmo di Christofides, Mingozzi e Toth si rivela migliore dell'euristico di inserzione di Mole e Jameson sia dal punto di vista dei risultati ottenuti, sia per quanto riguarda i tempi di calcolo (tabella 2.3).

2.2.2 Metodi a due fasi

Si è già parlato della distinzione, nell'ambito dei metodi a due fasi, tra algoritmi di tipo *cluster-first, route-second* e di tipo *route-first, cluster-second.* In questa sezione verranno presentati alcuni algoritmi del primo tipo, mentre si accennerà solo in generale ai metodi che prevedono la costruzione preliminare di un *route* che copre tutti i vertici.

Algoritmo *sweep*

Questo algoritmo, applicabile ad istanze planari di VRP, è generalmente attribuito a Gillett e Miller [33], che lo hanno reso popolare. Precedente-

mente, alcuni cenni o contributi sono stati proposti da Wren [64] e da Wren e Holliday [65].

L'idea è quella di raggruppare i vertici in *cluster* a seconda della loro posizione angolare rispetto al deposito e, successivamente, di risolvere un'istanza di TSP per ogni cluster. Successivamente, come previsto da alcune implementazioni, i *route* possono essere riottimizzati effettuando scambi di vertici tra *cluster* adiacenti.

Una semplice implementazione può essere così descritta. Si assume che ogni vertice i sia rappresentato, rispetto al deposito, dalle sue coordinate polari (θ_i, ρ_i), dove θ_i rappresenta l'angolo e ρ_i distanza dal deposito. L'angolo θ_i è misurato rispetto ad un angolo di riferimento $\theta_{i^*} = 0$ relativo ad un vertice arbitrario i^*.

(i) I vertici vengono ordinati i secondo valori crescenti di θ_i.

(ii) Si sceglie un veicolo libero k.

(iii) Iniziando dal vertice libero con il minimo valore di θ_i, si assegnano progressivamente vertici al veicolo k finchè la capacità massima del veicolo non viene superata. Eventualmente, ad ogni inserzione si può applicare una tecnica di ottimizzazione come *3-opt*. Se al termine dell'operazione sono presenti altri vertici liberi si ritorna al punto (ii).

(iv) Per ogni *cluster* si risolve un'istanza di TSP, in modo esatto o approssimato.

Algoritmo di Fisher e Jaikumar [23]

Questo noto algoritmo si applica a problemi in cui il numero K di veicoli è fissato a priori. La formazione dei *cluster* di vertici si attua mediante la risoluzione di un problema di assegnamento generalizzato (GAP): in sostanza, si attribuisce una disponibilità di merce pari a Q a K vertici, opportunamente scelti per rappresentare i K route, e si risolve il problema di assegnare a questi ultimi, in maniera ottima, tutti i clienti del VRP, in modo che ogni vertice sia assegnato ad uno ed uno solo di essi e che la richiesta complessiva di merce a carico di ognuno di tali vertici sia al più pari a Q.

I passi dell'algoritmo sono i seguenti:

(i) Si scelgono in V K vertici $j_1, \ldots, j_K$ per inizializzare i K cluster.

(ii) Per ogni vertice libero i si calcola il costo d_{ik} per la sua allocazione nel cluster k: $d_{ik} = min\{c_{0i} + c_{ij_k} + c_{j_k 0}, c_{0j_k} + c_{j_k i} + c_{i0}\} - (c_{0j_k} + c_{j_k 0})$.

(iii) Si risolve un'istanza di GAP con costi d_{ij}, richieste q_i e disponibilità Q.

(iv) Per ogni *cluster* formato si risolve un'istanza di TSP.

Per la fase (i) di inizializzazione, Fisher e Jaikumar [23] propongono di suddividere il piano in K coni e di scegliere come vertici di inizializzazione K vertici fittizi posti sulle bisettrici.

Algoritmo di Bramel e Simchi-Levi

Bramel e Simchi-Levi [6] descrivono un euristico a due fasi in cui la determinazione dei *cluster* avviene mediante la risoluzione di un'istanza di *Capacitated Plant Location Problem*, un altro problema NP-difficile che può essere illustrato come segue. Un certo numero di utenti deve accedere ad un servizio, per l'attivazione del quale sono individuate alcune possibili locazioni ed ogni cliente ha una certa richiesta, quantificata, del servizio stesso. Alle locazioni sono associati costi fissi di attivazione e per ogni possibile accoppiamento utente/locazione c'è uno specifico costo di collegamento. Ogni locazione, infine, può soddisfare un numero massimo di richieste di servizio. Il *Plant Location Problem* richiede che siano individuate sia le locazioni in cui attivare il servizio, sia gli assegnamenti dei clienti alle locazioni, in modo da minimizzare la somma dei costi fissi e dei costi di collegamento.

Nel caso dell'algoritmo di Bramel e Simchi-Levi [6] i clienti corrispondono ai clienti del VRP, con le loro richieste di merce; le locazioni, invece, corrispondono proprio alle posizioni dei clienti: ne devono essere selezionate K in cui individuare i cosiddetti *concentratori*. Ogni locazione ha associata una capacità massima pari a Q, ossia la capacità massima dei veicoli nel VRP.

Una volta identificati i K concentratori, i *route* sono costruiti inserendo ad ogni passo il cliente per cui l'operazione comporta un minimo aumento di costo. Si consideri un *route* parziale k descritto dal vettore $(0 = i_0, i_1, \ldots, i_l, i_{l+1} = 0)$, e sia $T_k = \{0, i_1, \ldots, i_l\}$; sia denotata con $t(T_k)$ la lunghezza di una soluzione ottima di TSP sull'insieme T_k. In questo caso, il costo di inserzione d_{ik} di un cliente libero i nel *route* k è pari a

$t(T_k \cup \{i\}) - t(T_k)$. Il calcolo di d_{ik} in questi termini risulta particolarmente oneroso, per cui sono proposte in particolare due approssimazioni $\overline{d}_{ik}$, pari rispettivamente al costo diretto $min_{h=1,\dots,l}\{2c_{ii_h}\}$ e al minimo costo di inserzione $min_{h=0,\dots,l}\{c_{i_h i} + c_{ii_{h+1}} - c_{i_h i_{h+1}}\}$.

Algoritmo di *Branch-and-Bound* troncato

Adatto per problemi con K variabile, questo metodo proposto da Christofides, Mingozzi e Toth [10] è essenzialmente la semplificazione di un algoritmo esatto proposto da Christofides [9]. Nella procedura, ogni livello dell'albero di ricerca contiene diversi *route* ammissibili; l'implementazione proposta dagli autori effettua un singolo *branching* ad ogni livello, scartando tutti i *route* tranne uno, valutato migliore mediante una opportuna funzione. Nel seguito F_h indica l'insieme dei vertici non ancora visitati al livello h.

(i) Si pongono $h = 1$ e $F_h = V \backslash \{0\}$.

(ii) Se $F_h = \emptyset$, l'algoritmo ha termine. Altrimenti si seleziona un vertice libero $i \in F_h$ e si genera un insieme R_i di *route* contenente i e altri vertici compresi in F_h. Questi *route* sono generati utilizzando una combinazione di due criteri: *saving* e costo di inserzione.

(iii) Si valuta ogni *route* $r \in R_i$ mediante la funzione $f(r) = t(S_r \cup \{0\}) + u(F_h \backslash S_r)$, dove S_r è l'insieme di vertici del *route* r, $t(S_r \cup \{0\})$ è il costo di una buona soluzione di un TSP su $S_r \cup \{0\}$ e $u(F_h \backslash S_r)$ è la lunghezza dell'albero di copertura di costo minimo sui clienti ancora non coperti.

(iv) Si determina un *route* $r^* \in R_i$ tale che $f(r^*) \leq f(r)\ \forall r \in R_i$. Si pone $h = h + 1$ e $F_h = F_{h-1} \backslash S_{r^*}$ e si torna al passo (ii).

In alternativa, è possibile ottenere un albero di ricerca più ricco, ma sempre limitato, mantenendo ad ogni livello alcuni *route* promettenti invece di sceglierne esclusivamente uno.

Se confrontato con l'algoritmo *sweep,* questo metodo si comporta meglio sia in termini di tempo di esecuzione, sia per quanto riguarda la qualità dei risultati ottenuti.

Algoritmi *petal*

Questo metodi sono essenzialmente versioni euristiche del modello di tipo *set-partitioning/set-covering* illustrato alla sezione 2.1.3. Si generano alcuni *route*, detti *petal*, e si effettua una selezione risolvendo un problema di *set-partitioning* della forma:

$$min \sum_{k \in S} d_k x_k$$

con vincoli:

$$\sum_{k \in S} a_{ik} x_k = 1 \quad \forall i = 1, \ldots, n$$

$$x_k \in \{0, 1\} \quad \forall k \in S$$

dove S è l'insieme di *route*, $x_k = 1$ se e solo se il *route* k appartiene alla soluzione, a_{ik} è una variabile binaria pari ad 1 se e solo se il vertice i appartiene al *route* k e, infine, d_k è il costo del *petal* k.

Ryan, Hjorring e Glover [53] hanno dimostrato che se i *route* corrispondono a *cluster* contigui di vertici il problema può essere risolto in tempo polinomiale: per questo motivo si possono ottenere buoni risultati dal punto di vista del tempo di calcolo implementando l'algoritmo come estensione, ad esempio, dell'algoritmo *sweep*.

Come già menzionato, la prima proposta di un algoritmo basato sul *set-partitioning* è dovuta a Balinski e Quandt [4]: il metodo diventa tuttavia impraticabile quando $|S|$ è elevata. Agarwal, Mathur e Salkin [2] propongono un approccio *column-generation* per la risoluzione all'ottimo di piccole istanze, contenenti fino a 25 vertici. Successivamente, regole euristiche per la generazione di *route* semplici e promettenti — i *petal*, appunto — sono state proposte da Foster e Ryan [24] e da Ryan, Hjorring e Glover [53]. In un'altra variante Renaud, Boctor e Laporte [51] fanno un passo avanti proponendo, oltre alla generazione di singoli *petal*, anche la generazione di configurazioni, dette *2-petal*, formate da due *route* accoppiati o intersecati.

I risultati ottenuti con l'algoritmo *2-petal* sono di buon livello, così come molto bassi sono i tempi di calcolo.

Istanza	Sweep	F/J	B/S	BBT	2-petal	Miglior sol. nota
E051-05e	532	524	**524.6**	534	**524.61**	524.61
E076-10e	874	857	848.2	871	854.09	835.26
E101-08e	851	833	832.9	851	830.40	826.14
E101-10c	560	560	-	560	560.08	555.43
E121-07c	933	916	-	924	922.75	909.68
E151-12c	888	885	-	885	877.29	865.94
E200-17c	1230	1230	-	1217	1194.51	1162.55
D051-06c	1518	1518	-	1509	1470.31	1395.85
D076-11c	1776	-	-	1608	1585.20	1541.14
D101-09c	949	876	-	878	885.87	866.37
D101-11c	1389	1420	1461.2	1386	1354.23	1291.45
D121-11c	1079	1014	1088.6	1064	1054.62	1028.42
D151-14c	937	824	826.1	816	824.77	819.56
D200-18c	1266	-	1051.5	1092	1109.14	1042.11

Tabella 2.4: Confronto tra euristici costruttivi.

Metodi *route-first, cluster-second*

Questi algoritmi procedono individuando inizialmente un *tour* che risolva un'istanza di TSP su tutti i vertici del problema, ignorando i vincoli propri del VRP come, ad esempio, quelli di capacità dei veicoli. In una seconda fase, il *tour* viene scomposto in diversi *route* che soddisfano i vincoli dell'istanza di VRP e che ne rappresentano la soluzione finale. Beasley [5] dimostra che la seconda fase di questo procedimento equivale ad un problema classico di cammino minimo su un grafo aciclico, e può essere completata in tempo $O(n^2)$ utilizzando, ad esempio, l'algoritmo di Dijkstra [16].

Dal punto di vista sperimentale, non sono noti risultati di rilievo ottenuti mediante l'applicazione di metodi *route-first, cluster-second*.

2.2.3 Metodi migliorativi

Lo scopo dei metodi migliorativi è quello di partire da una soluzione preesistente di un'istanza di VRP e cercare di migliorarla. Molto spesso metodi non classificabili come migliorativi integrano al loro interno fasi di ottimizzazione di soluzioni temporanee.

Nell'ambito dei metodi migliorativi, due strategie sono immaginabili: è possibile agire su ogni *route* singolarmente oppure cercare di ottimizzare

la soluzione effettuando scambi di vertici o di collegamenti tra due o più *route*.

Metodi di miglioramento *single-route*

Questa classe di metodi comprende, ovviamente, tutti i possibili metodi migliorativi per il TSP. Molte procedure di miglioramento possono essere considerate varianti del meccanismo di λ-*opt* di Lin [42]: λ vertici sono rimossi dal *route*, per ricomporre il quale è necessario aggiungere λ collegamenti tra di essi. Nel generare tutti i possibili schemi di riconnessione, può essere possibile migliorare la soluzione precedente applicando il primo o il migliore. Il miglioramento si arresta ad un minimo locale nel momento in cui nessun altro scambio utile può essere effettuato. La verifica della λ-ottimalità di una soluzione può essere effettuata in tempo $O(n^\lambda)$.

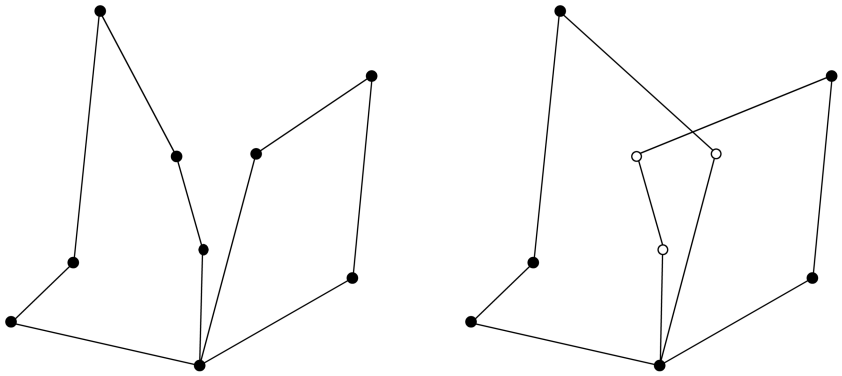
Molte varianti a questo schema di base sono state proposte. Lin e Kernighan [43] propongono un approccio che modifica dinamicamente il valore di λ durante la ricerca; Or [48] propone il metodo *Or-opt*, che consiste nello spostare stringhe di 1, 2 o 3 vertici consecutivi in altre posizioni. Quest'ultimo metodo richiede tempo $O(n^2)$ ed equivale ad una forma ristretta di *3-opt*. Nella stessa ottica, Renaud, Boctor e Laporte [50] hanno sviluppato una versione ristretta di *4-opt*, chiamata *4-opt**, che cerca di effettuare scambi di vertici tra catene di dimensione 2 e catene di dimensione massima w. La verifica dell'ottimalità secondo *4-opt** richiede tempo $O(wn)$.

In un'analisi empirica di questi metodi migliorativi applicati al TSP, Johnson e McGeoch [38] mostrano che, mediamente, i migliori risultati sono ottenuti con un'implementazione accurata del metodo di Lin e Kernighan.

Metodi di miglioramento *multi-route*

I metodi di miglioramento *multi-route* cercano di abbassare il costo di una soluzione mediante scambi di vertici o lati tra diversi *route*. Molti sono i possibili schemi di scambio e la letteratura è molto varia: ci si limiterà qui ad accennare a due contributi.

Thompson e Psaraftis [57] descrivono uno schema di scambio *b-ciclico* di grado k: considerati b *route*, se ne effettua una permutazione circolare e k vertici sono trasferiti da ogni *route* al successivo nella permutazione ciclica. L'applicazione di specifiche sequenze di scambi di grado k e b-ciclici — con $b = 2$ o variabile, con $k = 1$ o 2 — conduce a risultati interessanti.



(a) prima dello scambio (b) dopo lo scambio

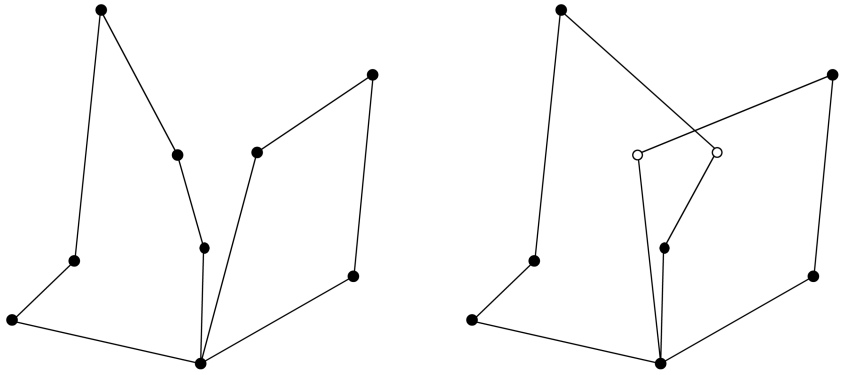
Figura 2.1: *String cross.*

Van Breedam [61] classifica le operazioni di scambio migliorativo in quattro categorie:

- *string cross* (SC): due *stringhe*, o *catene*, di vertici vengono scambiate tra loro incrociando due lati appartenenti a diversi *route* (fig. 2.1);
- *string exchange* (SE): due stringhe di al più k vertici vengono scambiate tra due diversi *route* (fig. 2.2);
- *string relocation* (SR): una stringa composta da al più k vertici — tipicamente con $k = 1$ o 2 — viene spostata dal *route* di appartenenza ad un altro (fig. 2.3);
- *string mix* (SM): viene effettuata la mossa migliore tra SR ed SE.

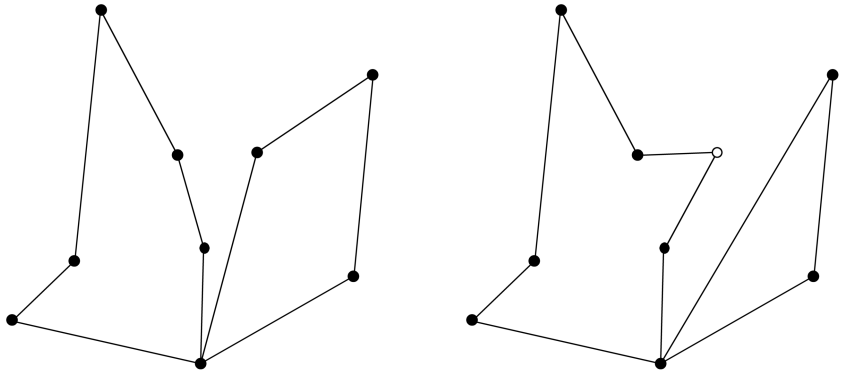
Per eseguire una valutazione di queste strategie, Van Breedam definisce un insieme di parametri che possono influenzare il comportamento della procedura di miglioramento. Questi sono:

- la qualità, buona o scarsa, della soluzione iniziale;
- la modalità di selezione della lunghezza k delle stringhe da considerare: per esempio è possibile valutare gli scambi tra due *route* per tutti i valori considerati di k (tipicamente 1 e 2), oppure si può scegliere



(a) prima dello scambio (b) dopo lo scambio

Figura 2.2: *String exchange.*



(a) prima dello scambio (b) dopo lo scambio

Figura 2.3: *String relocation.*

di valutare tutti gli scambi possibili tra coppie di *route* per un valore fissato di k per poi aumentare in un secondo momento questo valore;

- la strategia di selezione delle mosse, che può seguire i paradigmi *FI* (*First Improvement*) o *BI* (*Best Improvement*). In *FI* la prima mossa migliorativa individuata viene subito effettuata, mentre in *BI* la scelta della mossa migliore avviene solo dopo la generazione di tutte le possibilità di scambio migliorativo.

Nella valutazione Van Breedam ha utilizzato principalmente istanze di CVRP con proprietà particolari — in cui, per esempio, la richiesta sia la stessa per tutti i clienti — e ha ottenuto le migliori soluzioni mediante l'attuazione di mosse *SE*, le cui *performance* sono di più alto livello rispetto alle mosse *SC*, sebbene al prezzo di un tempo di calcolo ben più elevato; parimenti, scambiare stringhe di lunghezza $k = 2$ conduce a risultati migliori ma, ovviamente, in tempi più lunghi. Anche la qualità della soluzione iniziale influisce sui risultati ottenuti: è quasi sempre vantaggioso partire da una buona soluzione.

Queste considerazioni sono confermate anche dall'esperienza con altri tipi di algoritmi euristici e metaeuristici.

2.3 Algoritmi metaeuristici

Le caratteristiche dei metaeuristici, come brevemente accennato all'inizio della sezione 2.2, sono l'esplorazione approfondita di regioni dello spazio delle soluzioni considerate promettenti e l'impiego di sofisticate regole di ricerca nel *neighborhood*, particolari strutture dati e metodi di ricombinazione delle soluzioni. Un'altra caratteristica che spesso li distingue dagli euristici classici è quella di permettere, durante il processo di ricerca, il passaggio attraverso soluzioni non ammissibili o attraverso fasi non migliorative.

I risultati ottenuti con i metaeuristici sono solitamente di qualità superiore rispetto a quelli ricavabili con euristici classici, anche se il tempo di calcolo richiesto può essere sensibilmente più elevato. In genere, inoltre, questo tipo di procedure richiede la corretta valutazione ed impostazione di un certo numero di parametri propri dell'algoritmo, in modo da adattarli al problema da risolvere al fine di ottenere la soluzione migliore possibile.

I metaeuristici finora ideati per il VRP si possono catalogare in sei categorie:

- *Simulated Annealing* (*SA*);
- *Deterministic Annealing* (*DA*);
- *Tabu Search* (*TS*);
- *Algoritmi genetici* (*GA*);
- *Ant System* (*AS*);
- *Reti Neurali* (*NN*).

Nei primi tre casi, la ricerca prende il via da una soluzione iniziale x_1 e passa, ad ogni iterazione t, dalla soluzione corrente x_t alla migliore soluzione del *neighborhood* $N(x_t)$, x_{t+1}, finché non risulta soddisfatto un opportuno criterio di arresto. In generale, se $f(x)$ denota il costo della soluzione x, non è detto che valga $f(x_{t+1}) \leq f(x_t)$: per favorire la diversificazione ed impedire fasi di stallo intorno a minimi locali, può essere necessario passare attraverso sequenze di soluzioni peggiorative.

Nel caso degli algoritmi genetici, ad ogni passo viene esaminata una popolazione di soluzioni: ogni popolazione deriva dalla precedente combinando i suoi migliori elementi e, contemporaneamente, scartando i peggiori. Negli *Ant System* numerose nuove soluzioni sono create ad ogni passo sfruttando le informazioni raccolte durante le iterazioni precedenti. Le reti neurali, invece, sono meccanismi in grado di auto-regolare insiemi di coefficienti interni, progredendo verso soluzioni sempre migliori.

2.3.1 *Simulated Annealing*

In questo tipo di approccio, durante l'iterazione t viene selezionata in modo casuale una soluzione x appartenente al *neighborhood* $N(x_t)$ della soluzione precedente. Se poi $f(x) \leq f(x_t)$, allora x_{t+1} viene posta uguale a x; altrimenti

$$x_{t+1} = \begin{cases} x & \text{con probabilità } p_t \\ x_t & \text{con probabilità } 1 - p_t \end{cases},$$

dove p_t è, in genere, una funzione decrescente di t e di $f(x) - f(x_t)$. La funzione p_t viene comunemente definita nel seguente modo:

$$p_t = e^{-\frac{f(x)-f(x_t)}{\theta_t}}, \tag{2.22}$$

in cui θ_t è un parametro detto *temperatura* all'iterazione t. Di solito θ_t è una funzione decrescente di t: inizialmente θ_t è pari ad un valore predefinito $\theta_1 > 0$ ed è moltiplicato per un coefficiente α (con $0 < \alpha < 1$) ogni T iterazioni, in modo che la probabilità di accettare una soluzione peggiore diminuisca col numero di iterazioni.

Vi sono tre comuni criteri di arresto: il valore f^* della miglior soluzione attuale x^* non diminuisce di almeno il $\pi_1\%$ nell'arco degli ultimi k_1 cicli di T iterazioni, il numero di mosse accettate è minore del $\pi_2\%$ di T per k_2 cicli consecutivi di T iterazioni e, infine, k_3 cicli di T iterazioni sono stati eseguiti.

Algoritmo di Osman

L'implementazione di Osman [49] del *Simulated Annealing* applicato al VRP definisce la struttura del *neighborhood* mediante il meccanismo λ-*interchange*: sono selezionati due *route* p e q e da essi vengono estratti due sottoinsiemi di clienti S_p ed S_q tali che $|S_p| \leq \lambda$ e $|S_q| \leq \lambda$. Tra questi due insiemi viene effettuato uno scambio di vertici che consenta di mobilitarne il più possibile mantenendo ammissibili i *route* p e q. Il numero di ricombinazioni può essere molto elevato, per cui di solito la procedura è implementata con $\lambda = 1$ o $\lambda = 2$ e, nelle implementazioni più efficienti, il primo scambio migliorativo viene immediatamente effettuato, senza attendere la completa esplorazione del *neighborhood*. L'algoritmo implementato da Osman si può descrivere come segue.

Fase 1: discesa

(i) (*Soluzione iniziale*). Mediante il metodo di Clarke e Wright[11], si genera una soluzione iniziale.

(ii) (*Discesa*). Si avvia una ricerca migliorativa con lo schema λ-*interchange*, effettuando sempre il primo miglioramento identificato e proseguendo fino a quando, per la prima volta, non è possibile migliorare con l'esplorazione completa di un *neighborhood*.

Fase 2: ricerca Simulated Annealing

(iii) (*Soluzione iniziale*). Si fissa come soluzione di base quella ottenuta al termine della fase 1 oppure, in alternativa, una prodotta

con l'algoritmo di Clarke e Wright. Si opera una ricerca completa nel *neighborhood* tramite il meccanismo λ-*interchange*, ma senza effettuare alcuna mossa: vengono memorizzati Δ_{max} e Δ_{min}, ossia la più grande e la più piccola variazione in valore assoluto della funzione obiettivo, e β, il numero di scambi potenziali. Si pongono $\theta_1 = \Delta_{max}$, $\delta = 0$, $k = 1$, $k_3 = 3$, $t = 1$, $t^* = 1$; detta x_1 la soluzione corrente, si pone $x^* = x_1$.

(iv) (*Prossima soluzione*). Si esplora in *neighborhood* di x_t mediante lo schema λ-*interchange*: quando viene identificata una soluzione x con $f(x) < f(x_t)$ si pone $x_{t+1} = x$; se $f(x) < f(x^*)$ si pongono $x^* = x$ e $\theta^* = \theta_k$. Se dopo un'intera esplorazione del *neighborhood* nessuna soluzione migliore di x_t viene identificata, indicata con x la migliore soluzione incontrata nel *neighborhood* di x_t, si pone:

$$x_{t+1} = \begin{cases} x & \text{con probabilità } p_t \\ x_t & \text{con probabilità } 1 - p_t \end{cases},$$

dove p_t è definito come in (2.22). Se $x_{t+1} = x_t$, si pone $\delta = 1$.

(v) (*Aggiornamento di temperatura*). Vi sono due modalità di aggiornamento del parametro θ. Quando $\delta = 1$ si ha il cosiddetto *incremento occasionale*: si pongono $\theta_{t+1} = max\{\theta_t/2, \theta^*\}$, $\delta = 0$ e $k = k + 1$; se invece $\delta = 0$ si procede con il *decremento normale*, ponendo $\theta_{t+1} = \theta_t/[(n\beta + n\sqrt{t})\Delta_{max}\Delta_{min}]$. In entrambi i casi si pone, infine, $t = t + 1$; se $k = k_3$ si termina, altrimenti si torna a (iv).

La regola di aggiornamento della temperatura (*cooling schedule*) differisce da quella usualmente seguita nella tecnica *Simulated Annealing*: θ non è sempre diminuito. Quando $x_{t+1} = x_t$, la temperatura corrente può essere dimezzata oppure posta al valore di temperatura con il quale è stata identificata la soluzione migliore corrente.

Esperimenti computazionali (si veda la tabella 2.5) mostrano che questo algoritmo produce in genere discreti risultati, ma raramente identifica soluzioni pari alle migliori conosciute per le istanze di test. I tempi di calcolo tendono ad essere relativamente lunghi.

Istanza	Algoritmo di Osman	Miglior sol. nota
E051-05e	528	524.61
E076-10e	838.62	835.26
E101-08e	829.18	826.14
E101-10c	826	819.56
E121-07c	1176	1042.11
E151-12c	1058	1028.42
E200-17c	1378	1291.45
D051-06c	**555.43**	555.43
D076-11c	**909.68**	909.68
D101-09c	866.75	865.94
D101-11c	890	866.37
D121-11c	1545.98	1541.14
D151-14c	1164.12	1162.55
D200-18c	1417.85	1395.85

Tabella 2.5: Risultati ottenuti con l'algoritmo *SA* di Osman.

2.3.2 *Deterministic Annealing*

La tecnica *Deterministic Annealing* opera in maniera simile a quella *Simulated Annealing*, con la differenza che le mosse vengono accettate o rigettate in base ad una regola deterministica. Due implementazioni standard di questo metodo sono *threshold accepting* di Dueck e Scheurer [20] e *record-to-record travel* di Dueck [19].

Nella prima, ad ogni iterazione t, la soluzione x_{t+1} viene accettata se $f(x_{t+1}) < f(x_t) + \theta_1$, dove θ_1 è un parametro controllato dall'utente. Nel *record-to-record travel*, si definisce *record* la miglior soluzione x^* identificata durante la ricerca. All'iterazione t, la soluzione x_{t+1} è accettata se $f(x_{t+1}) < \theta_2 f(x_t)$, dove θ_2 è un parametro regolabile dall'utente, in genere impostato su un valore di poco superiore ad 1.

In un test condotto su istanze con un numero elevato di vertici, l'algoritmo *record-to-record travel* è stato messo a confronto con l'euristico *Tabu Search* di Xu e Kelly (si veda la sezione 2.3.3). Il confronto mostra che questa tecnica produce spesso soluzioni di miglior qualità, e vicine alle migliori conosciute, con un tempo di calcolo nettamente inferiore.

2.3.3 Metaeuristici *tabu search*

La tecnica *Tabu Search* [34, 35], è una tecnica generale applicabile ad algoritmi euristici. Il suo obiettivo è evitare che, durante l'esplorazione dello

spazio delle soluzioni, l'algoritmo si fermi in corrispondenza di un minimo locale: ottenendo questo viene favorita la diversificazione.

L'applicazione di questa tecnica richiede di definire la struttura del cosiddetto *neighborhood*, che consiste in un insieme di soluzioni ammissibili, ricavabili direttamente a partire da una data soluzione ammissibile x tramite l'applicazione di precise regole euristiche. Allo scopo di evitare cicli in cui si pervenga continuamente alle medesime soluzioni, la tecnica *tabu search* prevede che ogni soluzione ottenuta sia proibita, o *tabu*, per un certo numero di iterazioni: a tal fine, si può pensare di mantenere una lista — con politica, per esempio, FIFO — delle ultime N soluzioni e di adottare opportuni accorgimenti che impediscano il ritrovamento di tutte le iterazioni della lista. A causa di questo, se $f(x)$ denota il costo della soluzione x, non è detto che $f(x_{t+1}) \leq f(x_t)$. Spesso, per diminuire le richieste di tempo di calcolo o di memoria, nella *tabu list* vengono memorizzati solo alcuni attributi delle soluzioni, che consentano possibilmente di riconoscerle senza doverle memorizzare integralmente.

Negli ultimi anni la tecnica *tabu search* è stata applicata con successo al VRP da diversi autori, al punto che è oramai opinione comune che vi siano scarse possibilità di escogitare nuovi euristici classici in grado di dare risultati competitivi.

Di seguito si presentano brevemente alcuni tra i più interessanti metodi per il VRP basati su *Tabu Search*.

Taburoute

L'algoritmo *Taburoute*, proposto da Gendreau, Hertz e Laporte [27], contiene molte caratteristiche interessanti ed innovative ed è piuttosto complesso. Il *neighborhood* di una soluzione è costituito da tutte le soluzioni da essa raggiungibili rimuovendo un vertice dal suo *route* corrente ed inserendolo in un altro *route* contenente uno dei suoi p vertici più vicini. Questa operazione può comportare l'eliminazione di un *route* esistente oppure la creazione di uno nuovo. Lo spostamento viene attuato mediante una procedura, *GENI — Generalized Insertion* — proposta da Gendreau, Hertz e Laporte [28] per il TSP.

Una caratteristica importante di questo algoritmo è quella di permettere il transito attraverso soluzioni non ammissibili rispetto ai vincoli di capacità dei veicoli o di tempo di percorrenza *route*: la funzione obiettivo contiene due termini di penalizzazione, riguardanti l'uno l'eccesso di richie-

sta dei *route* e l'altro l'eccesso del tempo di percorrenza. Questi termini sono pesati da parametri il cui valore viene regolato durante l'esecuzione: ogni dieci iterazioni ogni parametro è diviso per 2 se tutte le 10 precedenti soluzioni risultano ammissibili e, al contrario, raddoppiato se tutte tali soluzioni sono non ammissibili. Questo procedimento produce un'alternanza di soluzioni ammissibili e non, con il risultato di favorire la diversificazione e di abbassare la probabilità di arenarsi su un minimo locale. In vari momenti, durante il processo di ricerca, ogni *route* viene riottimizzato tramite il metodo *US* (*Unstringing and Stringing*), una procedura migliorativa proposta per il TSP dagli stessi Gendreau, Hertz e Laporte [28].

In luogo della *tabu list*, *Taburoute* utilizza un meccanismo detto *random tabu tags*. Se un vertice viene spostato dal *route* r al *route* s durante l'iterazione t, il suo reinserimento nel *route* r rimane interdetto fino all'iterazione $t+\theta$, dove θ è un intero scelto in modo pseudocasuale nell'intervallo $[5, 10]$. Un'altra caratteristica di *Taburoute* è l'uso di una strategia di diversificazione, che consiste nel penalizzare vertici che sono stati spostati frequentemente aumentando, invece, la probabilità di considerare vertici poco mobili.

Infine, *Taburoute* utilizza la tecnica *false starts* per individuare una soluzione iniziale da cui far partire il processo di ricerca. Inizialmente, molte soluzioni sono generate e su ognuna di esse viene effettuata una ricerca per un breve periodo di tempo; la miglior soluzione così identificata è quindi selezionata come punto di partenza per la ricerca principale.

Si darà ora una schematizzazione dei passi dell'algoritmo; come notazione, W è l'insieme dei vertici candidati per lo spostamento verso altri *route*, $q \leq |W|$ è il numero di questi vertici per i quali si effettua un tentativo di reinserzione e i è il numero di iterazioni consecutive ammesse senza miglioramento.

(i) (*Inizializzazione*). Si generano $\left\lceil \sqrt{N}/2 \right\rceil$ soluzioni iniziali e si effettua il *Tabu Search* con $W = V \backslash \{0\}$, $q = 5K$ e $i = 50$. Questo valore di q assicura che la probabilità di selezionare un vertice da ogni *route* è almeno pari al 90%.

(ii) (*Miglioramento della soluzione*). Iniziando dalla miglior soluzione identificata al passo (i), si effettua il *Tabu Search* con $W = V \backslash \{v_o\}$, $q = 5K$ e $i = 50N$.

(iii) (*Intensificazione*). Iniziando dalla miglior soluzione ottenuta

dopo il punto (ii), si definisce W come l'insieme dei $\lceil |V|/2 \rceil$ vertici che sono stati spostati più volte durante i passi (i) e (ii), si pongono $q = |W|$ e $k = 50$ e si effettua il *Tabu Search*.

Le soluzioni prodotte da *Taburoute* sono di elevata qualità e coincidono spesso con le migliori note; il tempo di calcolo, inoltre, risulta assai contenuto.

Algoritmo di Taillard

L'implementazione di Taillard [56] di *Tabu Search* presenta alcune delle caratteristiche di *Taburoute*, come i *random tabu tags* e la diversificazione, ottenuta penalizzando i movimenti dei vertici che sono stati più spesso riposizionati.

Il *neighborhood* è definito mediante il meccanismo λ-*interchange* di Osman [49]. A differenza di *Taburoute* la reinserzione dei vertici spostati avviene mediante metodi standard, senza fare uso del meccanismo generalizzato *GENI*; questo rende più rapide le operazioni di scambio e fa in modo che l'ammissibilità sia sempre mantenuta. Con una certa frequenza i singoli *route* sono riottimizzati applicando l'algoritmo di Volgenant e Jonker [62].

Una caratteristica originale dell'algoritmo di Taillard è la scomposizione del problema principale in sottoproblemi. Nelle istanze planari di VRP, i sottoproblemi sono ottenuti suddividendo il piano in settori aventi per centro il deposito e, successivamente, i settori in regioni concentriche. Ogni sottoproblema contiene i vertici di una regione e può essere risolto in modo indipendente; naturalmente possono rendersi necessari periodici scambi di vertici tra settori adiacenti. Questa suddivisione è sensata in caso di istanze planari in cui il deposito sia localizzato centralmente rispetto alla disposizione dei vertici; se il deposito è invece decentrato oppure l'istanza non è planare l'autore suggerisce un partizionamento basato sul calcolo dell'arborescenza minima avente il deposito come radice. In generale, la scomposizione favorisce l'applicazione dell'algoritmo su sistemi multiprocessore.

I risultati ottenuti con questo metodo sono di livello eccellente.

Algoritmo di Xu e Kelly

Nell'algoritmo di Xu e Kelly [66] la struttura del *neighborhood* è piuttosto articolata. Vengono considerati scambi di vertici tra due *route*, riposizio-

namenti globali di più vertici e miglioramenti locali nei singoli *route*.

La strategia di riposizionamento impiega un modello *network flow* per riassegnare in maniera ottima un gruppo di vertici, spostandoli dal loro *route* di appartenenza ad altri; l'ottimizzazione dei singoli *route* è affidata al metodo *3-opt* [42] e ad una procedura di tipo *Tabu Search*. Il funzionamento dell'algoritmo è regolato da diversi parametri, il cui valore viene modificato dinamicamente durante l'esecuzione. Un insieme di buone soluzioni, le migliori identificate, viene mantenuto e periodicamente utilizzato per reinizializzare la ricerca in base a nuovi parametri.

Questo algoritmo ha prodotto le migliori soluzioni note per alcune istanze, ma in confronto ad altre tecniche *Tabu Search* sembra poco competitivo anche per il tempo di calcolo richiesto e per la difficoltà di impostare in modo ottimale i suoi numerosi parametri.

Memoria adattativa

Uno degli sviluppi di maggiore interesse nell'area del *Tabu Search* è il concetto di *memoria adattativa*, sviluppato da Rochat e Taillard [52]. Una memoria adattativa è una struttura contenente un insieme di buone soluzioni che viene aggiornata dinamicamente durante il processo di ricerca. Alcuni elementi di questo insieme vengono periodicamente estratti e combinati tra loro in qualche modo, così da produrre ulteriori buone soluzioni.

Nel caso del VRP, la combinazione tra diverse soluzioni avviene sulla base di estrazioni e di ricombinazioni di *route* completi. Nel processo di estrazione viene dato un peso maggiore a *route* appartenenti originariamente a soluzioni migliori; nella costruzione di una nuova soluzione si deve poi evitare l'accorpamento di *route* con vertici in comune. A causa di questa restrizione, la ricombinazione conduce in genere ad una soluzione parziale che copre soltanto un sottoinsieme, eventualmente ampio, dei vertici: i rimanenti devono essere successivamente inseriti per mezzo di una procedura euristica costruttiva.

Esperimenti computazionali mostrano che il concetto di memoria adattativa si presta a rendere più efficaci diverse strategie di ricerca. Nel caso del VRP, Rochat e Taillard hanno ottenuto tra l'altro alcune tra le migliori soluzioni tuttora note per i problemi standard comunemente utilizzati come *benchmark*.

Granular Tabu Search

Una tecnica molto promettente è il *Granular Tabu Search* (*GTS*) di Toth e Vigo [60]. Si tratta probabilmente della tecnica che offre al momento il miglior rapporto tra tempo di calcolo e qualità delle soluzioni ottenute.

Alla base del *GTS* c'è l'osservazione che, in un grafo associato ad un'istanza di VRP, i lati lunghi hanno scarsa probabilità di far parte di una soluzione ottima. L'eliminazione a priori di tutti i lati la cui lunghezza eccede una certa soglia, detta *soglia di granularità*, esclude che nel processo di ricerca compaiano molte soluzioni sicuramente poco promettenti. Toth e Vigo propongono di fissare una soglia $\nu = \beta\overline{c}$, dove β è un parametro di *sparsificazione* scelto tipicamente nell'intervallo $[1, 2]$ e $\overline{c}$ è la lunghezza media dei lati appartenenti ad una soluzione ottenuta mediante applicando un euristico veloce. Con questa scelta di β, la percentuale dei lati rimanenti e considerati si attesta al 10/20%. In pratica, il valore di β viene variato dinamicamente quando non vi è miglioramento per un certo numero di iterazioni e viene periodicamente riportato al suo valore originale.

Le soluzioni del *neighborhood* si ottengono mediante un limitato numero di scambi di lati, sia all'interno dello stesso *route*, sia tra *route* distinti. Gli autori propongono una procedura in grado di esaminare tutti i potenziali scambi in tempo $O(|E(\nu)|)$, dove $E(\nu) = \{(i, j) \in E : c_{ij} \leq \nu\} \cup I$ in cui I è un insieme di ulteriori lati che può essere importante considerare, come per esempio lati incidenti al deposito o lati appartenenti a soluzioni di qualità precedentemente ottenute. L'implementazione di Toth e Vigo contiene inoltre alcune caratteristiche di *Taburoute*.

Come già accennato, la qualità delle soluzioni è ottima mentre i tempi di calcolo sono tra i più bassi nell'ambito dei metaeuristici *Tabu Search*.

2.3.4 Algoritmi genetici

Il paradigma alla base degli algoritmi genetici è stato introdotto da Holland [36] nel 1975. Si tratta di tecniche di ricerca che cercano di risolvere i problemi imitando i processi tipici dell'evoluzione naturale. In generale, il procedimento consiste nel mantenere una popolazione di stringhe di bit, chiamate *cromosomi*; ogni cromosoma è la codifica binaria di una certa soluzione dell'istanza da risolvere. Questa popolazione viene fatta evolvere mediante l'applicazione di operatori che si ispirano a fenomeni osservati in natura, quali ad esempio quelli di *riproduzione* e *mutazione*.

Istanza	*Taburoute*	Taillard	Xu e Kelly	*GTS*	Migl. s. nota
E051-05e	**524.61**	**524.61**	**524.61**	**524.61**	524.61
E076-10e	835.77	**835.26**	**835.26**	838.60	835.26
E101-08e	829.45	**826.14**	**826.14**	828.56	826.14
E101-10c	**819.56**	**819.56**	**819.56**	**819.56**	819.56
E121-07c	1073.47	**1042.11**	**1042.11**	1042.87	1042.11
E151-12c	1036.16	**1028.42**	1029.56	1033.21	1028.42
E200-17c	1322.65	1298.79	1298.58	1318.25	1291.45
D051-06c	**555.43**	**555.43**	**555.43**	**555.43**	555.43
D076-11c	913.23	**909.68**	965.62	920.72	909.68
D101-09c	**865.94**	**865.94**	881.38	869.48	865.94
D101-11c	**866.37**	**866.37**	915.24	**866.37**	866.37
D121-11c	1573.81	**1541.14**	1618.55	1545.51	1541.14
D151-14c	1177.76	**1162.55**	-	1173.12	1162.55
D200-18c	1418.51	1397.94	1439.29	1435.74	1395.85

Tabella 2.6: Confronto tra metaeuristici *Tabu Search*.

Una semplice e generica implementazione di un algoritmo genetico può essere esposta nei seguenti termini. Si inizializza preventivamente una popolazione iniziale di cromosomi $X^1 = \{x_1^1, \ldots, x_N^1\}$ e successivamente, ad ogni iterazione $t = 1, \ldots, T$, si applicano k volte — con $k \leq N/2$ — i passi (i)–(iii) seguiti dal passo (iv).

(i) (*Riproduzione*). Si selezionano due cromosomi generatori da X^t, privilegiando statisticamente la scelta dei cromosomi migliori.

(ii) (*Ricombinazione*). Si generano due cromosomi discendenti combinando i due generatori mediante un operatore di *crossover*.

(iii) (*Mutazione*). Con una certa probabilità, solitamente piccola, si applica una mutazione ai discendenti.

(iv) (*Rinnovo generazionale*). A partire da X^t, si crea l'insieme X^{t+1} rimuovendo le $2k$ peggiori soluzioni e sostituendole con le $2k$ generate nelle k precedenti applicazioni dei passi (i)–(iii).

In questo metodo, il parametro T è il numero di generazioni da considerare e k è il numero di selezioni da effettuare per ogni generazione. Al termine delle T iterazioni, si prende come soluzione finale la migliore dell'ultima generazione.

Una caratteristica interessante dell'algoritmo descritto è la sua generalità. Nei termini in cui è stato descritto, infatti, esso può essere applicato a

qualsiasi tipo di problema. Il punto chiave, in realtà, riguarda la definizione degli operatori di crossover e di mutazione. Una volta definita la modalità di traduzione di una soluzione in stringa di bit, ogni problema impone delle regole di validità sulle stringhe/soluzioni: per rappresentare validamente una soluzione, una certa stringa di bit deve soddisfare questi requisiti. Una mutazione che consista nel modificare il valore di un bit, per esempio, potrebbe essere adatta nel caso in cui lo spazio delle soluzioni fosse quello dei numeri naturali con la loro rappresentazione binaria, ma certamente non sarebbe appropriata nel caso di problemi come il VRP, in cui una soluzione è costituita da un insieme di *route*, ognuno dei quali rappresentato da una sequenza di interi che non possono ripetersi.

Nel caso specifico di TSP e VRP, la rappresentazione delle soluzioni in termini di stringhe di bit è considerata poco adatta ed è spesso sostituita con una rappresentazione a sequenza di interi. La posizione di un intero nella stringa indica l'ordine di visita nel *route* del vertice corrispondente; nel caso VRP il vertice 0, che rappresenta solitamente il deposito, può apparire più volte nella stringa e funge di separatore tra i *route*. Una volta accettata una simile rappresentazione, gli operatori di crossover e di mutazione vanno definiti in maniera specifica.

Un operatore di crossover per il TSP, facilmente adattabile al VRP, è stato proposto da Oliver, Smith e Holland [47] con il nome di *order crossover* ed è illustrato in figura 2.4. L'operatore richiede inizialmente di selezionare in maniera casuale due punti di taglio, che nell'esempio si trovano prima della seconda e dopo la terza posizione. Un discendente può allora essere generato ricopiando dal primo generatore la sottosequenza delimitata dai punti di taglio nella stessa posizione e, successivamente, riempiendo una alla volta le altre posizioni, a partire da quella immediatamente successiva al secondo punto di taglio, considerando i vertici nell'ordine in cui compaiono nel secondo generatore. Durante questo riempimento si devono ovviamente evitare le duplicazioni e la stringa deve essere considerata ciclica: al raggiungimento dell'ultima posizione si prosegue ripartendo dalle prime.

Per quanto riguarda l'operatore di mutazione, è possibile applicare dei semplici schemi di scambio o di rimozione e reinserimento (*RAR — Remove-and-Reinsert*). Quest'ultimo è illustrato tramite un esempio in figura 2.5: a seguito di una scelta casuale, il vertice 2 viene spostato dalla posizione 3 alla posizione 5. Possono essere impiegati altri operatori di mutazione più complessi, come ad esempio l'operatore di inversione proposto da

Generatore 1	:	0	3	2	1	5	4
Generatore 2	:	0	5	2	4	3	1
Discendente 1 (passo 1)	:	-	3	2	-	-	-
Discendente 1 (passo 2)	:	0	3	2	4	1	5
Discendente 2 (passo 1)	:	-	5	2	-	-	-
Discendente 2 (passo 2)	:	0	5	2	1	4	3

Figura 2.4: Operatore *order crossover*.

Cromosoma	:	0	4	**2**	5	3	1
Cromosoma mutato	:	0	4	5	3	**2**	1

Figura 2.5: Operatore di mutazione *remove-and-reinsert*.

Holland [36].

La letteratura sullo sviluppo di algoritmi genetici applicati al CVRP è piuttosto scarsa. Molte implementazioni sono state valutate per il TSP e per problemi di VRP più vincolati, come il VRP con *time windows* (VRPTW) o con vincoli di precedenza. L'abbondanza di materiale è giustificata, in questi due ultimi casi, dalla scarsità di algoritmi convenzionali e di risultati particolarmente interessanti: grazie alla generalità della loro definizione, in cui non si richiamano le caratteristiche specifiche e la struttura del problema da risolvere, gli algoritmi genetici si adattano bene anche a versioni difficili del problema.

Nelle poche implementazioni sviluppate e valutate, i risultati ottenuti su istanze di CVRP sono di scarsa rilevanza. Nel caso di problemi con *time windows*, invece, è stato possibile ottenere con algoritmi genetici ottimi risultati: per questo motivo sono probabili in futuro sviluppi e miglioramenti anche nel campo dei problemi meno vincolati.

2.3.5 *Ant System*

Durante la ricerca del cibo, le formiche marcano il cammino grazie alla secrezione di una sostanza, il *feromone*, che è riconoscibile da tutte le appartenenti alla colonia. La presenza di questa sostanza costituisce un'informazione per le altre formiche, che vi sono attratte; la quantità di feromone presente su un percorso dipende dalla sua lunghezza dalla qualità della fonte di cibo raggiungibile percorrendolo. Con il passare del tempo i cammini più interessanti, cioè quelli che portano a fonti di cibo più ricche, diventano

sempre più frequentati e in essi è depositata una maggiore quantità di feromone. Questo rappresenta un metodo naturale efficiente tramite il quale le formiche si procurano il nutrimento.

Prendendo ispirazione da questi processi naturali Colorni, Dorigo e Maniezzo [12] hanno proposto una nuova classe di metaeuristici, gli *Ant System*, in cui formiche artificiali alla ricerca nello spazio delle soluzioni simulano vere formiche nell'esplorazione dell'ambiente, i valori delle funzioni obiettivo rappresentano la qualità delle fonti di cibo e i valori memorizzati in una memoria adattativa (si riveda al riguardo la sezione 2.3.3) sono associati alle tracce di feromone.

Per illustrare i principi di base di questa strategia si descriverà un semplice metaeuristico per il TSP, il problema al quale Colorni, Dorigo e Maniezzo hanno applicato per primo il loro metodo. Ogni lato (v_i, v_j) del grafo è associato a due valori: la *visibilità* n_{ij}, un valore costante pari all'inverso della lunghezza del lato, e la *traccia di feromone* Γ_{ij}, il cui valore è aggiornato dinamicamente al procedere dell'algoritmo. Ad ogni iterazione, formiche artificiali partono dai vertici del grafo costruendo n nuovi *tour* mediante un euristico *nearest neighbor* probabilistico con una misura di distanza modificata. Questa misura viene calcolata a partire da n_{ij} e Γ_{ij}, ed è tale da favorire la selezione di vertici vicini o situati all'estremità di collegamenti con elevata presenza di feromone. Al termine di ogni iterazione i valori delle tracce sono aggiornati ammettendo dapprima che una frazione pari ad $1 - \rho$ — con $0 \leq \rho \leq 1$ — del vecchio feromone si disperda e, successivamente, distribuendone nuove quantità lungo i lati dei *tour* appena costruiti. Se il lato (v_i, v_j) fa parte di un *tour* di lunghezza L_k, appena costruito dalla formica artificiale k, il valore della traccia di feromone viene aumentato della quantità $\Delta_{ij}^k = 1/L_k$. La dispersione del feromone è necessaria per evitare che soluzioni di scarsa qualità, quali ad esempio quelle individuate nelle prime iterazioni, condizionino troppo il processo di ricerca successivo, durante il quale possono emergere soluzioni migliori. La formula di aggiornamento al termine dell'iterazione t è dunque la seguente:

$$\Gamma_{ij}^{t+1} = \rho\Gamma_{ij}^{t} + \sum_{k=1}^{N} \Delta_{ij}^{k},$$

dove N è il numero di formiche. L'intero processo di costruzione e aggiornamento viene ripetuto per un numero prefissato di iterazioni.

Il metodo è stato successivamente raffinato mediante l'aggiunta di di-

verse estensioni e caratteristiche (Dorigo, Maniezzo e Colorni [18], Dorigo e Gambardella [17]); dagli esperimenti condotti si può concludere che il metodo può produrre talvolta risultati interessanti, ma non può competere con altri tipi di metaeuristici senza essere in qualche modo ibridato con procedure di ottimizzazione locale.

Pochi contributi sono stati dati nell'applicazione di *Ant System* al VRP. Kawamura et. al. [39] propongono una complessa variante ibrida che prevede di applicare il metodo *2-opt* e introduce regole probabilistiche che ricordano la strategia *Simulated Annealing* (sezione 2.3.1). Altri due contributi vengono da Bullnheimer, Hartl e Strauss [8, 7]. Nel primo dei due articoli viene proposto un sistema ibrido in cui, al termine di ogni iterazione e prima dell'aggiornamento delle tracce di feromone, ogni *route* identificato viene sottoposto ad un miglioramento locale tramite *2-opt*. Nella scelta dei vertici da visitare da parte delle formiche, inoltre, vengono tenuti in considerazione termini relativi alla capacità dei veicoli e al *distance saving* rispetto al deposito. Nella fase di aggiornamento dei valori di feromone, infine, si tiene conto della presenza di un certo numero di formiche *elitarie*, che si suppongono sempre percorrere i *route* corrispondenti alla migliore soluzione identificata al momento del ricalcolo di Γ. Esperimenti computazionali dimostrano che il miglioramento locale e le formiche elitarie sono accorgimenti utili per migliorare le prestazioni dell'algoritmo, anche se i risultati sulle istanze di test sono di qualità non omogenea.

Nel loro secondo contributo, Bullnheimer, Hartl e Strauss ridefiniscono il loro algoritmo sotto diversi aspetti. Innanzitutto, nella scelta dei vertici da visitare il calcolo del termine relativo alla capacità dei veicoli, piuttosto dispendioso, viene soppresso; in secondo luogo solo i $\lfloor n/4 \rfloor$ vertici più vicini sono considerati al momento di scegliere il successivo cliente da visitare. Infine, solo le cinque migliori soluzioni identificate sono utilizzate al momento dell'aggiornamento delle tracce di feromone, e la quantità rilasciata dipende dalla qualità relativa tra le soluzioni stesse. I risultati ottenuti su istanze di test sono di un certo interesse e i tempi di calcolo risultano più che accettabili.

In considerazione della scarsità di esperienze nell'applicazione di questi sistemi a formiche artificiali al VRP, i risultati sono nel complesso incoraggianti e sembra ragionevole attendersi miglioramenti e sviluppi interessanti in futuro.

2.3.6 Reti neurali

Le *reti neurali artificiali*, o semplicemente *reti neurali*, sono modelli computazionali la cui struttura prevede l'interconnessione di diversi elementi di elaborazione elementari. Queste celle di elaborazione si ispirano ai neuroni del sistema nervoso umano, mentre i loro collegamenti vorrebbero in qualche modo emulare le sinapsi. Nelle reti artificiali ogni collegamento è associato ad un proprio peso numerico, il cui valore varia dinamicamente nel tempo in funzione dell'esperienza maturata durante la computazione: è proprio il valore di questi coefficienti a determinare il comportamento della rete nel suo complesso. Una caratteristica particolare delle reti neurali è la capacità di induzione: esse sono in grado di astrarre concetti generali a partire da esempi specifici e questo processo avviene proprio grazie all'aggiustamento dei pesi associati alle connessioni.

L'applicazione delle reti neurali a problemi di ottimizzazione combinatoria ha origine da un lavoro di Hopfield e Tank [37]. Il loro e altri modelli, come ad esempio quelli di *rete elastica* (*EN — Elastic Net*) di Durbin e Willshaw [21] e di *mappa auto-regolante* (*SOM — Self-Organizing Map*) di Kohonen [40], sono stati applicati al TSP dando, peraltro, risultati non competitivi con quelli ottenuti da altri metaeuristici.

I modelli *EN* e *SOM*, solo ispirati alla definizione di rete neurale, prevedono sorte di *tour* deformabili, che si adattano gradualmente alla disposizione effettiva dei vertici dell'istanza. La figura 2.6 illustra schematicamente questo meccanismo. I vertici dell'istanza sono rappresentati in nero, mentre i cerchi bianchi raffigurano le cosiddette *unità* del modello, simili a dei segnaposto per i vertici effettivi; le unità sono tra di loro collegate e formano un circuito chiuso. A partire da una configurazione iniziale arbitraria (2.6a), la posizione di ognuna delle unità viene progressivamente regolata (2.6b) fino a che ognuna di esse si avvicina sufficientemente ad uno dei vertici (2.6c): ciò ricorda in qualche modo il fenomeno di aggiustamento dei pesi delle reti neurali. Quando l'evoluzione raggiunge questo stato, ogni vertice viene assegnato all'unità ad esso più vicina (2.6d).

Modelli deformabili di questo tipo possono essere facilmente applicati a problemi puramente geometrici, ma sono difficilmente in grado di gestire la presenza di vincoli aggiuntivi, come quelli del CVRP e, specialmente, delle varianti più difficili di VRP. Recentemente sono state sviluppate versioni, ispirate in particolare al modello *SOM*, adatte al CVRP: si tratta di generalizzazioni del metodo precedente, in cui coesistono diversi circuiti

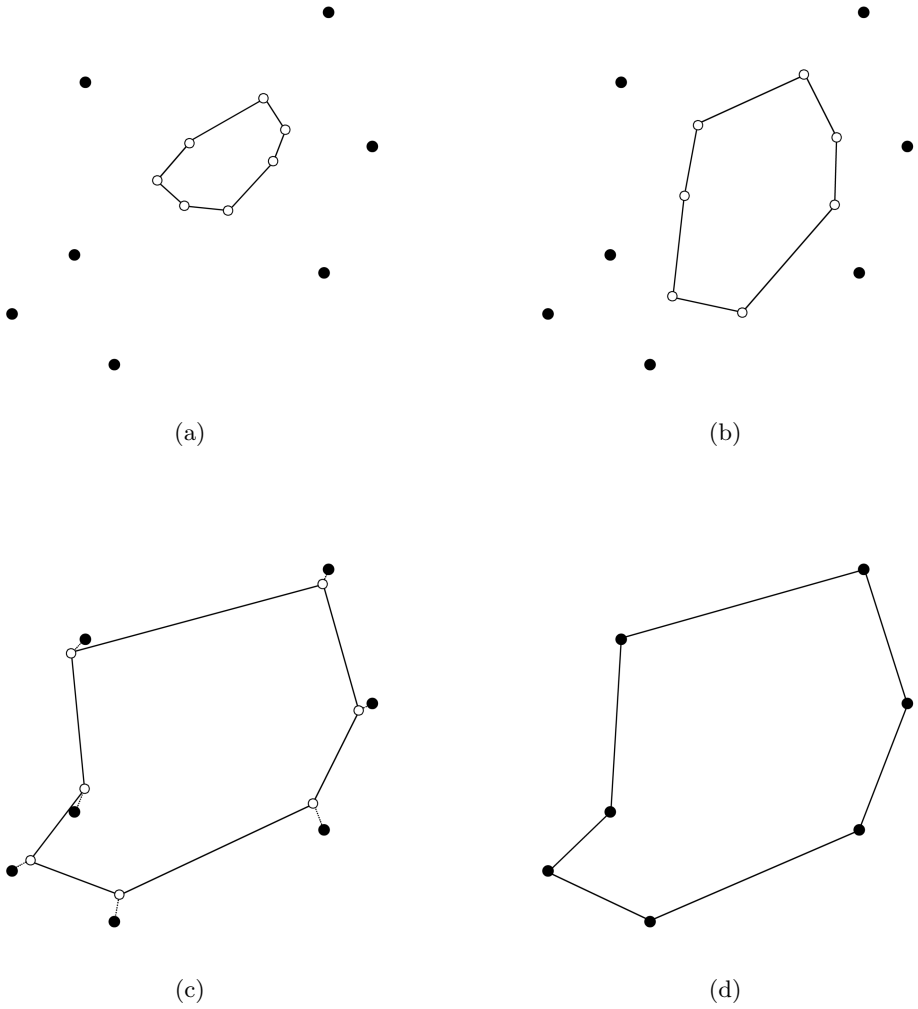


Figura 2.6: Evoluzione di modelli deformabili.

deformabili, uno per ogni *route* del problema. In questi casi è necessario anzitutto risolvere la competizione tra le unità del modello nell'atto dell'avvicinamento ai vertici ed in secondo luogo affrontare il problema della suddivisione ammissibile rispetto ai vincoli di capacità dei veicoli. Alcuni di questi algoritmi sono stati proposti da Ghaziri [30, 31, 32], Matsuyama [44] e Schumann e Retzko [55].

La procedura suggerita da Ghaziri [30] si può descrivere nel modo seguente.

(i) Si considera, ciclicamente, il vertice successivo dell'istanza e lo si indica come *vertice corrente*.

(ii) Si associa una probabilità di selezione ad ogni anello. Questa probabilità deve dipendere sia dalla distanza dell'anello dal vertice corrente, sia dal carico attuale dell'anello stesso in termini di vertici temporaneamente assegnati (si veda il punto (iv)).

(iii) Secondo la probabilità definita al punto (ii), si seleziona uno degli anelli.

(iv) Si assegna temporaneamente il vertice corrente all'unità più vicina dell'anello selezionato, e si deforma quest'ultimo in modo da avvicinare l'unità, e alcune tra le sue unità vicine, al vertice.

(v) Si ripetono i precedenti passi fino a quando ogni vertice ha un'unità del modello sufficientemente vicina.

(vi) Infine, si assegna permanentemente ogni vertice alla sua unità più vicina, così da pervenire una soluzione finale.

L'aspetto forse più importante riguarda la probabilità di selezione degli anelli, definita al punto (ii). Mediante questo meccanismo si fa in modo che essi non si avvicinino troppo simultaneamente a vertici le cui richieste di merce superano la capacità dei veicoli; inoltre, fissando eventualmente alcune probabilità a zero, è possibile adattare questo metodo alla risoluzione di problemi più vincolati, come ad esempio il VRP con vincoli di tempo. Un altro contributo di Ghaziri [32] è stato proprio lo sviluppo di un algoritmo simile adattato a quest'ultimo tipo di problema.

I risultati computazionali riguardanti, in generale, questi metodi evidenziano la loro capacità di produrre soluzioni di discreta qualità, ma al-

lo stesso tempo mostrano che al momento non sono competitivi con altri metaeuristici.

2.4 Conclusioni

L'analisi dei modelli matematici e degli algoritmi euristici e metaeuristici applicati finora al Vehicle Routing Problem mostra che si dispone ormai di mezzi adeguati per affrontare con successo istanze con centinaia di vertici. I risultati migliori si ottengono con gli approcci metaeuristici, ed in particolare con gli algoritmi basati sulla strategia *Tabu Search*: in questa famiglia si trovano metodi in grado di risolvere all'ottimo, con un certo sforzo computazionale, molte delle istanze oggi considerate difficili, ma anche metodi la cui applicazione porta ad ottimi risultati in tempi assolutamente accettabili.

Procedure basate su algoritmi genetici puri o metodi ispirati alle reti neurali sono al momento troppo poco studiati e non se ne possono dare valutazioni; al contrario, approcci *Simulated Annealing* o *Deterministic Annealing* sono stati oggetto di maggiore interesse ma sembrano poco competitivi. Al di là del *Tabu Search*, le tecniche che paiono promettere miglioramenti per il futuro sono gli *Ant System* e gli algoritmi genetici ibridati con tecniche di ottimizzazione locale.

Le istanze oggi utilizzate come *benchmark* paiono ormai inadeguate: è forse arrivato il momento di passare a problemi di maggiori dimensioni.

Capitolo 3

Un nuovo algoritmo

In questo capitolo verranno esposti i concetti alla base di un nuovo algoritmo euristico sviluppato per il CVRP e adattabile, in futuro, a varianti più complesse e vincolate, come ad esempio VRP con vincoli di lunghezza dei *route* (DVRP, DCVRP) o con vincoli di precedenza. Il buon livello dei risultati ottenuti, documentati al termine dell'esposizione con risultati su istanze classiche di *benchmark*, testimonia la validità dell'idea e incoraggia ulteriori approfondimenti.

L'algoritmo presenta diverse caratteristiche interessanti. In primo luogo, oltre a ricavare autonomamente una soluzione, esso è in grado di partire da una soluzione preesistente per cercare di migliorarla. Negli esperimenti effettuati, è stato più volte possibile migliorare soluzioni ottenute con altri euristici o metaeuristici; come testimoniato nella sezione 3.7, in diversi casi questo procedimento ha portato al miglioramento dei migliori risultati finora noti in letteratura[1]. Un'altra caratteristica degna di considerazione è l'assenza di parametri da impostare a cura dell'utente. Alcuni metaeuristici, pur eccellenti dal punto di vista delle prestazioni, sono ritenuti poco competitivi a causa della grande quantità di difficili configurazioni da tarare in base alle caratteristiche della particolare istanza da risolvere.

Dopo aver introdotto la notazione utilizzata, il capitolo inizia descrivendo sommariamente la struttura di un *neighborhood* proposto da Sarvanov e Doroshko nel 1981 per il TSP. Da questo, e da alcune semplici considera-

[1] In un recentissimo lavoro di Fukasawa *et. al.* [25], reso noto come rapporto tecnico nel settembre del 2003 ma non ancora pubblicato, sono state risolte all'ottimo, mediante l'applicazione di un algoritmo esatto *branch-and-cut-and-price*, molte istanze precedentemente ancora aperte. Questi risultati, ottenuti con grosso sforzo computazionale in termini di tempo di calcolo, hanno permesso di dimostrare che alcune delle soluzioni migliorate da noi ottenute sono ottime o molto vicine all'ottimo.

zioni, viene ricavato un semplice algoritmo per il problema del commesso viaggiatore, ma abbastanza generale da poter essere applicato al problema del *routing* di veicoli, dotato di una complessità intrinseca molto maggiore. Dopo aver analizzato le estensioni applicabili a questo algoritmo, ne viene proposta, nelle sue linee generali, una versione per il VRP. La sezione 3.6 illustra alcuni dettagli implementativi e alcune strategie di particolare interesse; la trattazione, infine, si chiude con la presentazione di risultati su istanze classiche di *benchmark* e con alcune considerazioni conclusive.

3.1 Notazione

Al fine di esporre in maniera formale i concetti trattati nella prossima sezione, è opportuno introdurre una simbologia ed una notazione adeguate.

La soluzione di un'istanza di TSP, denominata *tour*, consiste in un ciclo hamiltoniano, ossia una sequenza ciclica di vertici in cui ognuno di essi compare una ed una sola volta. Molto frequentemente per il TSP si considerano istanze descritte da grafi completi, e per gli scopi di questa trattazione si farà qui riferimento a questi casi. Denotato con $V = \{v_1, \ldots, v_n\}$ l'insieme dei vertici di una generica istanza, S_V rappresenta l'insieme di tutte le permutazioni dei vertici di V; una soluzione dell'istanza corrisponde allora ad un elemento dell'insieme S_V. Considerata una permutazione $\pi \in S_V$, si indica con $\pi[i]$ il vertice che occupa la i-esima posizione all'interno della sequenza e con π_i l'etichetta di tale vertice; la sequenza in sé viene indicata con la notazione $\pi = \langle v_{\pi_1}, \ldots, v_{\pi_n} \rangle$.

È opportuno, inoltre, dare una definizione di *neighborhood* in generale. Si consideri un'istanza $\mathcal{I}$ di un generico problema di ottimizzazione; data una soluzione s di questa istanza, il suo *neighborhood* $\mathcal{N}_s$ consiste in una famiglia di soluzioni $\{s_1, \ldots, s_m\}$ ognuna delle quali risulta ottenibile a partire da s mediante l'applicazione di operazioni sottoposte a regole ben definite.

3.2 Un *neighborhood* esponenziale per il TSP

3.2.1 Il *neighborhood* Assign

In uno studio pubblicato nel 1981, Sarvanov e Doroshko [54] propongono la struttura di un particolare *neighborhood*, chiamato Assign, per il problema del commesso viaggiatore. Assign è così strutturato:

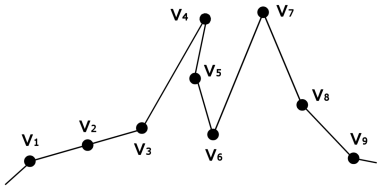
$$\textsc{Assign}_\pi = \{\phi \in S_V \mid \phi[2i-1] = \pi[2i-1] \quad \forall i = 1, \ldots, \left\lceil \frac{n}{2} \right\rceil\}.$$

In termini meno formali, le soluzioni appartenenti al *neighborhood* di una generica permutazione π sono tutte e sole quelle ottenibili a partire da π permutando in maniera arbitraria i vertici che occupano posizioni pari nella sequenza.

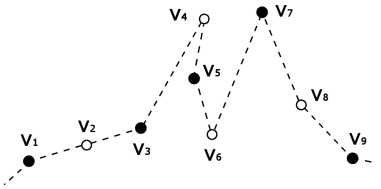
Partendo da una soluzione preesistente, un metodo migliorativo può esplorarne il *neighborhood*, con l'intento di pervenire ad una soluzione di costo minore. Il funzionamento di tale metodo può essere scandito in due fasi:

- *estrazione*, durante la quale i vertici in posizione pari vengono rimossi dal *tour* per lasciare posto ad un uguale numero di *lacune*;
- *riposizionamento*, in cui i vertici estratti vengono ricollocati, in corrispondenza delle lacune, in maniera ottima.

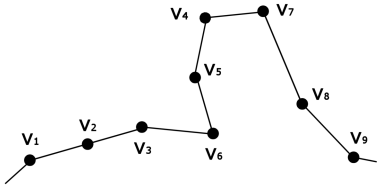
Il semplice esempio di figura 3.1 può aiutare a comprendere il tipo di mosse migliorative effettuabili. In figura 3.1a è rappresentata una porzione di una soluzione ammissibile di un'istanza di TSP: questa porzione corrisponde alla sequenza di vertici $\langle v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9 \rangle$. A seguito della selezione di tutti i vertici in posizione pari, v_2, v_4, v_6 e v_8 vengono estratti mentre tutti gli altri appartenenti alla sottosequenza rimangono nelle proprie posizioni. Oltre ai vertici fissi, indicati in nero, in figura 3.1b sono evidenziati con dei pallini bianchi i vertici estratti, la cui posizione all'interno della sequenza può cambiare a seguito della successiva fase di riposizionamento. Tutti i lati sono rappresentati in tratteggio: al termine della procedura alcuni di essi potranno ancora far parte della soluzione, mentre altri vi saranno esclusi. Indicando con il simbolo "$-$" la presenza di una lacuna, la sequenza corrispondente alla sottofigura (b) è $\langle v_1, -, v_3, -, v_5, -, v_7, -, v_9 \rangle$ mentre i vertici estratti sono inseriti nell'insieme $\mathcal{E} = \{v_2, v_4, v_6, v_8\}$. L'ultimo passo consiste nel considerare le possibili ricollocazioni dei vertici estratti nelle quattro lacune della sequenza, e tra queste selezionare quella che dà luogo ad una nuova soluzione con costo minore possibile. Nella sottofigura (c) è rappresentata la nuova soluzione, in cui i vertici v_4 e v_6 si sono scambiati di posto nella sequenza mentre v_2 e v_8 sono rimasti nelle loro posizioni originarie. La nuova sequenza è $\langle v_1, v_2, v_3, v_6, v_5, v_4, v_7, v_8, v_9 \rangle$.



(a) soluzione originaria



(b) selezione dei vertici da estrarre



(c) riposizionamento

Figura 3.1: Semplice esempio di selezione e riposizionamento.

Nell'esempio appena esaminato lo stesso miglioramento si sarebbe potuto ottenere, ad esempio, mediante l'applicazione della tecnica *2-opt* [42]. In realtà, immaginando estrazioni di vertici su tutto il circuito o comunque su una porzione più grande, il numero di diverse possibili reinserzioni diviene notevolmente elevato: possono quindi verificarsi dinamiche di scambio molto articolate, in cui i miglioramenti ottenibili in alcune zone possono bilanciare eventuali peggioramenti in altre.

Il *neighborhood* ASSIGN è stato oggetto di particolare interesse — si veda, ad esempio, il contributo Deineko e Woeginger [14] — grazie a due importanti proprietà di cui gode, alle quali si farà brevemente cenno. La prima di queste consiste nella sua cardinalità esponenziale: fissati i $\lceil \frac{n}{2} \rceil$ vertici dispari, infatti, rimangono $\lfloor \frac{n}{2} \rfloor !$ modi diversi di rimescolare i vertici nelle posizioni pari. L'altra importante proprietà è la possibilità di effettuare la ricerca completa in questo *neighborhood* in tempo polinomiale: il problema di assegnamento che richiede di ridistribuire i $\lfloor \frac{n}{2} \rfloor$ vertici pari si può risolvere in tempo $O(n^3)$ tramite metodi ben noti in letteratura.

Al di là di queste interessanti proprietà, il *neighborhood* ASSIGN fornisce l'ispirazione per implementare una famiglia di algoritmi euristici, dalla

struttura semplice e generale, per il problema del commesso viaggiatore. Se confrontato con la sterminata mole di proposte nella letteratura del TSP, un simile algoritmo ha scarse probabilità di essere competitivo; guardando oltre, tuttavia, l'approccio sembra essere più promettente per il VRP, problema caratterizzato da una difficoltà intrinseca molto maggiore.

3.2.2 Un semplice algoritmo euristico per il TSP

Una prima naturale osservazione sulla struttura di ASSIGN porta ad identificare un'intera famiglia di *neighborhood* dotati di simili proprietà. A partire da una soluzione ammissibile si può pensare di selezionare, con opportuni criteri svincolati dalla semplice alternanza pari/dispari, alcuni vertici non adiacenti. Ogni criterio di scelta che rispetti questo vincolo dà origine a *neighborhood* ricercabili in tempo polinomiale. Alcuni esempi sono illustrati in figura 3.2.

Si può allora pensare ad un algoritmo che, sfruttando l'efficienza della ricerca all'interno di questi insiemi di soluzioni, parta da una soluzione iniziale arbitraria e produca una sequenza di selezioni e reinserzioni di vertici fino al verificarsi di una opportuna condizione di arresto, pervenendo così ad una soluzione finale.

Se $\mathcal{S}$ è l'insieme degli schemi di selezione disponibili, l'algoritmo si può schematizzare come segue.

Algoritmo *SR* — *Selezione e Ridistribuzione*

(i) (*Inizializzazione*). Si calcola una soluzione di partenza applicando un metodo semplice, come ad esempio *nearest neighbor*.

(ii) (*Selezione*). Si applica alla soluzione corrente il prossimo schema di selezione disponibile nell'insieme $\mathcal{S}$.

(iii) (*Ridistribuzione*). Si risolve il problema di assegnamento dei vertici selezionati al punto (ii) per riposizionarli nelle *lacune*. Se il riassegnamento ha prodotto una nuova soluzione di costo inferiore, si applica la procedura di miglioramento *3-opt*.

(iv) (*Valutazione*). Se vi è stato almeno un miglioramento nell'arco delle ultime $|\mathcal{S}|$ iterazioni, si torna al punto (ii). In caso contrario, l'algoritmo termina.

(a) Assign-Even

(b) Odd-Even-Alternate

(c) Half-Odd-Half-Even

(d) Two-Odd-Two-Even

(e) Random

Figura 3.2: Semplici esempi di criteri di scelta che originano *neighborhood* con proprietà simili ad Assign.

L'implementazione di questo semplice algoritmo ha fornito risultati distanti pochi punti percentuali dai migliori noti in letteratura per diverse istanze di TSP con matrice dei costi simmetrica.

3.3 Dal TSP al VRP

Al di là della sua evidente semplicità, una importante caratteristica dell'algoritmo presentato al termine della precedente sezione è la propensione ad essere adattato anche a varianti vincolate del TSP o, addirittura, al VRP.

La sensazione che anima questo lavoro è che proprio il *Vehicle Routing Problem* rappresenti un campo adatto per verificare la validità di questo approccio e, soprattutto, di alcune sue importanti estensioni. Nel VRP, infatti, a causa della presenza di più *route* sottoposti ai vincoli di capacità dei veicoli, la disposizione geografica dei vertici o le loro reciproche distanze non sono gli unici fattori che concorrono a rendere buona una soluzione: l'altro aspetto fondamentale riguarda la spartizione della richiesta, ossia l'assegnazione dei clienti ai veicoli e, quindi, ai *route*. Per questo motivo, procedure che tentano di effettuare scambi di vertici o di lati magari poco efficaci per il TSP possono rivelarsi invece interessanti se applicate al VRP.

3.4 Limiti ed estensioni

È evidente che l'applicazione dell'algoritmo 1 ad ogni *route*, preso singolarmente, può portare a miglioramenti della soluzione di un'istanza di VRP: tutte le procedure migliorative adatte al TSP possono essere applicate in questo modo al problema del *routing*. Allo stesso modo, risulta chiaro che in particolari situazioni, quali ad esempio la presenza di *route* che si intersecano, la procedura così com'è stata descritta può già essere utile. Ci sono, tuttavia, molte situazioni in cui il semplice scambio di posto nella sequenza tra vertici non può essere efficace.

Per cercare di evidenziare proprio sul VRP le potenzialità e i limiti della procedura precedentemente descritta conviene rifarsi a tre semplici situazioni, illustrate di seguito.

Si consideri la figura 3.3, che rappresenta una parte di un'ipotetica soluzione, non ottima, ricavata su una certa istanza. Tralasciando per un istante le capacità dei veicoli e le richieste dei clienti, peraltro non indicate, si può notare a colpo d'occhio che la posizione del vertice v_3 non è partico-

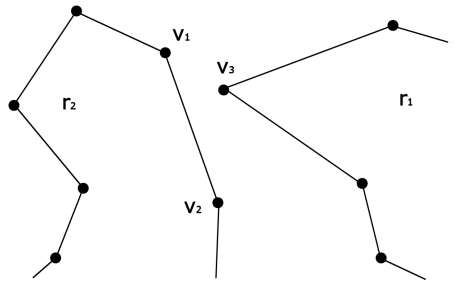


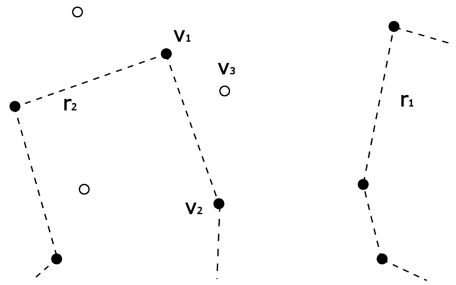
Figura 3.3: L'assegnazione di v_3 al *route* r_1 non è ottimale.

larmente felice: è evidente che questa soluzione potrebbe essere migliorata "spostando" v_3 nel *route* r_2 ed inserendolo tra i vertici v_1 e v_2 dopo aver rimosso il loro collegamento. Un'operazione di questo tipo, purtroppo, non può essere tra quelle consentite dall'algoritmo *SR*, che prevede in ogni caso lo scambio di posizione tra vertici e non ammette uno spostamento se non verso una lacuna, cioè una delle posizioni precedentemente occupate dai vertici estratti. Nel nostro caso, tra v_1 e v_2 non vi sarebbe una lacuna in grado di ospitare v_3.

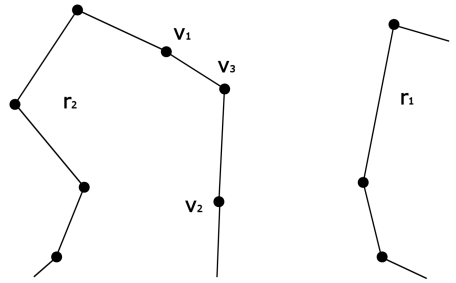
Da questa osservazione si può trarre ispirazione per proporre una prima estensione al metodo illustrato. Questa consiste nell'abbandonare il concetto di lacuna come unica posizione in cui consentire un inserimento; di fatto, si tratta di rompere la corrispondenza biunivoca tra lacune e vertici da reinserire. Conviene allora introdurre la nozione di *punto di inserzione*: una volta selezionati i vertici da rimuovere, questi vengono estratti dalla soluzione corrente che diventa così una soluzione *ristretta*, nel senso che visita solo i clienti rimasti fissi nelle loro posizioni perché non selezionati. Tutti i lati appartenenti alla soluzione ristretta, o un loro sottoinsieme, sono considerati poi possibili punti di inserzione in cui ricollocare, eventualmente, uno dei vertici estratti.

In questo modo, uno schema di selezione che rimuova v_3 lasciando contemporaneamente fissi v_1 e v_2 — per esempio quello che porta alla soluzione ristretta di figura 3.4a, in cui tutti i lati rimanenti, indicati in tratteggio, sono punti di inserzione — potrebbe portare alla soluzione riprodotta in figura 3.4b.

Nonostante questa prima estensione, esistono altri importanti limiti nella procedura. Si consideri ora la figura 3.5, che rappresenta un caso molto simile a quello esaminato in precedenza. Tralasciando sempre i vincoli di



(a) soluzione ristretta



(b) soluzione finale

Figura 3.4: Miglioramento della soluzione di figura 3.3.

capacità, si vede che i vertici v_3 e v_4 potrebbero essere più convenientemente visitati dal veicolo che percorre il *route* r_2 anzichè da quello assegnato al *route* r_1. Per far sì che lo spostamento congiunto dei due vertici possa avvenire non ci si può limitare, nella fase di scelta, a selezionare vertici isolati: è necessario consentire di selezionare per l'estrazione anche *sequenze* di vertici, estendendo ora anche il concetto di vertice estratto a quello di *sequenza estratta*. Nella figura 3.6a è visualizzata la soluzione ristretta ottenuta con uno schema di estrazione che lasci fissi i vertici v_1 e v_2, estraendo contemporaneamente i vertici vicini v_3 e v_4. Questi ultimi formano una sequenza: ciò è indicato esplicitamente racchiudendoli in una regione dal contorno punteggiato. La medesima notazione grafica si sarebbe potuta utilizzare anche per i vertici estratti isolati, che nella nuova nomenclatura corrispondono a sequenze estratte di dimensione 1. A questo punto le sequenze estratte devono essere ricollocate nei punti di inserzione in modo ottimale; nel caso specifico si può ottenere la configurazione di figura 3.6b.

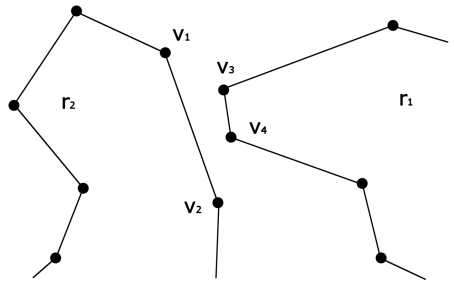
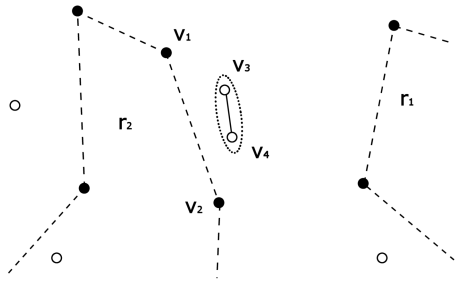


Figura 3.5: La collocazione dei vertici v_3 e v_4 non è ottimale.

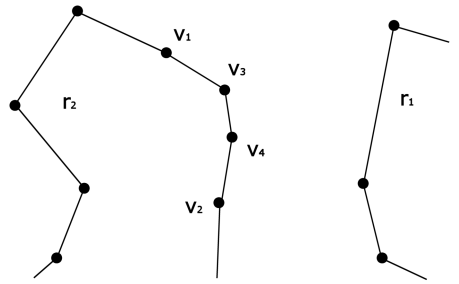
È importante evidenziare che il reinserimento di sequenze formate da più di un vertice comporta una difficoltà aggiuntiva: esistono infatti due modi per collocare una tale sequenza in un punto di inserzione, a seconda dell'orientamento prescelto. Questo aspetto può sembrare banale a fronte di una semplice osservazione della figura, tuttavia è necessario che l'algoritmo ne tenga conto.

Con riferimento al caso appena trattato, esiste un ultimo aspetto da considerare. Si è supposto, infatti, che lo schema di estrazione applicato rimuovesse i vertici v_3 e v_4 ma non v_1 e v_2; stando a quanto finora detto, se invece lo schema di estrazione applicato avesse selezionato anche v_1 o v_2 (figura 3.7) sarebbe stato impossibile pervenire, in un solo passo, alla soluzione di figura 3.6b. Il punto di inserzione I, infatti, potrebbe essere occupato da una sola delle sequenze $\langle v_1 \rangle$ o $\langle v_3, v_4 \rangle$. Questa può essere vista come una limitazione, e si può risolvere pensando di introdurre nel procedimento una fase di *generazione* di sequenze. A partire dalle sequenze di base estratte dalla soluzione corrente si tratta di costruire, secondo opportuni criteri, altre *sequenze derivate* formate dai vertici rimossi. Nel caso appena considerato, sarebbe utile che venisse costruita la sequenza $\langle v_1, v_3, v_4 \rangle$ e che si considerasse poi l'opportunità di collocarla nel punto di inserzione I. È anche prevista la possibilità di scomporre le sequenze estratte in sottosequenze: per esempio, da $\langle v_3, v_4 \rangle$ possono nascere $\langle v_3 \rangle$ e $\langle v_4 \rangle$; è però opportuno che tutte le sequenze di base vengano mantenute, in modo da poterle eventualmente riassegnare ai punti di inserzione da cui provengono e ritornare, così, alla soluzione precedente.

Naturalmente quest'ultima estensione complica notevolmente le cose. A seguito dell'estrazione delle sequenze di base e della costruzione di quelle derivate, ci si trova in presenza di un intero *pool* di sequenze. All'interno



(a) soluzione ristretta



(b) soluzione finale

Figura 3.6: Miglioramento della soluzione di figura 3.5.

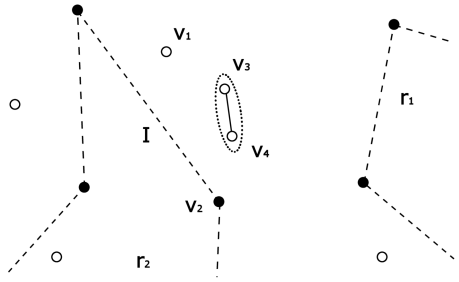


Figura 3.7: Un'estrazione che crea difficoltà.

di esso, un vertice estratto può essere rappresentato una o più volte in sequenze diverse; al momento della reinserzione, quindi, sarà necessario fare in modo che una sola delle sequenze contenenti ciascun vertice possa essere collocata.

Per concludere, va notato che l'implementazione di queste estensioni ha un prezzo non indifferente. Ci si è ormai allontanati molto dalla struttura del *neighborhood* ASSIGN, e in questo modo si rinuncia alla sua proprietà più interessante: la ricercabilità in tempo polinomiale.

3.5 L'algoritmo *SERR*

L'insieme di osservazioni finora esposte trova applicazione in un algoritmo per il VRP denominato *SERR — Selezione, Estrazione, Ricombinazione e Ricollocazione*. In questa sede ne verrà data una descrizione che tocca tutti gli aspetti fondamentali, rimandando alla successiva sezione 3.6 ogni approfondimento sui dettagli implementativi. La struttura generale presenta molti legami con quella dell'algoritmo *SR*, illustrato nella sezione 3.2.2.

Per il suo modo di operare, il metodo si può considerare membro della famiglia degli euristici migliorativi (si veda la sezione 2.2.3). Per questo motivo, si rende necessaria una fase di inizializzazione, nell'ambito della quale costruire una soluzione di partenza. Allo scopo si è scelto di sperimentare un euristico classico a due fasi basato sull'algoritmo proposto da Fisher e Jaikumar [23], già descritto nella sezione 2.2.2. La scelta è inizialmente ricaduta su questo metodo grazie ad alcune sue interessanti proprietà. In primo luogo, se il numero di veicoli disponibili K è almeno pari a K_{min} (si riveda la sezione 1.2.1), esso è sempre in grado di costruire una soluzione ammissibile utilizzando il numero richiesto di *route*; in secondo luogo, si tratta di un algoritmo in grado di fornire buoni risultati in tempi molto contenuti e risulta, infine, di facile implementazione. Esperimenti computazionali hanno tuttavia mostrato che le soluzioni iniziali ricavate con questo metodo sembrano ostacolare statisticamente gli scambi che l'algoritmo *SERR* tenta di effettuare; si può ipotizzare che questa difficoltà derivi dal fatto che l'algoritmo di Fisher e Jaikumar suddivide i clienti in *cluster* con una spartizione delle richieste molto omogenea. Nella pratica, l'algoritmo qui presentato pare essere molto più efficiente se applicato a partire da una soluzione iniziale di qualità anche non buona (20% sopra l'ottimo); per

ricavare una tale soluzione iniziale è stato implementato un metodo ispirato all'algoritmo *Sweep* (per una sua descrizione si veda la sezione 2.2.2).

Una seconda importante possibilità è quella di utilizzare, come soluzione iniziale, la soluzione prodotta da altri euristici o metaeuristici. Applicando il metodo a partire da soluzioni iniziali di buona qualità si ha un duplice vantaggio: oltre a limitare in maniera anche considerevole il tempo di calcolo, si aumenta la probabilità di giungere ad una soluzione più vicina all'ottimo. Grazie a questa modalità operativa è stato possibile ottenere soluzioni migliori di quelle note in letteratura per due problemi classici di *benchmark*[2].

Ad ogni iterazione è richiesto di scegliere ed applicare uno schema di selezione dei vertici da estrarre. Sono state individuate le seguenti possibilità:

- schema RANDOM-ALTERNATE, fortemente ispirato alla struttura del *neighborhood* ASSIGN, che prevede la selezione da ogni *route* dei vertici pari o dei vertici dispari: la scelta tra le due possibilità è effettuata, indipendentemente per ognuno dei *route*, in modo pseudo-casuale;
- schema SCATTERED, che prevede una estrazione uniforme con una probabilità del 40% per ogni vertice;
- schema NEIGHBORHOOD, secondo il quale viene definito e rimosso un vertice *pivot* e vengono selezionati i rimanenti con una probabilità decrescente in funzione della distanza dal vertice prescelto;

Per una descrizione dettagliata si rimanda, al solito, alla sezione 3.6.

La successiva fase prevede la costruzione, a partire dai vertici estratti, di un insieme di sequenze derivate. L'unione di queste ultime con le sequenze estratte dà luogo ad un insieme denominato *pool*. Le modalità di costruzione verranno esaminate nel dettaglio nella prossima sezione; ci si può qui limitare ad evidenziare due diverse possibilità. La prima consiste nel considerare ogni punto di inserzione e cercare di produrre delle sequenze che abbiano ragionevoli probabilità di esservi associate; la seconda possibilità, invece, prevede di costruire a partire dai vertici estratti delle "buone" sequenze per poi associarle eventualmente ad uno o più punti di inserzione.

Nella fase successiva, l'algoritmo deve risolvere il problema di riposizionare in maniera ottima un sottoinsieme delle sequenze del *pool*, in modo

[2]Si tratta di due delle ormai rarissime istanze di CVRP non ancora risolte all'ottimo con costi arrotondati.

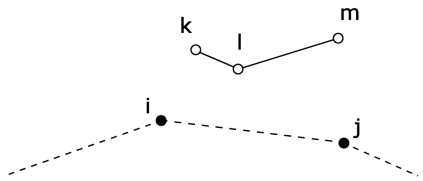


Figura 3.8: Esempio per la definizione dei costi delle sequenze e dei punti di inserzione.

che ognuno dei vertici estratti venga reintrodotto nella soluzione ristretta. A questo scopo è stato sviluppato un modello di programmazione lineare intera (PLI), che corrisponde ad un problema di set-covering con vincoli aggiuntivi. Per poterlo descrivere è ora necessario introdurre alcuni ulteriori concetti, con le rispettive notazioni, esemplificati in figura 3.8. Il primo di questi concetti è il costo di una sequenza, indicato con $c(s)$, pari alla somma dei costi dei lati che vi appartengono; nella figura, il costo della sequenza s, $c(s)$, è pari a $c_{kl} + c_{lm}$. In secondo luogo si considera il costo $c(i)$ di un punto di inserzione, pari semplicemente al costo del lato a cui corrisponde; nella figura, il costo $c(i)$ è pari a c_{ij}. Infine va introdotto il costo di concatenazione di una sequenza in un punto di inserzione: $c(s, i)$ è pari al costo complessivo dei due lati che devono essere introdotti. Nell'esempio, il costo $c(s, i)$ è pari a $c_{ik} + c_{mj}$. È bene precisare che ci si riferisce sempre al costo minimo di inserzione: ci sono sempre, infatti, due modi per concatenare la sequenza; nel caso illustrato l'altra possibilità ha un costo, evidentemente superiore, pari a $c_{im} + c_{kj}$. Infine, si indicano con $\mathcal{F}$ l'insieme dei vertici estratti, con $\mathcal{S}$ il *pool* di sequenze, con $\mathcal{I}$ l'insieme dei punti di inserzione e con $\mathcal{R}$ l'insieme dei *route* ristretti dopo l'estrazione. Per il resto valgono le notazioni introdotte nella sezione 1.2.1.

Detto questo, le variabili decisionali hanno il seguente significato:

$$x_{si} = \begin{cases} 1 & \text{se la sequenza } s \text{ viene collocata nel punto di inserzione } i \\ 0 & \text{altrimenti} \end{cases} \tag{3.1}$$

Il modello ha la seguente funzione obiettivo:

$$\min \sum_{s \in \mathcal{S}} \sum_{i \in \mathcal{I}} x_{si}[c(s) - c(i) + c(s, i)] \tag{3.2}$$

mentre i vincoli sono i seguenti:

$$\sum_{s \ni v} \sum_{i \in \mathcal{I}} x_{si} = 1 \ \ \forall v \in \mathcal{F} \tag{3.3}$$

$$\sum_{s \in \mathcal{S}} x_{si} \leq 1 \ \ \forall i \in \mathcal{I} \tag{3.4}$$

$$d(r) + \sum_{s \in \mathcal{S}} \sum_{i \in r} d(s) x_{si} \leq C \ \ \forall r \in \mathcal{R} \tag{3.5}$$

$$0 \leq x_{ij} \leq 1 \ \text{ intero} \tag{3.6}$$

La funzione obiettivo, da minimizzare, corrisponde alla somma di tutti i costi netti di inserzione delle sequenze che vengono ricollocate nei *route*. Ogni termine è pari al costo della sequenza sommato al costo di concatenazione meno il costo, risparmiato, del punto di inserzione. I vincoli (3.3) impongono che, per ciascuno dei vertici estratti, non più di una sequenza che lo contiene possa essere associata ad un qualche punto di inserzione. I vincoli (3.4) impediscono che un punto di inserzione possa ospitare più di una sequenza; infine, i vincoli (3.5), presenti nel numero di uno per ogni *route*, impongono il rispetto del limite di capacità dei veicoli.

Come si è visto, ognuna delle variabili del modello corrisponde al possibile accoppiamento tra una sequenza ed un punto di inserzione. Per evitare una proliferazione eccessiva di variabili, è necessario evitare sia di creare sequenze inutili, sia di associare indiscriminatamente sequenze a punti di inserzione quando l'accoppiamento non serve o ha scarse probabilità di essere impiegato. Questi obiettivi si conseguono grazie a due accorgimenti. Il primo consiste nel generare in maniera intelligente le sequenze derivate; il secondo, invece, prevede di creare questi accoppiamenti adottando una tecnica a generazione di colonne (*pricing*). In sostanza, vengono generate delle coppie sequenza/punto di inserzione, ma queste vengono introdotte nel modello solo se il *costo ridotto*, rispetto alla soluzione duale del rilassamento continuo, associato alla variabile che le rappresenta è minore di una certa soglia.

La risoluzione del modello può essere affidata a qualsiasi *software* di ottimizzazione; nel nostro caso, è stato utilizzato il prodotto ILOG Cplex nella sua versione 8. Al termine del lavoro dell'ottimizzatore, una nuova soluzione viene costruita sulla base della soluzione ristretta precedente e del

valore assunto dalle variabili decisionali. Per avere la garanzia che il costo della nuova soluzione non superi quello della precedente, vengono fornite al modello le coppie candidate formate dalle sequenze estratte e dai punti di inserzione da cui provengono: così facendo si beneficia della proprietà di *non peggiorativit*à.

Se dopo il riassemblaggio il costo risulta diminuito rispetto alla precedente soluzione, è opportuno riottimizzare singolarmente ogni *route* tramite un metodo classico, come per esempio *3-opt*. Questo *modus operandi* è comune a molti euristici e metaeuristici.

Riassumendo, l'algoritmo può essere schematizzato come segue.

Algoritmo *SERR*

(i) (*Inizializzazione*). Tramite l'utilizzo di un euristico veloce, si risolve l'istanza di VRP in modo da ottenere una soluzione di partenza per l'applicazione della successiva fase migliorativa.

(ii) (*Selezione*). Si sceglie, in base ad opportuni criteri, uno degli schemi di selezione previsti.

(iii) (*Estrazione*). Si estraggono i vertici selezionati al punto precedente, formando le sequenze estratte di base. Contemporaneamente, a partire dalla soluzione corrente, si produce la soluzione ristretta e si designa come possibile punto di inserzione ognuno dei suoi lati. Vengono aggiunte al modello le variabili decisionali corrispondenti all'accoppiamento tra ogni sequenza di base ed il suo punto di inserzione originario, in modo che la risoluzione del modello di PLI possa portare nuovamente alla soluzione precedente (*non peggiorativit*à).

(iv) (*Ricombinazione*). A partire dai vertici estratti, considerati singolarmente, si costruisce un insieme di sequenze derivate, che assieme alle sequenze di base formeranno il *pool*. Ognuna delle sequenze del *pool* viene associata ad uno o più punti di inserzione candidati ad ospitarla, creando le corrispondenti variabili decisionali; in questa fase si usa un approccio a generazione di colonne (*pricing*).

(v) (*Ricollocazione*). Si risolve il modello di PLI che permette di ricollocare un sottoinsieme delle sequenze del *pool*, che rappre-

sentino una ed una sola volta tutti i vertici estratti, in maniera ottimale. Esaminando i valori delle variabili decisionali, si costruisce quindi una nuova soluzione corrente; se quest'ultima risulta migliore di quella del passo precedente, si applica ad ogni *route* una serie di passi migliorativi *3-opt*.

(vi) (*Valutazione*). Se durante le ultime n iterazioni, essendo n il numero dei clienti, vi è stato almeno un miglioramento, si ritorna al punto (ii); in caso contrario, l'algoritmo termina.

3.6 Dettagli implementativi

Allo scopo di favorire la leggibilità, alcuni aspetti particolari relativi all'implementazione dell'algoritmo *SERR* sono stati finora omessi. In questa sezione si commentano i dettagli implementativi più importanti, in modo da completare in maniera esauriente la trattazione.

3.6.1 Inizializzazione

Lo scopo della fase di inizializzazione è costruire una soluzione di partenza sulla quale iterare il procedimento migliorativo. Questo compito è stato affidato ad un semplicissimo algoritmo ispirato al metodo *sweep*, già descritto nella sezione 2.2.2: si tratta di suddividere i clienti in K *cluster* secondo l'angolo formato, rispetto ad un riferimento fisso, dal segmento che li unisce al deposito, in modo che la richiesta totale dei vertici di ogni *cluster* sia compatibile con la capacità massima dei veicoli. Una volta individuati i *cluster*, ognuno dei *route* viene costruito con un algoritmo *nearest-neighbor* seguito poi da un miglioramento *3-opt*.

Le soluzioni individuate in questo modo sono spesso di scarsa qualità; esperimenti computazionali hanno però dimostrato l'estrema validità del metodo *SERR* quando applicato a partire da esse, evidenziando anche che ogni sforzo intrapreso per abbassare il costo della soluzione iniziale pare ripercuotersi negativamente sul risultato finale. Va osservato, in ogni caso, che già dalle primissime iterazioni la qualità delle soluzioni intermedie migliora notevolmente.

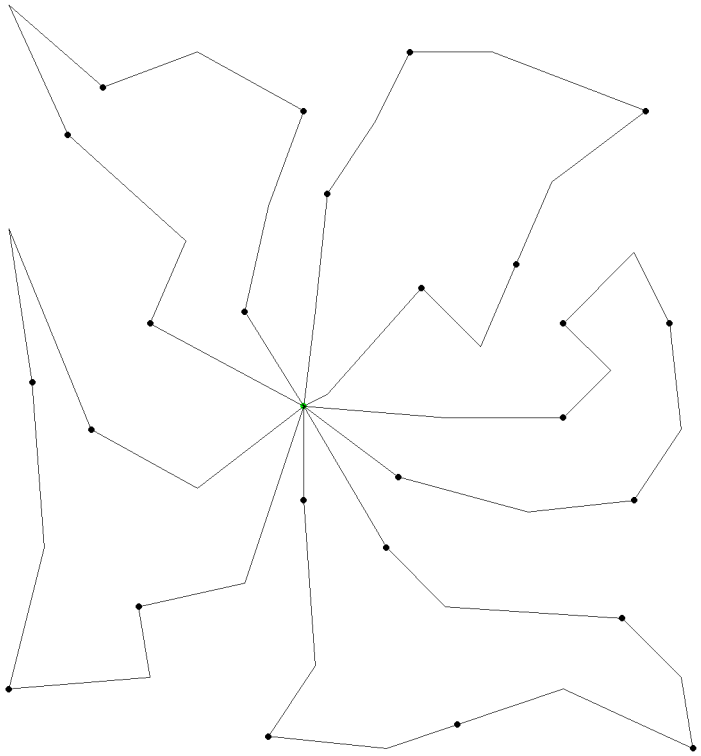


Figura 3.9: Estrazione RANDOM-ALTERNATE. I vertici selezionati sono contrassegnati da punti neri.

3.6.2 Schemi di selezione

Come si è accennato in precedenza, sono stati individuati tre possibili schemi di selezione dei vertici da estrarre. Ognuno di essi viene qui presentato in dettaglio, mentre in seguito si descriverà il criterio seguito dall'algoritmo *SERR* per la scelta dello schema da applicare ad ogni iterazione.

Schema Random-Alternate

Questo schema si ispira alla struttura del *neighborhood* ASSIGN per il TSP, ed offre un perfetto bilanciamento tra il numero di vertici estratti ed il numero di punti di inserzione. Per ogni *route*, considerato indipendentemente, si determina in modo pseudocasuale se rimuovere i vertici in posizione pari o quelli in posizione dispari: il risultato è una perfetta alternanza tra vertici estratti e vertici fissi (figura 3.9). Ad ogni applicazione dello schema, tuttavia, i possibili scambi di vertici tra *route* sono diversi proprio grazie al criterio di pseudocasualità.

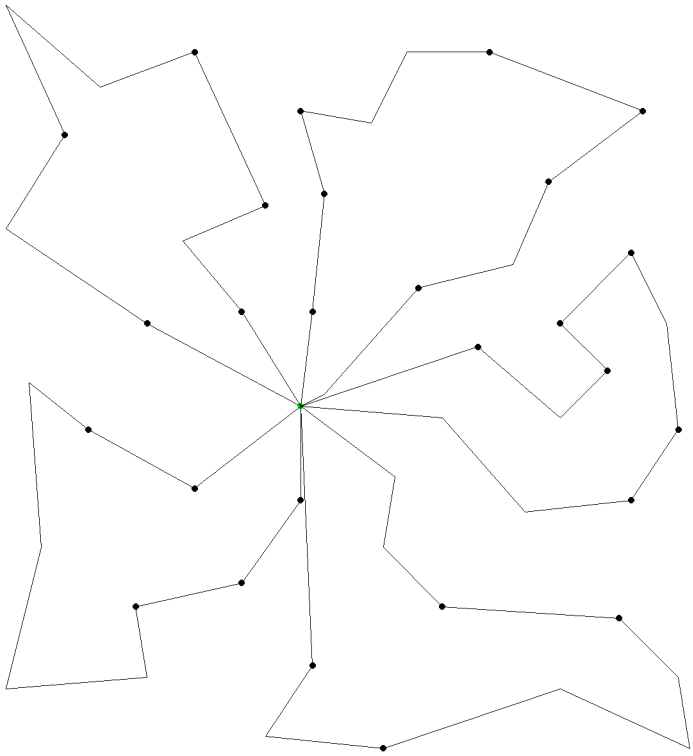


Figura 3.10: Estrazione SCATTERED. I punti neri rappresentano i vertici selezionati.

Schema Scattered

Lo schema SCATTERED cerca di ottenere, in modo diverso dal precedente, un buon bilanciamento tra il numero di vertici estratti ed il numero di punti di inserzione. Ogni vertice può essere estratto con una probabilità del 50%, senza i vincoli di alternanza propri dello schema RANDOM-ALTERNATE. Questo criterio può quindi portare alla rimozione di sequenze formate da più vertici consecutivi (figura 3.10).

Schema Neighborhood

Mentre i due schemi precedenti intendono favorire la ridistribuzione dei vertici tra tutti i *route*, NEIGHBORHOOD consente di focalizzare l'attenzione su una regione specifica per agevolare un eventuale miglioramento localizzato. Una volta scelto un determinato vertice, detto *pivot*, tutti i clienti dell'istanza vengono disposti in ordine di distanza crescente rispetto al *pivot*; quest'ultimo si considera incluso nella lista, e si trova inevitabilmente in

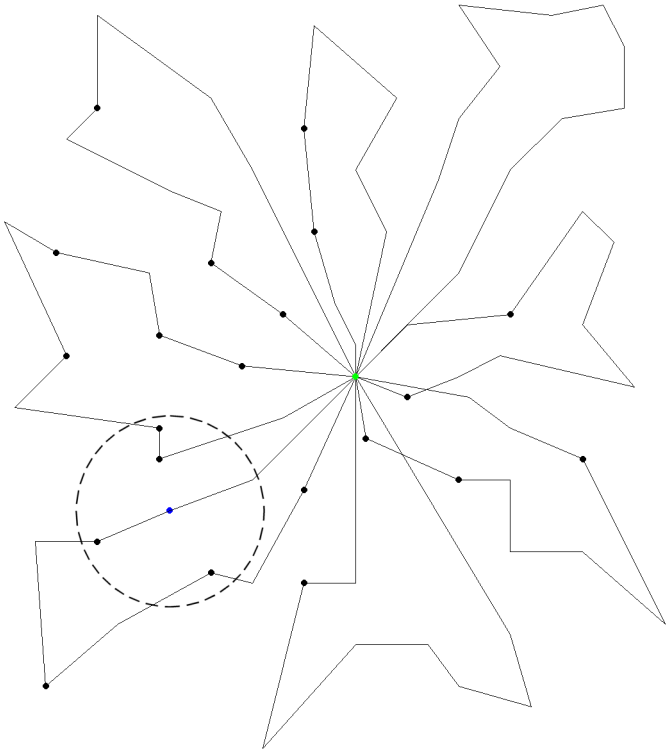


Figura 3.11: Estrazione con schema NEIGHBORHOOD. I punti neri corrispondono ai vertici selezionati, mentre la zona evidenziata contiene il vertice *pivot* e i suoi quattro vicini più immediati.

prima posizione. A questo punto, i primi cinque vertici vengono selezionati per l'estrazione; i rimanenti vengono suddivisi in cinque fasce, equipopolate, di distanza. A seconda della fascia di appartenenza, gli altri vertici vengono estratti con una probabilità pari all'80% per la prima fascia, al 60% per la seconda, al 30% per la terza, al 20% per la quarta ed, infine, al 10% per la quinta. In questo modo la probabilità di estrazione si fa sempre più bassa man mano che ci si allontana dal vertice *pivot* (figura 3.11).

3.6.3 Scelta dello schema

Ad ogni iterazione l'algoritmo *SERR* sceglie uno schema di selezione da applicare. Il criterio di scelta si basa sulle seguenti ipotesi:

- gli schemi RANDOM-ALTERNATE e SCATTERED paiono essere molto indicati per il miglioramento della soluzione iniziale;

- quando la qualità delle soluzioni intermedie aumenta[3], sembra essere più indicato lo schema NEIGHBORHOOD, che cerca di effettuare miglioramenti localizzati.

In virtù di queste osservazioni, il criterio ritenuto più efficace è il seguente:

- 3 ripetizioni dello schema RANDOM-ALTERNATE;
- 3 ripetizioni dello schema SCATTERED;
- ripetizione ad oltranza dello schema NEIGHBORHOOD fino ad un eventuale miglioramento o al verificarsi della condizione di arresto.

Si ricorda che la condizione di arresto prevede il trascorrimento di n iterazioni senza che abbia luogo alcun miglioramento. Quando invece si ha una diminuzione di costo tra una soluzione e la successiva, la sequenza viene riapplicata dall'inizio, ossia a partire dalle tre estrazioni RANDOM-ALTERNATE. Così facendo, nelle prime iterazioni sono applicati molte volte gli schemi RANDOM-ALTERNATE e SCATTERED, ritenuti più adatti; successivamente, quando questi ultimi diventano inefficaci, viene utilizzato in prevalenza lo schema NEIGHBORHOOD.

La scelta del vertice *pivot* per l'estrazione NEIGHBORHOOD è effettuata con l'intento di perseguire due obiettivi: fare in modo che i primi vertici scelti diano luogo ad una maggiore probabilità di miglioramento e garantire una frequenza di scelta uniforme su tutti i vertici. Per ottenere questo, in occasione della prima estrazione NEIGHBORHOOD ad ogni vertice viene assegnato un punteggio, tanto più alto quanti più *route* si trovano nelle vicinanze e quanto minore è la distanza media dei dieci vertici più vicini. Ad ogni iterazione con estrazione NEIGHBORHOOD viene quindi scelto come vertice *pivot* il successivo nella lista ordinata per punteggio decrescente; ogni qualvolta il vertice con il punteggio più basso viene scelto, e cioè dopo esattamente n estrazioni NEIGHBORHOOD dall'ultima applicazione del criterio di ordinamento, la lista viene aggiornata e si riparte dalla sua cima.

3.6.4 Ricombinazione

La fase di ricombinazione è probabilmente il punto più delicato dell'algoritmo: l'efficacia della strategia adottata è importante sia per i risultati,

[3]Si ricorda che ogni soluzione ottenuta ha costo non superiore alla precedente (*non peggiorativit*à).

sia per il tempo di calcolo. La produzione mal controllata di coppie sequenza/punto di inserzione, infatti, può far lievitare inutilmente il numero di variabili del problema di *set-covering* con vincoli aggiuntivi riducendo, allo stesso tempo, le probabilità che il reinserimento avvenga nel modo più efficace possibile[4].

La produzione di sequenze derivate avviene in due fasi. L'obiettivo di ciascuna delle due fasi è generare un certo numero di variabili che corrispondano ad accoppiamenti tra sequenze e punti di inserzione con buoni requisiti per essere selezionati. La valutazione quantitativa di questi requisiti avviene mediante il concetto di *costo ridotto*, e l'approccio utilizzato è noto come *pricing* o *generazione di colonne*. Inizialmente il modello viene generato senza i vincoli di interezza, e popolato da un piccolo numero di variabili: vengono considerati solo gli accoppiamenti tra le sequenze estratte e i punti di inserzione da cui provengono. Se venissero aggiunti i vincoli di interezza, il modello così ottenuto ammetterebbe un'unica soluzione valida, che consentirebbe di ricostruire la situazione precedente.

Prima delle due fasi di produzione, e solo quando la selezione è stata effettuata con lo schema NEIGHBORHOOD, un ulteriore gruppo di variabili viene aggiunto al modello: esse rappresentano accoppiamenti tra sequenze formate dal vertice *pivot* e dai suoi immediati vicini estratti e punti di inserzione che si trovano in prossimità del vertice *pivot* stesso. Dopo l'estrazione di quest'ultimo con i suoi quattro vertici più vicini, vengono costruite tutte le sequenze di lunghezza da 1 a 5 da essi composte. Tutti i punti di inserzione che ospitavano sequenze contenenti uno di questi cinque vertici vengono associati ad ognuna delle sequenze così create. In questo modo si vuole aumentare il più possibile la probabilità che un miglioramento locale nella zona del vertice *pivot* abbia luogo.

Ha quindi inizio la prima fase di *pricing*, che ha come scopo quello di considerare ogni punto di inserzione e costruire un certo numero di sequenze con buona attitudine ad esservi inserite. Considerato il generico punto di inserzione i, viene seguito il seguente procedimento:

(i) si inizializza un insieme di sequenze $S = \varnothing$;

(ii) si generano tutte le sequenze ottenibili aggiungendo in tutti i possibili modi uno dei vertici estratti ad ognuna delle sequenze in S;

[4]Per motivi che saranno chiariti successivamente, il numero di nodi dell'albero decisionale viene infatti limitato.

(iii) ad ognuna delle sequenze così ottenute viene associato un costo, $c_i(s)$; se la lunghezza della sequenza s è 1 questo costo è pari a $c(s, i)$ (secondo la simbologia introdotta nella sezione 3.5), altrimenti $c_i(s)$ è pari al costo ridotto che la variabile x_{is} avrebbe rispetto alla soluzione duale corrente del rilassamento continuo del modello che si sta popolando;

(iv) le N_{min} sequenze con costo $c_i(s)$ minore vengono associate al punto di inserzione i, e la variabile x_{si} corrispondente viene aggiunta al modello; se la lunghezza delle sequenze è maggiore di 1, vengono aggiunte al modello al massimo altre $N_{max} - N_{min}$ sequenze se il loro costo ridotto è minore di una soglia RC_{max}. Si ridefinisce S come l'insieme di tutte le sequenze appena aggiunte al modello; S conterrà, quindi, al massimo N_{max} elementi;

(v) il rilassamento continuo con le variabili appena aggiunte viene nuovamente risolto per mezzo dell'algoritmo del simplesso duale;

(vi) se le ultime sequenze aggiunte hanno lunghezza pari ad L_{max}, la prima fase di generazione di sequenze derivate termina; in caso contrario, si ritorna al punto (ii).

È utile soffermarsi sul diverso trattamento riservato alle sequenze di lunghezza 1. Queste ultime, infatti, vengono introdotte nel modello non in base al costo ridotto della variabile che le associa al punto di inserzione i, ma in base al costo di inserzione $c(s, i)$: la motivazione riguarda l'affidabilità del valore di costo ridotto. Se il problema di base contiene troppo poche variabili, il costo ridotto valutato per una nuova variabile candidata può mal rappresentare l'attitudine di quest'ultima ad abbassare il valore della funzione obiettivo. Per questo motivo, è preferibile valutare le prime variabili generate — quelle con sequenze di lunghezza 1, appunto — calcolandone il costo effettivo di inserzione anzichè il costo ridotto; dopo questi primi inserimenti, il modello risulta sufficientemente popolato ed è possibile basarsi sui costi ridotti per tutte le successive valutazioni.

I parametri che caratterizzano questa prima fase sono dunque N_{min}, N_{max}, RC_{max}, L_{max}. I risultati computazionali riportati nella sezione 3.7 sono stati ottenuti ponendo $N_{min}^{(1)} = 15$ per le sequenze di lunghezza 1,

$N_{min} = 10$ per tutte le altre sequenze, $N_{max} = 20$, $L_{max} = 6$ e $RC_{max} = 10$. In particolare, RC_{max} è strettamente positivo in quanto si stanno aggiungendo variabili che serviranno a risolvere un problema di programmazione lineare intera; la teoria assicura che la loro aggiunta non può essere in alcun modo utile per migliorare la soluzione del rilassamento continuo, ma quando si inserisce il vincolo di interezza la presenza di queste variabili può invece rivelarsi utile.

La seconda fase di generazione mira ad integrare l'insieme delle variabili già aggiunte al modello fornendo ulteriori possibili accoppiamenti, costruiti stavolta generando nuove sequenze e individuando eventuali punti di inserzione in cui potrebbero essere collocate. Il procedimento si snoda come segue:

(i) considerate tutte le sequenze estratte, tutte quelle eventualmente costruite a partire dal vertice *pivot* e dai suoi quattro vicini e tutte le sequenze possibili formate da uno solo dei vertici estratti[5], si aggiungono al modello le variabili che corrispondono agli accoppiamenti con i punti di inserzione quando il costo ridotto è minore di RC_{max};

(ii) considerata ognuna delle possibili sequenze formate da due dei vertici estratti, le variabili associate al loro accoppiamento con ognuno dei punti di inserzione vengono inserite nel modello se il loro costo ridotto è minore di RC_{max};

(iii) considerate tutte le coppie sequenza/punto di inserzione (s, i) che hanno formato variabili inserite nel modello al punto (ii), si cerca di estendere la sequenza s, di lunghezza 2, in una di lunghezza 3 aggiungendovi in tutti i modi possibili uno dei vertici liberi; tutte le variabili associate alle coppie ottenute (s', i) vengono poi valutate ed inserite nel modello se il loro costo ridotto è inferiore a RC_{max};

(iv) le fasi (i)–(iii) vengono reiterate finchè nessuna ulteriore variabile viene aggiunta al modello.

La completa enumerazione delle possibili sequenze di lunghezza 2 fa sì che questa seconda fase possa rilevare buone quantità di variabili aggiuntive

[5] I due insiemi possono essere non disgiunti.

utili. Durante il procedimento sono utilizzate diverse strutture *hash* per garantire che non siano inserite nel modello variabili corrispondenti alle stesse coppie sequenza/punto di inserzione. Come ulteriore accorgimento, quando il posizionamento di una sequenza in un punto di inserzione porterebbe a violazioni dei vincoli di capacità, la corrispondente variabile non viene nemmeno generata.

3.6.5 Reinserimento

Nella sezione 3.5 è stato descritto il modello di programmazione lineare intera la cui risoluzione consente, ad ogni iterazione, di ricollocare nella soluzione ristretta tutti i vertici in precedenza estratti. Come si è già osservato, il modello corrisponde ad un problema di *set-covering* con vincoli aggiuntivi: si tratta dunque di un problema difficile da trattare, la cui risoluzione può richiedere tempi non brevi. Considerata la condizione di arresto, che prevede che non vi siano stati miglioramenti nelle ultime n iterazioni, è evidente che la fase di reinserimento è destinata ad essere effettuata molte volte. Per cercare di limitare il tempo di calcolo totale, quindi, è opportuno configurare il risolutore in modo che ad ogni iterazione:

- il problema di *set-covering* non venga risolto all'ottimo se questo richiede un tempo troppo elevato;
- nel caso in cui il procedimento di risoluzione venga interrotto prima di giungere all'ottimo, sia favorito il raggiungimento di una soluzione di costo comunque inferiore alla precedente.

Nell'implementazione si è tenuto conto di queste necessità impostando adeguatamente alcuni parametri del risolutore utilizzato, ILOG Cplex 8.0. In particolare, si è imposto il limite massimo di 30000 nodi nell'albero di ricerca e si è data maggiore enfasi all'ottenimento di soluzioni ammissibili piuttosto che puntare direttamente alla soluzione ottima, che potrebbe essere difficile ricavare in virtù del suddetto limite. Come ulteriore ausilio, è sempre fornita una soluzione iniziale, corrispondente al reinserimento delle sequenze appena estratte nel loro corrispondente punto di inserzione: in questo modo l'ottimizzatore dispone di un buon valore di *bound* che consente di rendere più efficiente e rapido il suo lavoro.

Problema	Sol. ottima	M. prec.	Sol. SERR	Gap	Tempo
P-n50-k8	**631** (2:50.51)	649	**631**	0.00%	11:08
P-n55-k10	**694** (18:03.56)	696	700	0.86%	16:50
P-n60-k10	**744** (41.11)	756	**744**	0.00%	25:01
P-n60-k15	**968** (5:40.54)	1033	975	0.72%	12:27
P-n70-k10	**827** (114:48.16)	834	831	0.48%	50:08
B-n68-k9	**1272** (158:06.52)	1275	1275	0.24%	3:02:01

Tabella 3.1: Istanze con costi arrotondati e con soluzione ottima individuata per la prima volta in Fukasawa et al. [25].

3.7 Risultati su istanze di *benchmark*

I test condotti riguardano un certo numero di istanze di particolare interesse, con costi reali od arrotondati all'intero più vicino; i tempi di calcolo sono riportati nel formato [hh:]mm:ss. L'hardware di riferimento è un PC AMD Athlon XP 2400+, 512 Mb di RAM; l'ambiente operativo è Linux RH9, il risolutore è ILOG Cplex 8.0. L'algoritmo è implementato in C++ e sfrutta l'interfaccia ILOG Concert Technology 1.2; il compilatore utilizzato è GNU GCC 3.0.4 con le librerie GNU GLIBC 2.2.

Un primo blocco di istanze (tabella 3.1) raggruppa problemi la cui nuova soluzione ottima è stata identificata nel recentissimo lavoro di Fukasawa *et al.* [25], un resoconto tecnico non ancora pubblicato su un nuovo algoritmo esatto di tipo *branch-and-cut-and-price*. La tabella riporta il valore della recente soluzione ottima (*Sol. ottima*), accompagnato dal tempo di calcolo necessario ad individuarlo[6] ed il valore della migliore soluzione nota in precedenza (*M. prec.*), così come indicato nel suddetto rapporto tecnico; per quanto riguarda la soluzione individuata da *SERR*, sono riportati il costo (*Sol. SERR*), lo scostamento percentuale di quest'ultimo dal costo ottimo (*Gap*) e il tempo di calcolo impiegato (*Tempo*). In cinque casi su sei l'algoritmo *SERR* ha individuato una soluzione con costo minore o uguale a quello della migliore nota in precedenza, proponendosi quindi come miglior euristico finora applicato a queste istanze; in due di questi cinque casi la soluzione è anche ottima. I tempi di calcolo sono nettamente più bassi rispetto a quelli richiesti dal nuovo algoritmo esatto.

Un secondo blocco molto interessante è formato da istanze non ancora risolte all'ottimo, ma di cui è di dominio pubblico almeno una buona solu-

[6] La macchina utilizzata negli esperimenti del lavoro citato è un Intel Pentium IV 2 GHz.

Problema	Migl. sol. nota	Sol. iniz.	Sol. SERR	Tempo
E-n101-k14	1071	1076 (X)	**1067**	1:36:05
M-n151-k12-a	1023	1023 (G)	**1022**	7:46:33
M-n200-k17	1294	1294 (G)	1294	27:07:01

Tabella 3.2: Istanze con costi arrotondati, con soluzione ottima non nota e con una buona soluzione disponibile. (G) = Gendreau, Hertz e Laporte, 1993. (X) = Xu e Kelly, 1996.

Problema	Sol. ottima	Sol. SERR	Gap	Tempo
E-n51-k5	**521**	**521**	0.00%	4:30
E-n76-k7	**682**	**682**	0.00%	27:35
E-n76-k8	**735**	742	0.95%	30:39
E-n76-k10	**830**	835	0.60%	1:19:30
E-n76-k14	**1021**	1032	1.08%	2:45:20
E-n101-k8	**815**	820	0.61%	2:54:04
P-n65-k10	**792**	796	0.51%	12:26
E051-05e	**524.61**	**524.61**	0.00%	4:51
E076-10e	**835.26**	835.32	$< 0.01\%$	1:12:05
E101-08e	**826.14**	831.91	0.70%	2:30:55
E101-10c	**819.56**	**819.56**	0.00%	2:35:36

Tabella 3.3: Istanze con costi arrotondati e con soluzione ottima nota.

zione (tabella 3.2). Si tratta di un gruppo poco nutrito, anche in virtù del fatto che il numero di istanze ancora aperte è molto limitato. In due casi su tre l'algoritmo *SERR* ha migliorato la soluzione di partenza, ottenendo la nuova migliore soluzione disponibile. La colonna *Sol. iniz.* riporta il costo della soluzione iniziale adottata, mentre il valore della precedente migliore soluzione nota è riportato nella colonna *Migl. sol. nota*.

Il terzo, ed ultimo, gruppo di istanze (tabella 3.3) raccoglie alcuni problemi, con costi arrotondati o reali, aventi soluzione ottima nota. Anche in questi casi si può apprezzare la tendenza dell'algoritmo *SERR* ad avvicinarsi notevolmente all'ottimo, raggiunto comunque in quattro casi su undici.

Una valutazione qualitativa delle prestazioni dell'algoritmo proposto può essere data commentando le seguenti citazioni, da due recenti lavori di rassegna di Laporte e Semet e di Gendreau, Laporte e Potvin sugli algoritmi euristici e metaeuristici per il VRP:

"Because metaheuristics for the CVRP outperform classical

> methods in terms of solution quality (and sometimes now in terms of computing time), we believe there is little room left for significant improvement in the area of classical heuristics." [41, p. 125].
>
> "With respect to the classical heuristics [...] metaheuristics are rather time consuming, but they also provide much better solutions. Tipically, classical methods yield solution values between 2% and 10% above the optimum (or the best known solution value), while the corresponding figure for the best metaheuristic implementation is often less than 0.5%." [29, p. 148].

L'assenza di qualsiasi tipo di sovrastruttura metaeuristica posiziona l'algoritmo *SERR* nella categoria degli euristici classici[7]: alla luce di queste considerazioni si deve quindi interpretare l'eccellente scarto medio dall'ottimo, pari allo 0.40% per il campione scelto, come una precisa indicazione sull'efficacia del metodo qui presentato.

Questa efficacia comporta lo sfruttamento di molte risorse computazionali, al punto che i tempi di calcolo richiesti per l'arresto autonomo dell'algoritmo superano di molto quelli tipici degli altri euristici. A questo riguardo si può però proporre una interessante osservazione. Il grafico di figura 3.12 rappresenta, per tre processi di risoluzione di altrettante istanze con dimensioni paragonabili, l'andamento temporale dello scarto percentuale sopra al costo ottimo. Pur partendo, deliberatamente[8], da una soluzione iniziale di scarsa qualità, già nelle prime iterazioni l'algoritmo si porta ad un livello corrispondente a quello di un ottimo euristico. La fase successiva presenta una convergenza più lenta, ma in molti casi pratici questo non rappresenta un problema, non essendo necessario concludere la computazione in tempi brevissimi. Vale anche la pena osservare che l'eventuale utilizzo di soluzioni iniziali di maggiore qualità, come quelle ottenute tramite il citato algoritmo di Fisher e Jaikumar, influisce molto positivamente sul costo delle soluzioni individuate anche nelle primissime iterazioni. Come si è già osservato, l'effetto sul costo finale sembra purtroppo essere opposto.

[7] Come già osservato nel capitolo 2, per metaeuristico si intende uno schema di controllo per un euristico classico che tenta di superare gli ottimi locali mediante una serie controllata di iterazioni peggiorative del costo della soluzione corrente.

[8] L'argomento è stato affrontato nella sezione 3.5.

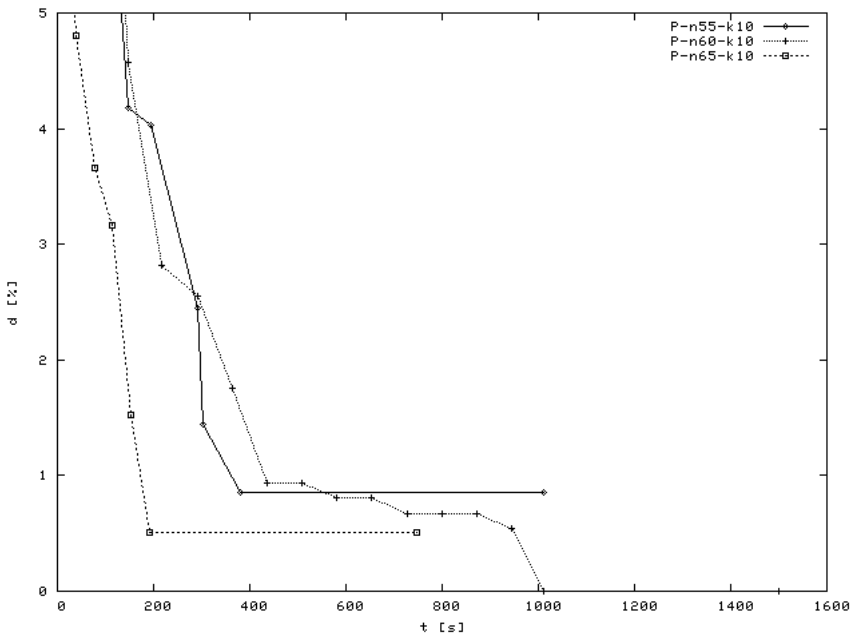


Figura 3.12: andamento nel tempo dello scarto percentuale dall'ottimo per tre istanze campione.

3.8 Conclusioni

I risultati ottenuti con l'algoritmo *SERR* sono senza dubbio molto positivi. Oltre a manifestare buona attitudine a rilevare autonomamente soluzioni di elevata qualità, con uno scarto medio sopra il costo ottimo pari allo 0.40% e con un buon numero di soluzioni ottime raggiunte nel campione, l'algoritmo si propone anche come valida procedura migliorativa, in grado di manipolare soluzioni di qualità già molto buona ricavate con altri metodi: in due dei tre esperimenti effettuati è stato possibile migliorare la soluzione ottenuta da eccellenti metaeuristici, ottenendo così le nuove migliori soluzioni disponibili.

Futuri sviluppi potrebbero portare all'abbassamento dei tempi di calcolo agendo sia sulla soluzione iniziale, sia sulle fasi di selezione e ricombinazione. Una possibile, interessante estensione consiste nel permettere un lieve rilassamento dei vincoli relativi alla capacità dei veicoli: ciò consentirebbe di allargare l'esplorazione ad una porzione più estesa dello spazio delle soluzioni. Il futuro dell'algoritmo *SERR* comprenderà anche adattamenti a versioni più vincolate di VRP, come *Distance-Constrained CVRP* (*DCVRP*) e, in particolare *Precedence-Constrained VRP*.

3.9 Soluzioni di interesse

In chiusura, trovano spazio le raffigurazioni delle nuove migliori soluzioni euristiche individuate dall'algoritmo *SERR*, ed in particolare le nuove migliori soluzioni note per le istanze E-n101-k8 e M-n151-k12-a con costi arrotondati.

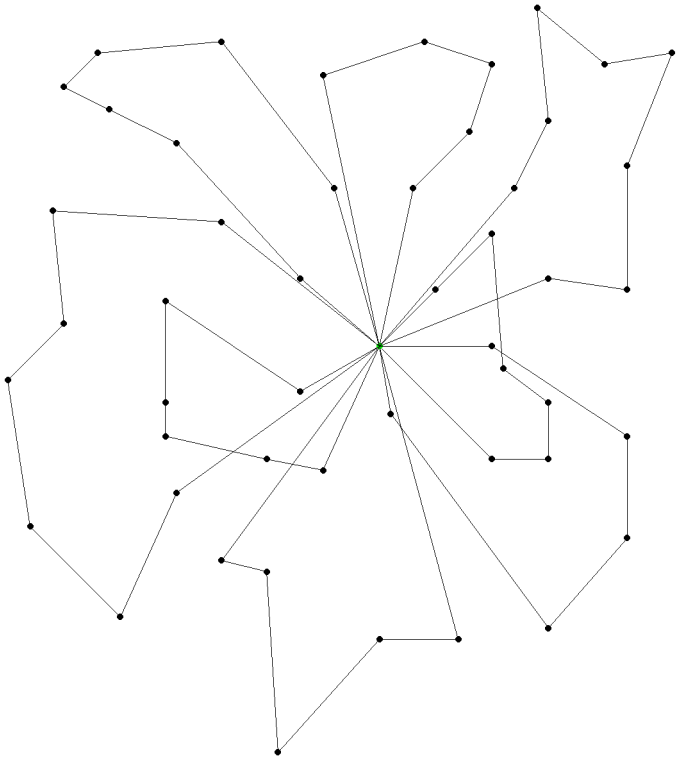


Figura 3.13: Nuova migliore soluzione euristica per l'istanza P-n50-k8, di costo 631 (poi dimostrata ottima in [25]).

```
R1:  4, 45, 46, 8, 35, 7
R2:  26, 11, 14, 19, 34
R3:  38, 10, 31, 39, 9
R4:  17, 16, 3, 44, 40, 12
R5:  32, 25, 18, 24, 49, 23, 33
R6:  6, 1, 43, 41, 42, 22, 2
R7:  30, 48, 47, 21, 28
R8:  29, 5, 36, 37, 20, 15, 13, 27
```

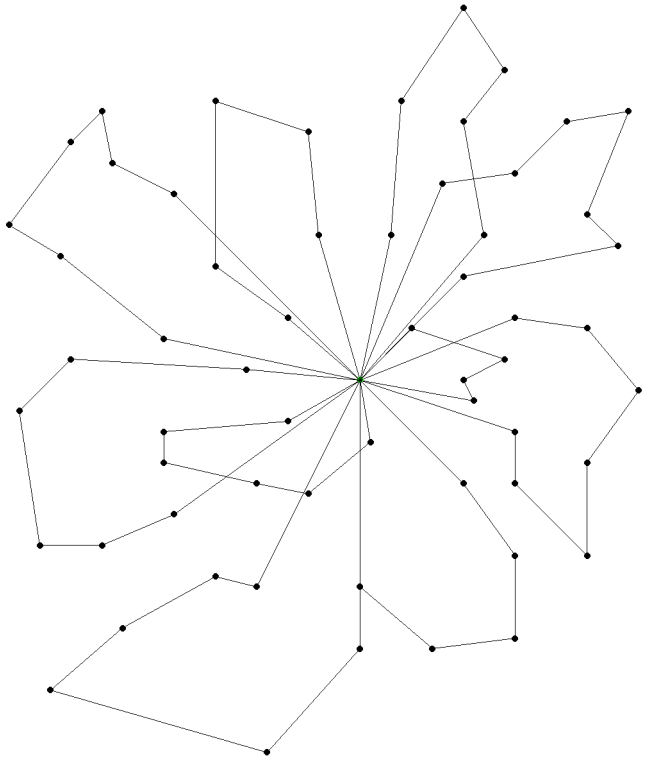


Figura 3.14: Nuova migliore soluzione euristica per l'istanza P-n60-k10, di costo 744 (poi dimostrata ottima in [25]).

```
R1:  29, 47, 36, 59, 21, 30
R2:  48, 5, 37, 20, 15, 57, 45
R3:  2, 28, 22, 33, 6
R4:  8, 35, 14, 19, 54, 13, 27
R5:  7, 53, 11, 38, 58
R6:  39, 9, 25, 55, 31, 10
R7:  32, 50, 18, 24, 49, 51
R8:  17, 3, 44, 40, 12, 26
R9:  1, 43, 42, 41, 56, 23, 16
R10:  4, 52, 34, 46
```

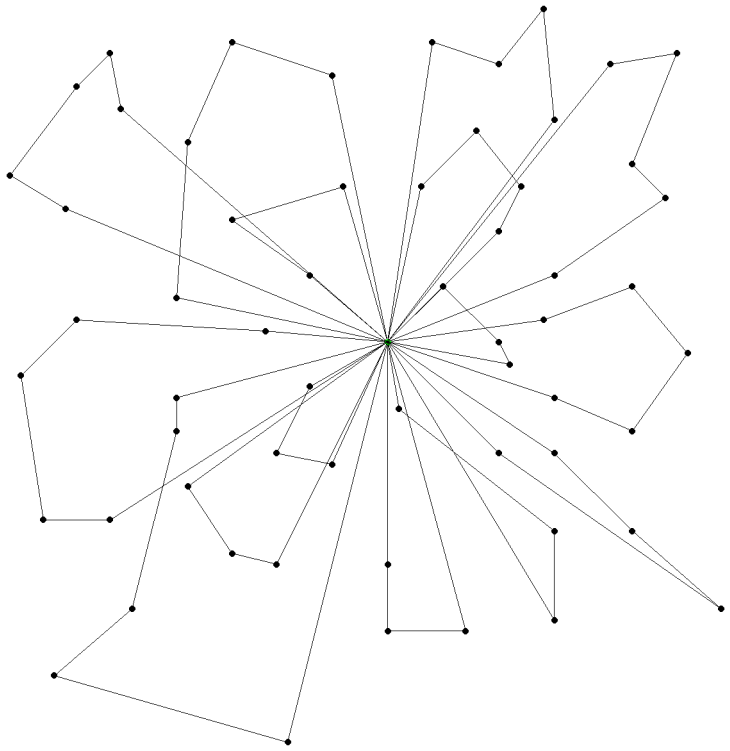


Figura 3.15: Nuova migliore soluzione euristica per l'istanza P-n60-k15, di costo 975.

```
R1:  5, 36, 47, 21
R2:  37, 20, 15, 57, 27
R3:  46, 34, 4
R4:  8, 19, 54, 13, 52
R5:  45, 29, 48, 30
R6:  7, 59, 14, 35
R7:  26, 53, 11
R8:  58, 10, 38
R9:  50, 18, 24, 49, 51
R10:  32, 9, 39
R11:  3, 44, 25, 55, 31
R12:  17, 40, 12
R13:  6, 33, 2
R14:  43, 42, 41, 56, 23
R15:  16, 1, 22, 28
```

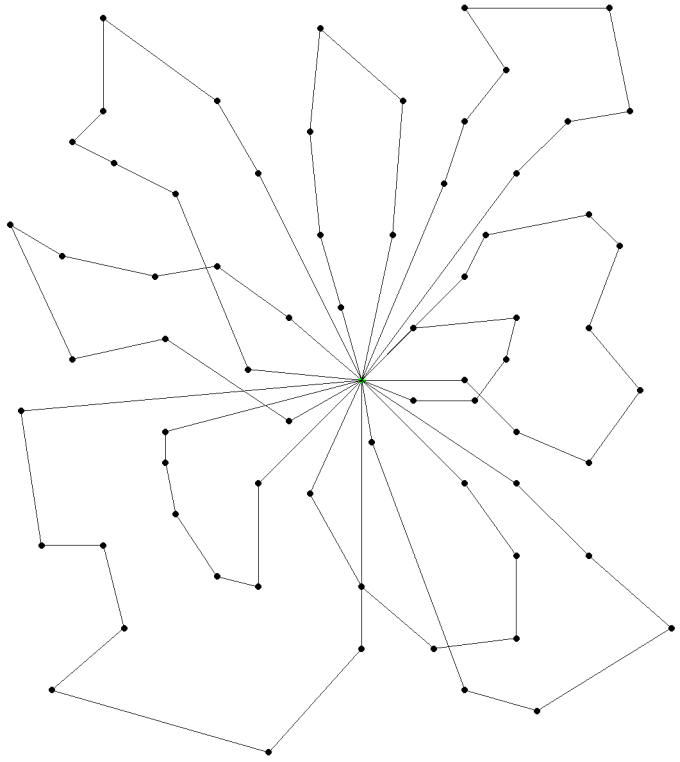


Figura 3.16: Nuova migliore soluzione euristica per l'istanza P-n70-k10, di costo 831.

```
R1:  48, 47, 36, 69, 60, 20, 37, 5
R2:  4, 27, 52, 46, 67
R3:  34, 8, 19, 54, 13, 57, 15, 29, 45
R4:  35, 14, 59, 66, 65, 26
R5:  12, 58, 38, 11, 53, 7
R6:  10, 31, 55, 25, 50, 18, 24
R7:  3, 44, 32, 9, 39, 40
R8:  6, 33, 63, 23, 56, 49, 16, 17
R9:  62, 22, 64, 42, 41, 43, 1, 51
R10:  68, 2, 28, 61, 21, 30
```

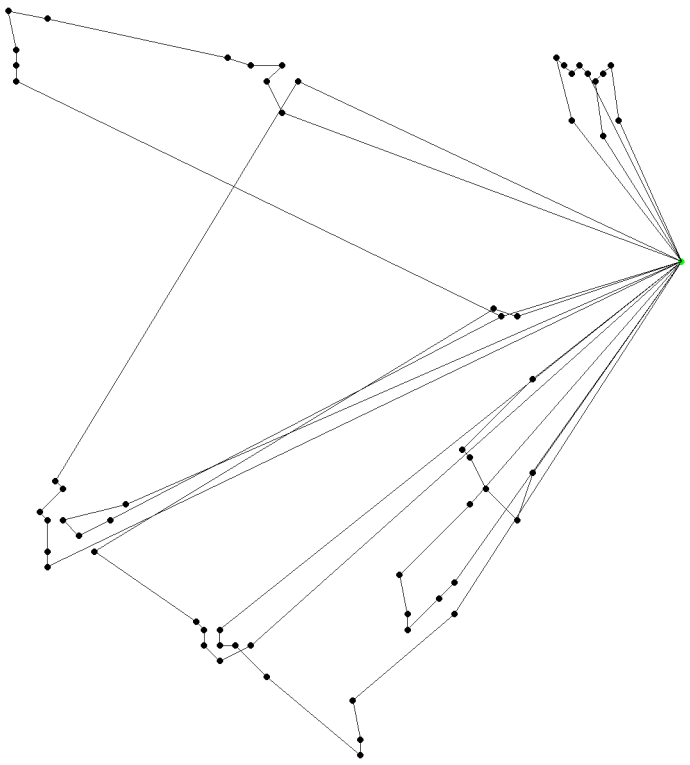


Figura 3.17: Migliore soluzione euristica per l'istanza B-n68-k9, di costo 1075.

```
R1:  47, 7, 46, 13, 66, 65
R2:  31, 4, 60, 35, 27, 64
R3:  21, 16, 34, 44, 48, 9, 26, 1, 15
R4:  43, 5, 14, 62, 54, 30, 59, 45, 12
R5:  29, 8, 37, 41, 2, 22, 51, 61
R6:  17, 49, 57, 20, 52, 56
R7:  42, 36, 67, 32, 6, 58, 10, 24, 18, 28, 33
R8:  25, 3, 11, 39, 23, 55, 40
R9:  63, 50, 38, 53, 19
```

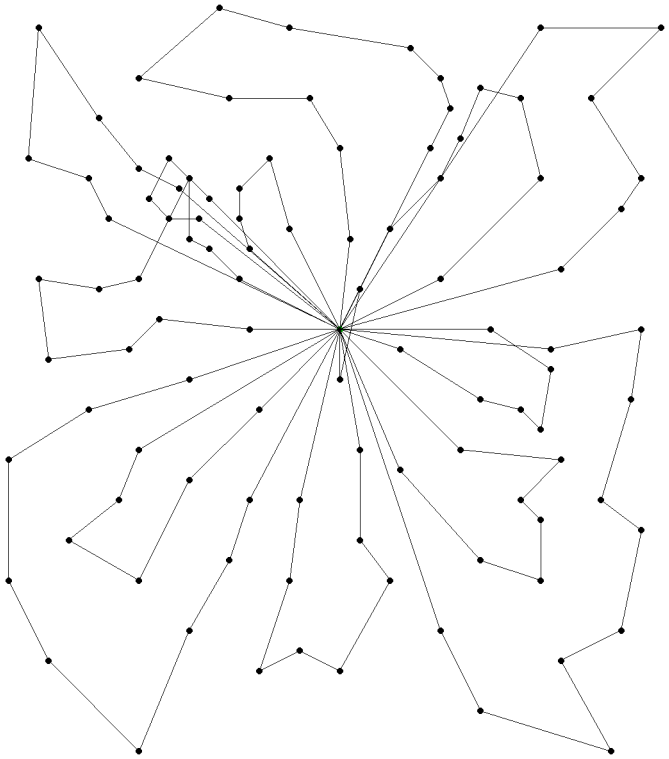


Figura 3.18: Soluzione di partenza per l'istanza E-n101-k14, con costo 1076 (Xu e Kelly, 1996).

```
R1:  61, 16, 86, 38, 44, 91, 98
R2:  1, 51, 9, 81, 33, 79, 50
R3:  58, 2, 57, 42, 14, 43, 15, 41, 22, 74, 73
R4:  69, 70, 30, 32, 90, 63, 10, 31
R5:  80, 24, 29, 78, 34, 35, 71, 65, 66, 20
R6:  12, 68, 3, 77, 76, 28
R7:  82, 48, 47, 19, 7, 52
R8:  88, 62, 11, 64, 49, 36, 46, 8, 18
R9:  59, 93, 85, 100, 92
R10:  54, 55, 25, 39, 67, 23
R11:  6, 96, 99, 37, 5, 84, 17, 45, 83, 60, 89
R12:  40, 21, 72, 75, 56, 4, 26
R13:  94, 95, 97, 87, 13
R14:  53, 27
```

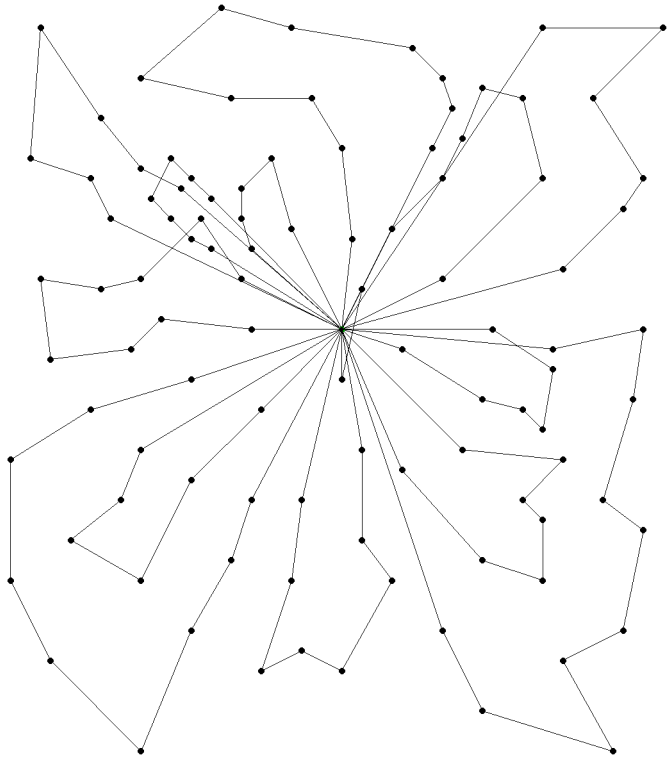


Figura 3.19: Nuova migliore soluzione in assoluto per l'istanza E-n101-k14, con costo 1067.

```
R1:  61, 16, 86, 38, 44, 91, 98
R2:  1, 51, 9, 81, 33, 79, 50
R3:  58, 2, 57, 42, 14, 43, 15, 41, 22, 74, 73
R4:  69, 70, 30, 32, 90, 63, 10, 31
R5:  80, 24, 29, 78, 34, 35, 71, 65, 66, 20
R6:  12, 68, 3, 77, 76, 28
R7:  82, 48, 47, 19, 7, 52
R8:  88, 62, 11, 64, 49, 36, 46, 8, 18
R9:  96, 99, 93, 85, 100, 37, 92
R10:  54, 55, 25, 39, 67, 23
R11:  6, 59, 5, 84, 17, 45, 83, 60, 89
R12:  40, 21, 72, 75, 56, 4, 26
R13:  94, 95, 97, 87, 13
R14:  53, 27
```

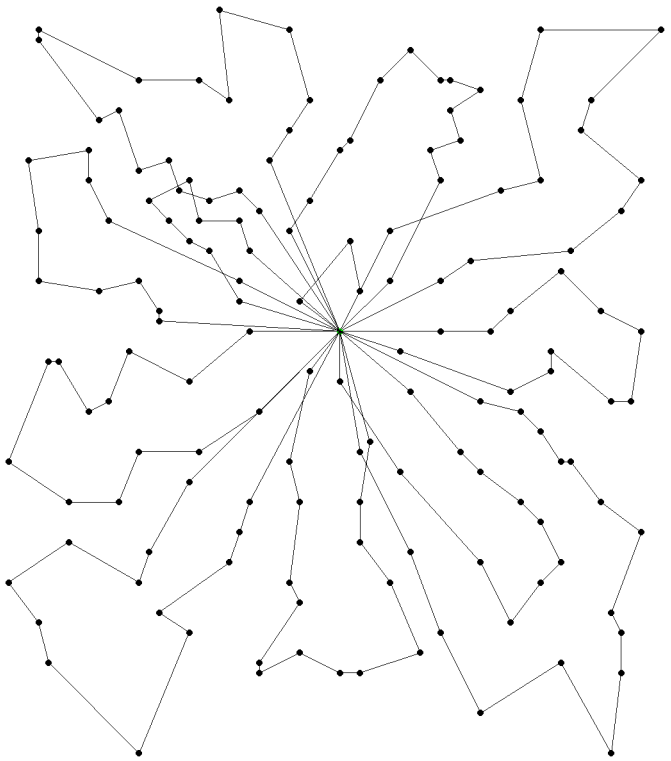


Figura 3.20: Soluzione di partenza per l'istanza M-n151-k12-a, con costo 1023 (Gendreau, Hertz e Laporte, 1996).

```
R1:  61, 114, 99, 43, 86, 97, 69, 23, 57
R2:  32, 1, 120, 80, 28, 31, 82, 140, 113, 26, 8, 60, 81, 27, 46
R3:  11, 100, 2, 83, 131, 20, 59, 3, 101, 51, 77
R4:  119, 22, 70, 116, 121, 115, 36, 85, 35, 84, 128, 29, 129, 53, 127, 126
R5:  78, 16, 118, 130, 50, 21, 79, 74, 34, 104, 9, 62, 38
R6:  90, 10, 54, 106, 73, 117, 89, 39, 75, 105, 30, 49, 76
R7:  137, 44, 107, 65, 93, 92, 42, 64, 88, 40, 94, 19, 141, 150, 148, 142,
147, 17
R8:  63, 37, 52, 15, 45, 91, 72, 33, 125, 124, 122, 123, 71, 5
R9:  144, 145, 109, 87, 135, 143, 4, 149, 146, 47
R10:  110, 18, 55, 134, 67, 13, 136, 41, 66, 111, 56
R11:  139, 68, 133, 14, 58, 25, 95, 96, 24, 98, 132, 6, 102
R12:  103, 108, 12
```

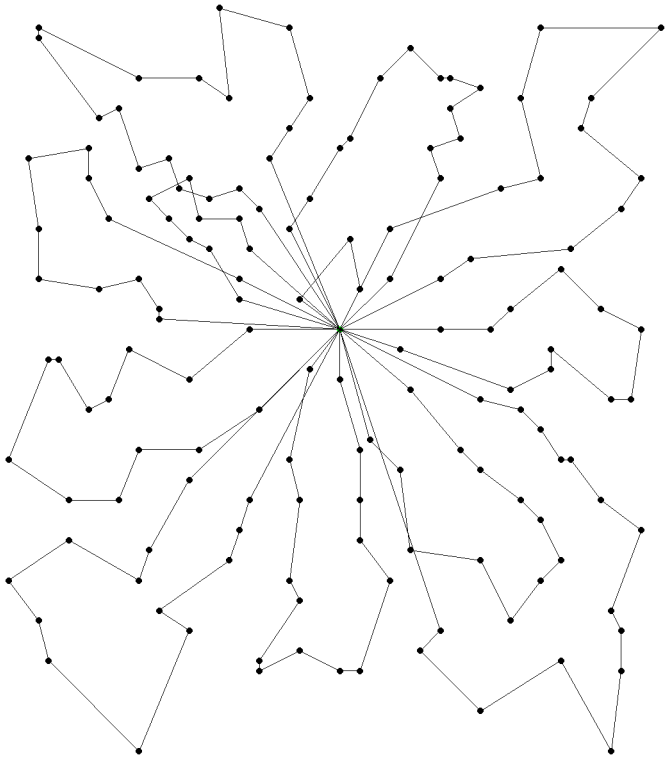


Figura 3.21: Nuova migliore soluzione in assoluto per l'istanza M-n151-k12-a, con costo 1022.

```
R1:  138, 48, 112, 7, 61, 114, 99, 43, 86, 97, 69, 23, 57
R2:  77, 119, 1, 120, 80, 31, 82, 140, 113, 26, 8, 60, 81, 27, 46
R3:  11, 100, 2, 83, 131, 20, 59, 3, 101, 22, 51, 32
R4:  70, 28, 116, 121, 115, 36, 85, 35, 84, 128, 29, 129, 53, 127, 126
R5:  78, 16, 118, 50, 130, 21, 79, 74, 34, 104, 9, 62, 38
R6:  90, 10, 54, 106, 73, 117, 89, 39, 75, 105, 30, 49, 76
R7:  137, 44, 107, 65, 93, 92, 42, 64, 88, 40, 94, 19, 141, 150, 148, 142,
147, 17
R8:  63, 37, 52, 15, 45, 91, 72, 33, 125, 124, 122, 123, 71, 5
R9:  144, 145, 109, 87, 135, 143, 4, 149, 146, 47
R10:  110, 18, 55, 134, 67, 13, 136, 41, 66, 111, 56
R11:  139, 68, 133, 14, 58, 25, 95, 96, 24, 98, 132, 6, 102
R12:  103, 108, 12
```

Bibliografia

[1] K. Altinkemer e B. Gavish, Parallel savings based heuristic for the delivery problem, *Operations Research*, 39:456–469, 1991.

[2] Y. Agarwal, K. Mathur e H.M. Salkin, A set-partitioning-based exact algorithm for the vehicle routing problem, *Networks*, 19:731–749, 1989.

[3] R. Baldacci, A. Mingozzi e E. Hadjiconstantinou, An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation, Technical Report 16, Dipartimento di Matematica, Università di Bologna, 1999.

[4] R. Balinski e R. Quandt, On an integer program for a delivery problem, *Operations Research*, 12:300–304, 1964.

[5] J.E. Beasley, Route-first cluster-second methods for vehicle routing, *Omega*, 11:403–408, 1983.

[6] J.B. Bramel e D. Simchi-Levi, A location based heuristic for general routing problems, *Operations Research*, 43:649–660, 1995.

[7] B. Bullnheimer, R.F. Hartl e C. Strauss, An improved ant system for the vehicle routing problem, *Annals of Operations Research*, 89:319–328, 1999.

[8] B. Bullnheimer, R.F. Hartl e C. Strauss, Applying the ant system to the vehicle routing problem, in *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, a cura di S. Voss, S. Martello, I.H. Osman e C. Roucairol, Kluwer, Boston, 1998, pp. 109–120.

[9] N. Christofides, The vehicle routing problem, *RAIRO (Recherche opérationnelle)*, 10:55–70, 1976.

[10] N. Christofides, A. Mingozzi e P. Toth, The vehicle routing problem, in *Combinatorial Optimization*, a cura di N. Christofides, A. Mingozzi, P. Toth e C. Sandi, Wiley, Chichester, 1979, pp. 315–338.

[11] G. Clarke e J.W. Wright, Scheduling of vehicles from a central depot to a number of delivery points, *Operations Research*, 12:568–581, 1964.

[12] A. Colorni, M. Dorigo e V. Maniezzo, Distributed optimization by ant colonies, in *Proceedings of the European Conference on Artificial Life*, a cura di F. Varela e P. Bourgine, Elsevier, Amsterdam, 1991.

[13] G.B. Dantzig e R. II. Ramser, The truck dispatching problem, *Management Science* 6 (1959), 80.

[14] V.G. Deineko e G.J. Woeginger, A study of exponential neighborhoods for the travelling salesman problem and for the quadratic assignment problem, *Mathematical Programming*, 87:519–542, 2000.

[15] M. Desrochers e T.W. Verhoog, A matching based savings algorithm for the vehicle routing problem, Les Cahiers du GERAD G-89-04, École des Hautes Études Commerciales de Montréal, 1989.

[16] E.W. Dijkstra, A note on two problems in connexion with graphs, *Numerische Mathematik*, 1:269–271, 1959.

[17] M. Dorigo e L.M. Gambardella, Ant colony system: A cooperative learning approach for the traveling salesman problem, *IEEE Transactions on Evolutionary Computation*, 1:53–66, 1997.

[18] M. Dorigo, V. Maniezzo e A. Colorni, Ant system: Optimization by a colony of cooperating agents, *IEEE Transactions on Systems, Man and Cybernetics Part B*, 26:29–41, 1996.

[19] G. Dueck, New optimization heuristics: The great deluge algorithm and the record-to-record travel, *Journal of Computational Physics*, 104:86–92, 1993.

[20] G. Dueck e T. Scheurer, Threshold accepting: A general purpose optimization algorithm. *Journal of Computational Physics*, 90:161–175, 1990.

[21] R. Durbin e D. Willshaw, An analogue approach to the travelling salesman problem using an elastic net method, *Nature*, 326:689–691, 1987.

[22] R.T. Firla, B. Spille, R. Weismantel, Exponential irreducible neighborhoods for combinatorial optimization problems, *Mathematical Methods of Operations Research*, 56:29–44, 2002.

[23] M.L. Fisher e R. Jaikumar, A generalized assignment heuristic for vehicle routing, *Networks*, 11:109–124, 1981.

[24] B.A. Foster e D.M. Ryan, An integer programming approach to the vehicle scheduling problem, *Operations Research*, 27:367–384, 1976.

[25] R. Fukasawa, M. Poggi de Aragão, M. Reis e E. Uchoa, Robust Brach-and-Cut-and-Price for the capacitated vehicle routing problem, *Relatórios de Pesquisa em Engenharia de Produção*, RPEP vol. 3 n. 8, Universidade Federal Fluminense, 2003.

[26] T.J. Gaskell, Bases for vehicle fleet scheduling, *Operational Research Quarterly*, 18:281–295, 1967.

[27] M. Gendreau, A. Hertz e G. Laporte, A tabu search heuristic for the vehicle routing problem, *Management Science*, 40:1276–1290, 1994.

[28] M. Gendreau, A. Hertz e G. Laporte, New insertion and postoptimization procedures for the traveling salesman problem, *Operations Research*, 40:1086–1094, 1992.

[29] M. Gendreau, G. Laporte e J.-Y. Potvin, Metaheuristics for the capacitated VRP, in *The Vehicle Routing Problem*, a cura di P. Toth e D. Vigo, SIAM monographs on discrete mathematics and applications, 2002, cap. 6.

[30] H. Ghaziri, Algorithmes connexionnistes pour l'optimisation combinatoire, Tesi di Dottorato, École Polytechnique Fédérale de Lausanne, Svizzera, 1993.

[31] H. Ghaziri, Solving routing problems by a self-organizing map, in *Artificial Neural Networks*, a cura di T. Kohonen, K. Makisara, O. Simula e J. Kangas, North-Holland, Amsterdam, 1991, pp. 829–834.

[32] H. Ghaziri, Supervision in the self-organizing feature map: Application to the vehicle routing problem, in *Meta-Heuristics: Theory and Applications*, a cura di I.H. Osman e J.P. Kelly, Kluwer, Boston, 1996, pp. 651–660.

[33] B.E. Gillett e L.R. Miller, A heuristic algorithm for the vehicle dispatch problem, *Operations Research*, 22:340–349, 1974.

[34] F. Glover e M. Laguna, Tabu search, in *Modern Heuristic Techniques for Combinatorial Problems*, a cura di C.R. Reeves, Blackwell, Oxford, 1993, pp. 70–150.

[35] F. Glover e M. Laguna, *Tabu Search*, Kluwer, Boston, 1997.

[36] J.H. Holland, *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, 1975.

[37] J.J. Hopfield e D.W. Tank, Neural computation of decisions in optimization problems, *Biological Cybernetics*, 52:141–152, 1985.

[38] D.S. Johnson e L.A. McGeoch, The traveling salesman problem: A case study, in *Local Search in Combinatorial Optimization*, a cura di E.H.L. Aarts e J.K. Lenstra, pp. 215–310, Wiley, Chichester, 1997.

[39] H. Kawamura, M. Yamamoto, T. Mitamura, K. Suzuki e A. Ohuchi, Cooperative search on pheromone communication for vehicle routing problems, *IEEE Transactions on Fundamentals*, E81-A:1089–1096, 1998.

[40] T. Kohonen, *Self-Organization and Associative Memory*, Springer, Berlino, 1988.

[41] G. Laporte e F. Semet, Classical heuristics for the capacitated VRP, in *The Vehicle Routing Problem*, a cura di P. Toth e D. Vigo, SIAM monographs on discrete mathematics and applications, 2002, cap. 5.

[42] S. Lin, Computer solutions of the traveling salesman problem, *Bell System Technical Journal*, 44:2245–2269, 1965.

[43] S. Lin e B. Kernighan, An effective heuristic algorithm for the traveling salesman problem, *Operations Research*, 21:498–516, 1973.

[44] Y. Matsuyama, Self-organization via competition, cooperation and categorization applied to extended vehicle routing problems, in *Proceedings of the International Joint Conference on Neural Networks*, Seattle, 1991, pp. 385–390.

[45] R.H. Mole e S.R. Jameson, A sequential route-building algorithm employing a generalized savings criterion, *Operational Research Quarterly*, 27:503–511, 1976.

[46] D. Naddef e G. Rinaldi, Branch-and-cut algorithms for the capacitated VRP, in *The Vehicle Routing Problem*, a cura di P. Toth e D. Vigo, SIAM monographs on discrete mathematics and applications, 2002, cap. 3.

[47] I.M. Oliver, D.J. Smith e J.R.C. Holland, A study of permutation crossover operators on the traveling salesman problem, in *Proceedings of the Second International Conference on Genetic Algorithms*, a cura di J.J. Grefenstette, Lawrence Erlbaum, Hillsdale, 1987, pp. 224–230.

[48] I. Or, *Traveling salesman-type combinatorial optimization problems and their relation to the logistics of regional blood banking*, dissertazione Ph. D., Northwestern University, Evanston, IL, 1976.

[49] I.H. Osman, Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem, *Annals of Operations Research*, 41:421–451, 1993.

[50] J. Renaud, F.F. Boctor e G. Laporte, A fast composite heuristic for the symmetric traveling salesman problem, *INFORMS Journal on Computing*, 8:134–143, 1996.

[51] J. Renaud, F.F. Boctor e G. Laporte, An improved petal heuristic for the vehicle routing problem, *Journal of the Operational Research Society*, 47:329–336, 1996.

[52] Y. Rochat e E.D. Taillard, Probabilistic diversification and intensification in local search for vehicle routing, *Journal of Heuristics*, 1:147–167, 1995.

[53] D.M. Ryan, C. Hjorring e F. Glover, Extensions of the petal method for vehicle routing, *Journal of the Operational Research Society*, 44:289–296, 1993.

[54] V.I. Sarvanov e N.N. Doroshko, The approximate solution of the travelling salesman problem by a local algorithm with scanning neighborhoods of factorial cardinality in cubic time, (in russo), *Software: Al-*

gorithms and Programs 31, Mathematical Institute of the Belorussian Academy of Sciences, Minsk, 1981, 11–13.

[55] M. Schumann e R. Retzko, Self-organizing maps for vehicle routing problems — minimizing an explicit cost function, in *Proceedings of the International Conference on Artificial Neural Networks*, a cura di F. Fogelman-Soulie, Parigi, 1995, pp. 401–406.

[56] E.D. Taillard, Parallel iterative search methods for vehicle routing problems, *Networks*, 23:661–673, 1993.

[57] P.M. Thompson e H.N. Psaraftis, Cyclic transfer algorithms for the multivehicle routing and scheduling problems, *Operations Research*, 41:935–946, 1993.

[58] P. Toth e D. Vigo, An overview of vehicle routing problems, in *The Vehicle Routing Problem*, a cura di P. Toth e D. Vigo, SIAM monographs on discrete mathematics and applications, 2002, cap. 1.

[59] P. Toth e D. Vigo, Branch-and-bound algorithms for the capacitated VRP, in *The Vehicle Routing Problem*, a cura di P. Toth e D. Vigo, SIAM monographs on discrete mathematics and applications, 2002, cap. 2.

[60] P. Toth e D. Vigo, The granular tabu search (and its application to the vehicle routing problem), Technical Report OR/98/9, DEIS, Università di Bologna, 1998.

[61] A. Van Breedam, *An analysis of the behavior of heuristics for the vehicle routing problem for a selection of problems with vehicle-related, customer-related, and time-related constraints*, dissertazione Ph.D, University of Antwerp, 1994.

[62] A. Volgenant e R. Jonker, The symmetric traveling salesman problem and edge exchange in minimal 1-trees, *European Journal of Operational Research*, 12:394–403, 1983.

[63] P. Wark e J. Holt, A repeated matching heuristic for the vehicle routing problem, *Journal of the Operational Research Society*, 45:1156–1167, 1994.

[64] A. Wren, *Computers in Transport Planning and Operation*, Ian Allan, London, 1971.

[65] A. Wren e A. Holliday, Computer scheduling of vehicles from one or more depots to a number of delivery points, *Operational Research Quarterly*, 23:333–344, 1972.

[66] J. Xu e J.P. Kelly, A network flow-based tabu search heuristic for the vehicle routing problem, *Transportation Science*, 30:379–393, 1996.

[67] P. Yellow, A computational modification to the savings method of vehicle scheduling, *Operational Research Quarterly*, 21:281–283, 1970.