UNIVERSITÀ DEGLI STUDI DI PADOVA

Tesina

Algoritmi Euristici per il Vehicle Routing Problem

Relatore: prof. M. Fischetti

Laureando: Aldà Michele

Padova, 22 luglio 2004

Indice

Introduzione

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

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

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

In questo lavoro si vuole specificatamente approfondire il *Vehicle Routing Problem*, strumento principale per modellare la realtà della logistica e del trasporto dei beni, attraverso lo studio degli algoritmi euristici e metaeuristici che lo risolvono. Tra questi ultimi sarà evidenziata in maniera particolare la tecnica del *Tabu Search*, un approccio che garantisce ottime soluzioni in relativamente brevi tempi computazionali.

Capitolo 1

Il Vehicle Routing Problem

Il problema inerente alla distibuzione di beni materiali tra un deposito o un insieme di depositi e i clienti è generalmente noto con il nome di Vehicle Routing Problem. Tale problema è stato proposto da Dantzig e Ramser nel 1959; in letteratura VRP è il nome generico con cui ci si riferisce ad una intera classe di problemi dalle numerose implicazioni pratiche diffuse in tutti i settori concernenti il trasporto di merci, come ad esempio:

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

Di seguito verrà descritta la versione più comune della famiglia dei VRPs, il *Capacitated Vehicle Routing Problem (CVRP)* e altre significative varianti. Passiamo ora a descrivere le caratteristiche comuni e caratterizzanti di ogni VRP.

1.1 Concetti generali

La distribuzione di merce riguarda il servizio di un insieme di clienti attuato mediante una flotta di *veicoli*, localizzati in uno o più *depositi* e affidati ad *autisti*, che si muovono su di 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 clientela e distributore e, contemporaneamente, da minimizzare il costo globale del trasporto.

La rete stradale è generalmente descritta da un grafo orientato o meno; i suoi vertici rappresentano la posizione dei clienti e dei depositi, mentre gli archi modellano i collegamenti stradali. Si tratta nella fattispecie di un grafo pesato, ovvero ad ogni arco è associato un costo, che generalmente simboleggia la lunghezza del collegamento, ma in modelli particolarmente raffinati può rappresentare il tempo di percorrenza, dipendente dal tipo di veicolo che percorre tale collegamento o dal periodo di tempo durante il quale l'arco è attraversato.

Ad ogni vertice del grafo, e quindi ad ogni potenziale cliente, è associato:

- la quantità di merce (*domanda*), di uno o più tipi, che deve essere recapitata o raccolta;
- il periodo del giorno (*time window*) durante il quale può o deve avvenire il servizio, ad esempio, orari legati all'apertura di un esercizio. Si tratta di un'informazione addizionale caratteristica solamente di una ristretta classe di VRP;
- il tempo necessario per consegnare o raccogliere la merce;
- un sottoinsieme dei veicoli utilizzati per servirlo, ristretti, ad esempio, a causa di problemi logistici o di accessibilità;
- un'eventuale *priorità* nel caso non sia possibile servire tutti i clienti, ed l'eventuale *penalità* 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 associato 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 si può suddividere in sottoproblemi indipendenti, ognuno associato al singolo deposito.

Il trasporto delle merci è affidato ad una flotta di veicoli la cui composizione e dimensione può essere un parametro distintivo del problema. Caratteristiche tipiche dei veicoli sono:

- deposito di partenza, al quale i veicoli sono obbligati o meno a far ritorno alla fine del loro *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, ad esempio, alla presenza di celle frigorifere assieme a vani non refrigerati;
- *costo* associato all'utilizzo del veicolo, per unità di distanza e/o per unità di tempo;
- sottoinsieme dei collegamenti della rete stradale attraversabili dal veicolo.

In ultima analisi, i veicoli sono condotti da *autisti*; a tal proposito possono essere esplicitati ulteriori vincoli sulle modalità di lavoro riguardanti orari, numero e durata delle pause durante il servizio, straordinari, ecc. Comunemente, questi vincoli sono associati direttamente ai veicoli.

Trovare soluzione ad un problema di VRP significa che ogni *route* deve soddisfare determinati vincoli, che dipendono dalla natura della merce trasportata, dal livello di qualità di servizio e dalle caratteristiche di clienti e veicoli presentate in precedenza. Alcuni tipici vincoli sono i seguenti:

- la richiesta totale dei clienti posti lungo un determinato *route* non può superare la capacità del veicolo che lo serve;
- i clienti possono richiedere solamente la consegna di merce, solo il prelievo o entrambi i servizi;
- i clienti dispongono di un arco di tempo limitato (*time window*) nel quale possono ricevere il servizio richiesto e chiaramente durante il periodo lavorativo degli autisti;
- devono essere rispettati eventuali vincoli di precedenza definiti tra i clienti – si pensi al caso in cui la merce da consegnare ad un determinato cliente debba essere preventivamente raccolta presso un secondo cliente presente sul *route* (*pickup and delivery problem*); in questo caso inoltre, interi gruppi di clienti devono essere serviti dallo stesso veicolo.

Gli obiettivi che si possono cercare di conseguire con il calcolo di una soluzione ad un problema di *vehicle routing* sono molteplici e, non di rado, contrastanti. Tipici obiettivi sono:

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

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

Inoltre potrebbe rendersi necessario considerare una versione stocastica del problema, in quanto potrebbe non essere possibile conoscere con certezza l'intera caratterizzazione, in termini di vincoli, dei clienti che il problema richiede di servire.

1.2 Descrizione dei problemi

Dopo questa prima introduzione ai concetti fondamentali dei problemi di vehicle routing, si presenterà una definizione formale, sotto forma di modello su grafo. Per ognuno di questi problemi si ritrovano in letteratura diverse varianti minori e, talvolta, si sono attribuiti gli stessi nomi a problemi diversi. Per questo motivo, verrà descritto il problema nella sua forma base (riferito tramite un acronimo) e successivamente, se necessario, verranno illustrate le eventuali varianti.

In questa sezione, verrà inoltre introdotta una notazione di base cui riferirsi nel seguito della trattazione.

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. Ciò che caratterizza questa tipologia di problemi è il fatto che il servizio è di semplice consegna senza raccolta. Inoltre le richieste dei clienti sono note a priori e deterministiche e devono essere soddisfatte da un solo veicolo; tutti i veicoli sono identici e basati su di un singolo deposito centrale. Gli unici vincoli imposti riguardano le capacità dei veicoli. L'obiettivo è minimizzare il costo totale di servizio,

che può essere una funzione del numero dei *route*, della loro lunghezza complessiva o del tempo di percorrenza.

Consideriamo ora la rappresentazione su grafo di questo problema. 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* non è consentito e ciò è imposto definendo $c_{ii} = +\infty$ per tutti gli $i \in V$. Se il grafo è diretto, la matrice dei costi c sarà asimmetrica e il corrispondente problema è detto *Asymmetric CVRP* (ACVRP), altrimenti $c_{ij} = c_{ji} \forall (i, j) \in A$ e il problema è chiamato *Symmetric CVRP* (SCVRP).

Apriamo ora una breve parentesi sulla notazione utilizzata in teoria dei grafi, per la definizione successiva degli algoritmi. Definiamo quindi:

- $\Delta^+(i)$ la *foward star* (stella uscente) del vertice i come l'insieme di tutti i vertici j tali che $(i, j) \in A$;
- $\Delta^-(i)$ la *backward star* (stella entrante) del vertice i come l'insieme di tutti i vertici l tali che $(l, i) \in A$;
- $\delta(S)$, dato un insieme $S \subseteq V$, denota l'insieme dei lati $e \in E$ con solo un estremo in S. Con un leggero abuso di linguaggio, quando si considera un solo vertice si utilizza la formula $\delta(i)$ in luogo di $\delta(\{i\})$;
- $E(S)$, dato un insieme $S \subseteq V$, indica il sottoinsieme di lati con entrambi i vertici in S.

Riprendendo, ogni cliente $i = 1, \ldots, n$ è associato ad una richesta 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: $d(S) = \sum_{s \in S} d_s$. Indicando con r un *route*, $d(r)$ denota la richiesta complessiva dei vertici da esso visitati.

Un insieme K di veicoli è disponibile presso il deposito. Tali veicoli sono tutti identici e di capacità C; una semplice condizione di ammissibilità del problema richiede $d_i \leq C$ 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. Se consideriamo un sottoinsieme $S \subseteq V$ e indichiamo con $r(S)$ il numero minimo di veicoli necessari per servire tutti i vertici in S, una stima grossolana di $r(S)$ può essere fornita dal *lower bound* $\lceil d(S)/C \rceil$, essendo $r(V \backslash \{0\}) = K_{min}$.

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 del trasporto, definito dalla somma dei costi espressi sugli archi, sia minimo. I vincoli del problema sono i seguenti:

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

Associamo a questi vincoli il modello matematico che li esprime nella maniera più semplice e immediata. Utilizziamo un *modello a tre indici* che, sul grafo orientato $G = (V, A)$, impiega variabili binarie del tipo:

$$x_{ij}^k = \begin{cases} 1 & \text{sse } (i,j) \in A \text{ appartiene al } \textit{route} \text{ servito dal veicolo k, } k \in K \\ 0 & \text{altrimenti} \end{cases}$$

Definiamo inoltre q_i la richiesta associata ad ogni cliente visitato da un circuito e C_k la capacità del generico veicolo k.

Il modello è il seguente:

$$\min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} x_{ij}^k \tag{1.1}$$

al quale sono applicati i seguenti vincoli:

$$\sum_{k \in K} \sum_{j \in V} x_{ij}^k = 1 \quad \forall i \in V \tag{1.2}$$

$$\sum_{i \in V} q_i \sum_{j \in V} x_{ij}^k \leq C_k \quad \forall k \in K \tag{1.3}$$

$$\sum_{j \in V} x_{0j}^k = 1 \quad \forall k \in K \tag{1.4}$$

$$\sum_{i \in V} x_{ih}^k - \sum_{j \in V} x_{hj}^k = 0 \quad \forall h \in C, \ \forall k \in K \tag{1.5}$$

$$\sum_{i \in V} x_{i,n+1}^k = 1 \quad \forall k \in K \tag{1.6}$$

$$x_{ij} \in \{0, 1\} \quad \forall i, j \in V \tag{1.7}$$

Il vincolo (1.2) impone che ogni cliente sia assegnato esattamente ad un solo veicolo che lo serva, il vincolo (1.3) assicura inoltre che nessun veicolo possa servire più clienti di quanti non permetta la sua capacità. I vincoli (1.4), (1.5) e (1.6) sono i vincoli che impongono ad ogni veicolo di partire dal deposito (nodo 0), lasciare un generico nodo h, appartenente all'insieme V, solamente se è entrato in tale nodo e tornare al fittizio nodo $n+1$ [1]. Si può notare come il vincolo (1.6) sia ridondante ma permetta di sottolineare la struttura dei *route*.

Infine (1.7) definisce la natura binaria delle variabili x_{ij}^k.

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

VRP con vincoli di lunghezza dei *route*

Una importante 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 lato o arco (i, j), e la lunghezza totale degli archi appartenenti ad un *route* non può superare un valore massimo definito come T.

Quando, invece, 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 lunghezza o un tempo di percorrenza massimo e, nel contempo, il veicolo che lo percorre ha una limitata capacità di trasporto. 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

[1] Tale vertice può coincidere con il vertice 0

minimizzare la lunghezza totale dei *route* oppure, se il tempo di servizio è incluso nei costi temporali degli archi, la loro durata.

1.2.2 VRP con *Time Window*

Il VRP con *Time Window* (VRPTW) 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 rimane fermo attendendo il tempo a_i perchè possa quindi essere effettuato il servizio. Ogni cliente è 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 staziona presso il cliente. Altri dati del problema sono la matrice dei tempi di viaggio, la cui generica *entry* 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. Usualmente le matrici di costi e tempi coincidono e si suppone che tutti i veicoli partano dal deposito all'istante $t_0 = 0$. VRPTW è di norma rappresentato come un problema asimmetrico, in quanto i valori dei *time window* inducono implicitamente un orientamento dei *route*.

Riassumendo la soluzione di un'istanza di VRPTW consiste nella determinazione di K circuiti semplici di costo minimo tali che :

- ogni circuito visiti il deposito;
- ogni cliente sia visitato da esattamente un circuito;
- la somma delle richieste dei clienti visitati da un *route* non ecceda la capacità C del veicolo che li serve;
- 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 a s_i.

Riprendendo il modello matematico definito per il CVRP in precedenza (vedi 1.2.1) si può definire un modello per il VRPTW assumendo validi tutti i vincoli già esplicitati (eccezion fatta per il vincolo (1.3) se non si tratta di un'istanza di CVRPTW) e introducendo una nuova variabile decisionale y_i^k rappresentante il tempo in cui il veicolo $k \in K$ inizia il servizio presso il cliente i-esimo.

I nuovi vincoli aggiuntivi sono perciò:

$$x_{ij}^k(y_i^k + t_{ij} - y_j^k) \leq 0 \quad \forall (i, j) \in A \ , \forall k \in K \tag{1.8}$$

$$a_i \leq y_i^k \leq b_i \quad \forall(i)\ , \forall k \in K \tag{1.9}$$

Il vincolo (1.8) impone che il veicolo k non possa arrivare a j prima di $y_i^k + t_{ij}$ se percorre l'arco da i a j; (1.9) assicura che ogni *time window* sia rispettata.

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

1.2.3 VRP con *Pickup and Delivery*

Nella versione di base del VRP con *Pickup and 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 può essere memorizzato per comodità solamente un parametro, $d_i - p_i$, che rappresenta la differenza netta di merce necessaria (eventualmente negativa).

Per ogni vertice i, inoltre, sono presenti altri due parametri, O_i e D_i, che caratterizzano così il 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 deve essere consegnata al cliente D_i che deve quindi essere visitato successivamente. Per convenzione si assume che lo scarico della merce avvenga sempre prima del caricamento.

Un problema di VRPPD consiste perciò nel determinare K *route* di costo minimo e tali che:

- ogni circuito visiti il deposito;
- ogni cliente sia visitato da uno e un solo circuito;
- il carico dei veicoli, in ogni punto del *route*, sia non negativo e non ecceda la capacità totale C;
- per ogni cliente i, il vertice O_i, se diverso dal deposito, venga visitato nello stesso circuito e prima della visita di i;
- per ogni cliente i, il vertice D_i, se diverso dal deposito, venga visitato nello stesso circuito e dopo della visita di i;

Spesso l'origine O_i e la destinazione D_i coincidono per tutti i vertici - possono corrispondere, ad esempio, con il deposito - e non sono indicati esplicitamente.

VRPPD generalizza CVRP, ed è quindi NP-difficile in senso stretto.

Capitolo 2

Algoritmi euristici

In questo capitolo saranno presentati alcuni dei più interessanti algoritmi euristici proposti per il *Vehicle Routing Problem*. Nella prima parte verranno illustrati i modelli matematici che vengono utilizzati più di frequente per una risoluzione esatta del problema tramite tecniche di *Branch-and-Bound* e *Branch-and-Cut*.

In tutti i casi che si affronteranno ci si riferirà alla variante classica del CVRP; ad ogni modo tutti i metodi si possono adattare a trattare altre varianti del problema.

2.1 Modelli matematici

Presentiamo ora brevemente le tre categorie di modelli di programmazione lineare proposti in letteratura:

- 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 un 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 debole in presenza di vincoli rigidi.

Abbiamo già presentato un esempio di questi modelli introducendo il CVRP (vedi 1.2.1), dove abbiamo considerato esplicitamente tutti i veicoli che attraversano gli archi: abbiamo una *formulazione a tre indici*, che usa $O(n^2K)$ variabili binarie, ognuna associata ad un arco e ad un veicolo, che specificano da quale veicolo vengano serviti i vari clienti.

I modelli a tre indici si adattano anche a problemi più vincolati rispetto al CVRP (vedi 1.2.2); 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, senza perdita di generalità, solamente grafi orientati.

Accenniamo ad un esempio conosciuto che utilizza questo modello; si tratta dell'approccio esatto al problema SCVRP[1]. Si richiede in primo luogo di aggiungere una copia del deposito al grafo $G = (V, A)$ denominandolo come nodo $n + 1$: ogni *route* viene perciò considerato come il percorso semplice dal nodo 0 al nodo $n + 1$. Ad ogni arco (i, j) vengono associate due variabili di flusso, non negative, le quali hanno il seguente significato (considerando un veicolo che transita dal nodo i al nodo j):

- y_{ij} indica il carico del veicolo;
- y_{ji} indica la capacità residua.

Ciò 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 che 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.

2.1.3 Modelli *set-partitioning*

Proposta per la prima volta nel 1964, la formulazione *set-partitioning* del VRP impiega un'ampia collezione di *route* ammissibili, ognuno dei quali è associato ad una variabile decisionale binaria. Si considera un insieme

[1]Proposto da Baldacci, Mingozzi e Hadjiconstantinou

di circuiti ammissibili $\mathcal{H} = \{H_1, \ldots, H_q\}$, l'$i$-esimo avente costo c_i, ed una serie di coefficienti binari a_{ij}, pari a 1 se e solo se il vertice i è visitato dal *route* H_j. La generica variabile decisionale 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:

$$\min \sum_{j=1}^{q} c_j x_j \tag{2.1}$$

con i vincoli:

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

$$\sum_{j=1}^{q} x_j = K \tag{2.3}$$

$$x_j \in \{0, 1\} \quad \forall j = 1, \ldots, q \tag{2.4}$$

I vincoli (2.1) impongono che ogni vertice sia coperto da uno ed uno solo dei *route* scelti, mentre il vincolo (2.3) determina il numero di circuiti che formano la soluzione.

Quando la matrice dei costi soddisfa la *disuguaglianza triangolare*[2] i vincoli (2.2) possono essere sostituiti dai seguenti:

$$\sum_{j=1}^{q} a_{ij} x_{ij} \geq 1 \quad \forall i \in V \backslash \{0\} \tag{2.5}$$

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*.

[2]Ciò accade quasi sempre nel caso di istanza di tipo geografico senza considerare alcun arrotondamento dei costi che faccia perdere questa proprietà

2.2 Algoritmi euristici classici

Lo sviluppo di algoritmi euristici è mirato a fornire una soluzione di buona qualità ad un problema difficile con un limitato tempo di calcolo. Nella maggior parte dei casi infatti, non si ha il tempo neccessario per applicare metodi esatti proposti dai modelli matematici esposti in precedenza.

In questo senso ci vengono in aiuto gli algoritmi euristici, che impiegano tempi di calcolo relativamente ristretti per fornire soluzioni di buona qualità. Laport e Semet hanno fornito una classificazione di questi metodi, distinguendo tra euristici classici e metaeuristici; la differenza principale sta nel livello di profondità che questi metodi raggiungono nell'esplorazione dello spazio delle soluzioni. Mentre gli euristici classici ottengono buone soluzioni con limitati tempi di calcolo, i metaeuristici approfondiscono la ricerca della *soluzione ottima* nelle zone più promettenti dello spazio delle soluzioni, implementando sofisticate regole di ricerca e di ricombinazione dei risultati parziali ottenuti. Questi metodi, seppur impiegando un tempo di risoluzione maggiore, ottengono soluzioni migliori rispetto ai metodi classici. Rinviamo la trattazione di questi ultimi al capitolo successivo, focalizzando ora l'attenzione sugli euristici classici.

A grandi linee è possibile definire una classificazione degli euristici classici proposti per il VRP:

- euristici *costruttivi*, che operano costruendo gradualmente una soluzione ammissibile, cercando di contenere il costo totale della soluzione stessa;
- euristici *a due fasi*, che scompongono il problema in una fase di divisione dei vertici in gruppi (*cluster*) e una fase di costruzione dei *route* ammissibili. A loro volta questi metodi si suddividono in due classi:
  - *cluster-first, route-second*, i vertici sono in un primo momento raggruppati in *cluster* e poi viene calcolato un *route* per ogni *cluster*;
  - *route-first, cluster-second*, un *route* viene costruito su tutti i vertici per poi essere suddiviso;
- euristici *migliorativi*, che si applicano ad una soluzione preesistente e non necessariamente ammissibile con l'intento di migliorarla operando tipicamente attraverso lo scambio di archi o vertici tra diversi *route*.

Quasi tutti i metodi presentati in questa sezione faranno riferimento alla variante CVRP, anche se sono possibili riadattamenti per poter risolvere problemi più vincolati.

2.2.1 Metodi costruttivi

Gli algoritmi costruttivi consistono principalmente nell'inserire gradualmente i vertici nei *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 unico *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 divisi in:

- *sequenziali*, viene costruito un *route* alla volta sino all'esaurimento dei vertici; in nessun caso si può decidere tra più *route* in cui inserire un vertice;
- *paralleli*, più *route* vengono costruiti contemporaneamente, il loro numero può essere fissato a priori o derivare dalla fusione progressiva di *route* più piccoli già calcolati.

Algoritmo di Clarke e Wright

Si tratta probabilmente del più noto algoritmo euristico proposto per il VRP, e si applica in maniera naturale a problemi per i quali il numero di veicoli non è predefinito. Questo algoritmo ha una versione parallela e una sequenziale.

- *Operazione comune* - 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 ordine decrescente.
- *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 dar luogo ad un nuovo *route* ammissibile. Se questo passo va a buon fine si crea il nuovo *route* con la fusione, altrimenti si applica sempre questo passo al *route* successivo. L'algoritmo si arresta quando non è più possibile effettuare alcuna fusione.

o in alternativa,

- *Versione parallela* - Si considerano i *saving* ordinati e si procede determinando se è possibile fondere insieme due *route* esistenti, contenenti rispettivamene l'arco $(0, j)$ e l'arco $(i, 0)$, dato il *saving* s_{ij}, ottenendo un nuovo *route* ammissibile.

Testato su note istanze del problema, questo algoritmo presenta nella sua versione parallela le prestazioni migliori, anche se i risultati rimangono lontani dalle soluzioni ottime conosciute. Entrambe le versioni, inoltre, producono buoni *route* inizialmente ma tendono poi a perdere di interesse, divenendo talvolta troppo estesi geograficamente. Per ovviare a questo difetto è stato proposto l'utilizzo di un *parametro di forma dei route*, λ, che modifica i *saving* secondo la formula: $s_{ij} = c_{i0} + c_{0j} - \lambda c_{ij}$. Viene in questo modo data più enfasi alla distanza dei vertici da connettere, per valori maggiori di λ.

Algoritmo di inserzione di Mole e Jameson

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

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

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

Definite queste funzioni è possibile definire i passi dell'algoritmo:

1. Se esistono vertici liberi, si seleziona un generico vertice v tra questi e il *route* così generato $(0, v, 0)$. In caso contrario il procedimento ha termine.

2. Per ogni vertice libero k si determinano i vertici i_k e j_k, tra tutti i vertici r e s adiacenti al *route* emergente, tali che il costo di inserzione ammissibile sia $\alpha^*(i_k, k, j_k) = min\{\alpha(r, k, s)\}$. Se non si rende disponibile nessun vertice per l'inserimento nel *route* in costruzione 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 che possono essere inseriti. Tale vertice andrà inserito tra i vertici i_{k^*} e j_{k^*}.

3. Si ottimizza il *route* corrente con una procedura *3-opt*[3] e si riprende dal punto 2.

[3]Tale procedura consiste nell'effettuare un cambiamento nel *route* sostituendo casualmente tre archi che lo compongono con tre nuovi archi scelti ancora a caso, garantendo sempre la connessione del *route* stesso

I parametri λ e μ possono essere variati per consentire una diversa regola di inserimento.

Algoritmo di inserzione sequenziale di Christofides, Mingozzi e Toth

Questo algoritmo implementa una metodologia più sofisticata che fa ancora uso di due parametri λ e μ. Tale strategia si suddivide in due fasi, la prima sequenziale e la seconda parallela.

1. *Inizio della Fase 1 (sequenziale)*
   Si inizializza un indice di *route* k pari a 1.

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

3. Definiamo $\delta_{i^*} = min_{i \in S_k}\{\delta_i\}$, 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 3, finchè nessun altro vertice può essere inserito nel *route* k.

4. Se tutti i vertici sono stati inseriti il procedimento ha termine; in caso contrario si pone $k = k + 1$ e si torna al passo 2.

5. *Inizio della Fase 2 (parallela)*
   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. Definiamo $J = \{R_1, \ldots, R_k\}$ l'insieme dei *route* così inizializzati.

6. Per ogni vertice i non ancora associato ad un *route* e per ogni *route* $R_t \in J$, si calcola $\epsilon_{ti} = c_{0i} + \mu c_{ii_t}$ e il valore $\epsilon_{t^*i} = min\{\epsilon_{ti}\}$. Associamo quindi il vertice i al *route* R_{t^*} e si ripete il passo 6, sino a quando ciascun vertice non è assegnato ad un *route*.

7. 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 $\tau_i = \epsilon_{t'i} - \epsilon_{ti}$.

8. Si inserisce nel *route* R_t il 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. Successivamente il *route* viene ottimizzato nuovamente con una

procedura *3-opt*. Infine il passo 8. viene ripetuto sino a quando nessun altro vertice può essere inserito nel *route* R_t.

9. Se $|J| \neq 0$ si torna direttamente al passo 6. Se J è vuoto e tutti i vertici sono stati inseriti, l'algoritmo termina; in caso contrario vengono creati nuovi *route* a partire dal passo 1.

Questo algoritmo risulta essere migliore sia in termini di soluzioni trovate che in termini di tempi di calcolo, rispetto all'algoritmo di inserzione di Mole e Jameson presentato in precedenza.

Istanza	M/J	C/M/T	Miglior sol. nota
E051-05e	575	547	524.61
E101-08e	882	851	826.14
E101-10c	599	565	555.43
E151-12c	999	915	865.94
D051-06c	1770	1508	1395.85
D101-09c	883	876	866.37
D101-11c	1545	1418	1291.45
D151-14c	879	827	819.56

Tabella 2.1: Confronto tra euristici ad inserzione sequenziale

2.2.2 Metodi a due fasi

Come già evidenziato, nell'ambito dei metodi a due fasi si possono distinguere due categorie di algoritmi, *cluster-first, route-second* e *route-first, cluster-second*. In questa sezione verranno presentati i più importanti algoritmi del primo tipo, mentre sarà solamente accennata in generale la tecnica che prevede la costruzione preliminare di un *route* che copra tutti i vertici.

Cluster-First, Route-Second*: Algoritmo *sweep

La diffusione di questo algoritmo è attribuita a Gillett e Miller per la risoluzione di istanze planari VRP. L'idea è quella di raggruppare i vertici in insiemi, *cluster*, a seconda della loro posizione angolare rispetto al deposito, e successivamente risolvere un'istanza TSP per ogni *cluster*. Alcune implementazioni prevedono che al termine del procedimento i

route proposti vengano ottimizzati attraverso lo scambio di vertici tra *cluster* adiacenti.

Assumiamo che ogni vertice i sia rappresentato, rispetto al deposito, con le sue coordinate polari (θ_i, ρ_i), dove θ_i rappresenta l'angolo e ρ_i la distanza dal deposito. L'angolo θ_i viene calcolato rispetto al valore di riferimento θ_{i^*} relativo ad un vertice arbitrario i^*. I passi dell'algoritmo sono i seguenti:

1. Si numerano i vertici secondo valori crescenti di θ_i.
2. Si seleziona un veicolo libero k.
3. Iniziando dal vertice libero con il minimo valore di θ_i, si assegnagno progressivamente vertici al veicolo k sino a quando non viene violato il vincolo di capacità dello stesso. Eventualmente, ad ogni inserimento è possibile ottimizzare con una procedura *3-opt*. Se al termine di questo passo sono ancora liberi vertici, si riprende l'esecuzione a partire dal punto 2.
4. Per ogni *cluster* così definito si risolve un'istanza TSP, in modo esatto o approssimato.

Cluster-First, Route-Second: Algoritmo di Fisher e Jaikumar

Questo importante e noto algoritmo si applica quando sono noti a priori il numero di veicoli K. Per determinare la suddivisione in *cluster* si risolve un'istanza 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*. In seconda battuta si risolve il problema di assegnare i *route* a tutti i clienti del VRP in maniera ottima, senza violare i vincoli per i quali ogni vertice deve essere assegnato ad un solo *route* e che la richiesta di merce a carico di ogni vertice sia al più Q.

I passi dell'algoritmo sono i seguenti:

1. Si scelgono in V K vertici $j_1, \ldots, j_K$ per inizializzare i K *cluster*.
2. Per ogni vertice libero i si calcola il costo d_{ik} per la sua allocazione nel *cluster* k:

$$d_{ik} = min\{c_{0i} + c_{ij_k} + c_{j_k0}, c_{j_k0} + c_{j_ki} + c_{i0}\} - (c_{0j_k} + c_{j_k0})$$

3. Si risolve un'istanza GAP con costi d_{ij}, richieste q_i e disponibilità Q.

4. Per ogni *cluster* formato si risolve un'istanza TSP, in modo esatto o approssimato.

Per la fase di inizializzazione (passo 1), gli stessi autori propongono di suddividere il piano in K coni e di scegliere come vertici di inizializzazione K vertici fittizi posti sulle bisettrici.

Cluster-First, Route-Second: Algoritmo di Bramel Simchi-Levi

Questo euristico a due fasi prevede che la suddivisione dei vertici in *cluster* avvenga attraverso la risoluzione di un'istanza di *Capacitated Plant Location Problem*, un altro problema NP-difficile che può essere descritto nella maniera seguente: un determinato numero di clienti deve poter usufruire di un servizio attivabile in diverse locazioni. Ogni locazione ha un suo costo di attivazione e se è possibile il collegamento utente-locazione sono previsti dei costi per il collegamento stesso. Ogni cliente quantifica espressamente la propria richiesta di servizio e il PLP consiste nel determinare quali locazioni fornitrici attivare e tutti gli assegnamenti dei clienti alle locazioni, in modo da minimizzare la somma di costi fissi e costi di collegamento.

Nel caso dell'algoritmo di Bramel Simchi-Levi, i clienti corrispondono ai clienti del VRP con la loro richiesta di merce, le locazioni sono proprio le posizioni dei vertici stessi: ne devono essere individuate K rappresentanti altrettanti *concentratori*. Ogni locazione ha associata una capacità massima pari a Q ovvero la capacità dei veicoli nel VRP.

Determinati i concentratori, i *route* sono costruiti inserendo ad ogni passo il cliente per cui si ha il minimo aumento di costo. Se consideriamo il *route* parzialmente costruito k, descritto dal vettore di indici $(0 = i_0, i_1, \ldots, i_l, i_{l+1} = 0)$, e l'insieme $T_k = \{0, i_1, \ldots, i_l\}$, denotiamo con $t(T_k)$ la lunghezza della soluzione ottima di TSP sull'insieme T_k. A questo punto, il costo di inserimento del vertice libero i nel *route* k si definisce essere $d_{ik} = t(T_k \cup \{i\}) - t(T_k)$. Il calcolo di d_{ik} può essere alle volte molto oneroso se definito in questi termini molto generali, per questo si ricorre spesso a delle approssimazioni.

Cluster-First, Route-Second: Algoritmo di *Branch-and-Bound* troncato

Questo algoritmo si rende molto utile quando si affronta la risoluzione di problemi con K variabile. Presentato da Christofides, Mingozzi e Toth, questa procedura prevede la costruzione dell'albero di ricerca all'interno

del quale ogni livello contiene diversi *route* ammissibili. Nell'implementazione proposta si effettua un solo *branching* ad ogni livello, mantenendo un solo *route* considerato come il migliore mediante una opportuna funzione.

Per la descrizione dell'algoritmo, definiamo F_h l'insieme di vertici non ancora visitati al livello h; i passi sono i seguenti:

1. Si pone $h = 1$ e $F_h = V \backslash \{0\}$.
2. 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 contenente *route* generati da i e dagli altri vertici in F_h, calcolati tramite metodi basati su *saving* e costi di inserimento.
3. Si valuta ogni *route* $r \in R_i$, mediante una funzione $f(r)$ basata sul costo di una buona soluzione al TSP sui vertici del route e sul costo dell'albero di copertura minimo sui vertici non ancora coperti.
4. 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 si tolgono da F_h i vertici inseriti nel *route* appena calcolato e si torna al passo 2.

Le prestazioni di questo algoritmo sono migliori se confrontate, ad esempio, con l'algoritmo *sweep* sia in termini di soluzione ottenute sia in tempo di calcolo per le stesse.

Cluster-First, Route-Second: Algoritmi *petal*

Questa categoria di algoritmi sono la versione euristica del modello *set-covering/set-partitioning* illustrato nella sezione 2.1.3. In sostanza, si generano alcuni *route*, detti *petal*, e si selezionano risolvendo un problema di *set-partitioning* del tipo:

$$min \sum_{k \in S} d_k x_k \tag{2.6}$$

con i vincoli:

$$\sum_{k \in S} a_{ik} x_k = 1 \quad \forall i = 1, \ldots, n \tag{2.7}$$

$$x_k \in \{0, 1\} \quad \forall k \in S \tag{2.8}$$

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

È stato dimostrato che se i *route* corrispondo a *cluster* contigui di vertici, il problema può essere risolto in tempo polinomiale: si ottengono quindi buoni risultati se si prova ad implementare questo algoritmo come estensione, ad esempio, dell'algoritmo *sweep*. Grossi vantaggi, dal punto di vista della qualità delle soluzioni, si possono ottenere dalla generazione di due *route* accoppiati o intersecati: tali algoritmi si definiscono *2-petal* e ottengono risultati di buon livello anche sotto il profilo del tempo d'esecuzione.

Istanza	Sweep	F/J	B/S	BBT	2-petal	Miglior sol. nota
E051-05e	532	524	**524.61**	534	**524.61**	524.61
E101-08e	851	833	832.9	851	830.4	826.14
E101-10c	560	560	-	560	560.08	555.43
E151-12c	888	885	-	885	877.29	865.94
D051-06c	1518	1518	-	1509	1470.31	1395.85
D101-09c	949	876	-	878	885.87	866.37
D101-11c	1389	1420	1461.2	1386	1354.23	1291.45
D151-14c	937	824	826.1	816	824.77	819.56

Tabella 2.2: Confronto tra euristici costruttivi

Route-First, Cluster-Second

Questi algoritmi prevedono la costruzione preliminare di un *tour* come soluzione di un'istanza TSP su tutti i vertici del grafo in considerazione, ignorando qualsiasi vincolo proprio del VRP, come, ad esempio, la capacità del singolo veicolo. In un secondo momento il *tour* viene scomposto in più *route* tenendo presente i vincoli imposti dal problema: questi *route* rappresentano la soluzione finale all'istanza VRP. È stato dimostrato che la seconda fase di ripartizione in *route* equivale a risolvere un'istanza di cammino minimo su di un grafo aciclico e quindi può benissimo essere affrontata utilizzando, ad esempio, l'algoritmo di Dijkstra in tempo $O(n^2)$. Seppur in teoria molto semplici come metodi, in pratica non si registrano risultati rilevanti di questi algoritmi su *benchmark* di VRP.

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. In questo senso, sono identificabili due strategie principali: focalizzare l'azione al fine di migliorare il singolo *route* oppure guidare l'ottimizzazione attraverso lo scambio di vertici e collegamenti tra diversi *route*.

Metodi migliorativi *single-route*

Molte delle procedure classificabili in questa sezione possono essere considerate varianti del meccanismo di λ-*opt* proposto da Lin: λ vertici sono rimossi dal *route*, per ricomporre il quale sono necessari λ nuovi collegamenti tra di essi. Nel generare i vari schemi di riconnessione, la procedura si arresta in presenza di un minimo locale quando nessun altro scambio utile può essere effettuato. La verifica della λ-ottimalità della soluzione può essere calcolata in tempo $O(n^\lambda)$.

Sono stati proposti numerosi schemi di scambio per ottenere i risultati migliori, alcuni prevedono, ad esempio, la variazione dinamica del parametro λ in fase di esecuzione, ma i più interessanti sono:

- *Or-opt*, proposto da Or, consiste nello spostare stringhe di 1, 2 o 3 vertici consecutivi in altre posizioni. È un metodo che richiede tempo $O(n^2)$ ed assomiglia molto al *3-opt*;
- *4-opt** che effettua scambi di catene di vertici di dimensione 2 con catene di dimensione massima w e ottimizza in tempo $O(wn)$.

Metodi migliorativi *multi-route*

I metodi migliorativi *multi-route* cercano di abbassare il costo di una soluzione mediante lo scambio di vertici o lati tra diversi *route*. Gli schemi di scambio sono anche qui molto vari, ma focalizzeremo la nostra attenzione soprattutto su due versioni notevoli:

- Thompson e Psaraftis descrivono uno schema di scambio *b-ciclico* di grado k: considerati b *route* se ne effettua una permutazione circolare e k vertici sono tasferiti da ogni *route* al successivo nella permutazione ciclica.
- Van Breedam classifica le operazioni di scambio in quattro categorie:

- *string cross (SC)*: due *stringhe*, o *catene*, di vertici vengono scambiate tra loro incrociando due lati appartenenti a diversi *route*;
- *string exchange (SE)*: due stringhe di al più k vertici vengono scambiate tra due diversi *route*;
- *string relocation (SR)*: una stringa di al più k vertici viene riassegnata ad un altro *route*;
- *string mix (SM)*: viene scelta la mossa migliore tra *SR* e *SE*.

Per valutare la bontà di una strategia rispetto ad un'altra, a seconda dei casi in esame, Van Breedam ha proposto l'utilizzo di diversi parametri. Con questi parametri si definiscono la lunghezza delle catene di vertici da scambiare (k) e la qualità della soluzione ottenuta. Inoltre la selezione delle mosse può essere effettuata a seconda che si segua un paradigma del tipo *First Improvement*, nel quale la prima mossa migliorativa viene attuata, o *Best Improvement*, con il quale viene praticata la mossa che realizza il maggior miglioramento.

Capitolo 3

Algoritmi metaeuristici

Le caratteristiche principali degli algoritmi metaeuristici sono l'esplorazione approfondita delle regioni, considerate più promettenti, dello spazio delle soluzioni e l'impiego di sofisticate regole di ricerca del *neighborhood*, particolari strutture dati e metodi di ricombinazione delle soluzioni. Una caratteristica che spesso distingue questi algoritmi da quelli euristici è che il procedimento di ricerca può passare attraverso soluzioni non ammissibili e/o fasi non migliorative.

Il tempo necessario a questi algoritmi per giungere ad una soluzione ottima è sensibilmente maggiore rispetto alle *performance* degli euristici classici, ma i risultati ottenuti sono solitamente di qualità superiore. Inoltre, l'esecuzione di questi algoritmi è subordinata alla corretta valutazione e impostazione di un predeterminato numero di parametri, propri dell'algoritmo stesso, al fine di adattare il metodo di risoluzione al problema e ottenere la soluzione migliore possibile.

La famiglia dei metaeuristici proposti per il problema del *vehicle routing problem* si può suddividere 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 ammissibile x_i, e passa, ad ogni iterazione t, dalla soluzione corrente x_t alla migliore soluzione, contenuta nel *neighborhood* $N(x_t)$, x_{t+1}, finchè non risulta soddisfatto un opportuno criterio di arresto. Come detto, i metodi metaeuristici possono non rispettare la clausola di diminuire ad ogni iterazione il costo della soluzione ottima, perciò, in generale, non è detto che, se $f(x)$ denota il costo della soluzione x, $f(x_{t+1}) \leq f(x_t)$. Infatti, per favorire la diversificazione ed evitare fasi di stallo intorno a punti di minimo locale, può essere necessario attraversare sequenze di soluzioni peggiorative.

Nel caso degli algoritmi genetici, ad ogni iterazione viene considerata una popolazione di soluzioni: ogni popolazione deriva dalla precedente attraverso la combinazione delle soluzioni migliori e l'eliminazione delle peggiori. Negli *Ant System* vengono sfruttate informazioni raccolte alle iterazioni precedenti per creare numerose nuove soluzioni. Le *reti neurali*, invece, sono meccanismi in grado di auto-regolare un insieme di coefficienti interni, progredendo verso soluzioni sempre migliori.

3.1 *Simulated Annealing*

L'approccio proposto con il metodo *SA* prevede che, ad ogni iterazione t, venga scelta casualmente una soluzione x appartenente al *neighborhood* $N(x_t)$; se è verificato che $f(x) \leq f(x_t)$, allora x_{t+1} è 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 $f(x) - f(x_t)$ definita come:

$$p_t = e^{-\frac{f(x)-f(x_t)}{\theta_t}} \tag{3.1}$$

in cui θ_t è un parametro proprio dell'algoritmo chiamato *temperatura* all'iterazione t. Di solito θ_t è una funzione decrescente in t, proprio per limitare la probabilità che all'aumentare delle iterazioni venga scelta una soluzione peggiore.

I criteri d'arresto sono tre:

- il valore f^* della migliore soluzione attuale non diminuisce di una percentuale prefissata negli ultimi cicli di iterazioni;
- il numero di mosse accettate negli ultimi cicli è inferiore ad una percentuale prefissata;

- sono stati eseguiti un numero predeterminato di cicli.

3.1.1 Algoritmo di Osman

L'algoritmo di Osman implementa il meccanismo di *Simulated Annealing* applicato al VRP. In primo luogo, definisce la struttura del *neighborhood* attraverso un meccanismo di λ-*interchange*: selezionati due *route* p e q vengono scelti due insiemi di clienti S_p e S_q di taglia inferiore o uguale a λ. Vengono effettuati tutti gli scambi possibili che mantengano ammissibili entrambi i *route* p e q. L'algoritmo di Osman si può quindi riassumere in questi passi:

1. *Fase 1 - discesa*
   Viene calcolata una soluzione iniziale attraverso il metodo di Clarke e Wright.

2. Si avvia la ricerca migliorativa λ-*interchange* sino a quando non è più possibile migliorare il *neighborhood*.

3. *Fase 2 - ricerca del Simulated Annealing*
   Si fissa come soluzione di partenza la soluzione ottenuta al termine della Fase 1. Si opera poi una ricerca all'interno del *neighborhood* e senza effettuare nessuna mossa si memorizzano $\Delta_m in$ e $\Delta_m ax$, quali variazioni minima e massima della funzione obiettivo, e β come numero di scambi potenziali.

4. Si esplora il *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 dopo un'intera esplorazione del *neighborhood* nessuna soluzione migliore di x_t è stata trovata, si indica con x la migliore soluzione nel *neighborhood* e 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 definita come in (3.1).

5. Si aggiorna il parametro θ secondo delle regole definite a seconda dei casi (si può avere un *incremento occasionale* o un *decremento normale*). Se si è raggiunto il termine dell'iterazione si chiude la procedura, altrimenti si torna al punto 4.

La regola di aggiornamento della temperatura (*cooling schedule*) differisce dalla teoria definita nel *Simulated Annealing*: θ non è sempre diminuito. Quando $x_{t+1} = x_t$, la temperatura corrente può essere dimezzata o posta al valore di temperatura con il quale è stata identificata la soluzione migliore corrente. In conclusione, l'algoritmo impiega tempi d'esecuzione relativamente lunghi per ottenere soluzioni discrete.

3.2 *Deterministic Annealing*

La principale differenza che distingue la tecnica *Deterministic Annealing* dal precedente *Simulating Annealing* consiste nel fatto che le mosse che si intendono effettuare sono accettate o rigettate secondo regole deterministiche. Due implementazioni standard di questa tecnica sono:

1. *threshold accepting*: ad ogni iterazione t, la soluzione x_{t+1} viene accettata solamente se $f(x_{t+1}) < f(x_t)+\theta_1$; quest'ultimo parametro è controllato dall'utente;

2. *record-to-record travel*: si definisce *record* la miglior soluzione ottenuta nella ricerca. La soluzione x_{t+1} è accettata se $f(x_{t+1}) < \theta_2 f(x_t)$ dove θ_2 è un parametro definito dall'utente, di solito poco superiore ad 1. In un test con istanze formate da numerosi vertici, questa implementazione ha ottenuto ottimi risultati, dal punto di vista delle soluzioni ma soprattutto per quanto riguarda i tempi di calcolo.

3.3 Metaeuristici *Tabu Search*

La tecnica *Tabu Search* è stata introdotta da Glover nel 1986. Questa tecnica esplora lo spazio delle soluzioni attraverso la determinazione, ad ogni iterazione t, della soluzione migliore contenuta in un sottoinsieme del *neighborhood* $N(t)$. Al fine di evitare cicli all'interno della ricerca e lo stazionamento della stessa in un intorno di un minimo locale, le buone soluzioni recentemente visionate vengono marcate da un attributo che le rende non selezionabili nelle iterazioni successive: sono definite quindi *tabu*. La soluzione marcata rimane proibita per un intervallo di tempo variabile e il suo stato può cambiare a seconda che si verifichino eventi particolari, ad esempio, una soluzione tabu risulta essere la migliore delle precedenti soluzioni viste.

Rendere non selezionabile un determinato gruppo di soluzioni può essere realizzato attraverso l'utilizzo di una struttura dati (una *tabu list*) regolata secondo una politica FIFO. In questa lista nera di soluzioni, possono essere memorizzati solamente gli attributi principali e distintivi di ogni soluzione, per rapidità di consultazione e ricostruzione delle stesse.

L'applicazione di questa tecnica negli ultimi anni ha portato ottimi risultati nella maggior parte delle istanze più studiate, al punto che è opinione comune che difficilmente possano essere sviluppati nuovi metodi euristici in grado di competere con la tecnica *Tabu Search*.

3.3.1 L'algoritmo *Taburoute*

Gendreau, Hertz e Laporte proposero questo complesso algoritmo dalle numerose caratteristiche innovative. A partire dal *neighborhood*, che viene calcolato selezionando un *route* e determinando tutte le soluzioni ammissibili raggiungibili se viene eliminato un vertice successivamente inserito in un *route* diverso, contenente uno dei suoi p vertici più vicini. Con questa operazione si può creare un nuovo *route* o eliminarne uno esistente. La sequenza di spostamento è stata scritta e proposta dagli stessi tre autori per il TSP, e prende il nome di *GENI - Generalized Insertion*.

Un'ulteriore particolarità di questo algoritmo è che possono venire temporaneamente ignorati i vincoli di capacità dei veicoli percorrenti i *route* o i tempi di percorrenza degli stessi. Si può pervenire quindi a soluzioni non ammissibili, evenienza compensata dal fatto che la funzione obettivo contiene due termini penalizzanti l'eccesso di carico o di percorrenza. Questi termini sono regolati durante l'esecuzione: ogni parametro è diviso a metà, se nelle ultime dieci iterazioni non si sono mai violati i vincoli sopracitati, raddoppiato altrimenti. Lo statistico alternarsi di soluzioni ammissibili e non, permette di diminuire la probabilità che la ricerca si areni su di un minimo locale. In vari momenti poi, ogni *route* è riottimizzato.

Invece di usare una *tabu list*, l'algoritmo *Taburoute* utilizza un sistema di *random tabu tag*, ovvero, se un vertice è spostato dal *route* r al *route* s, il suo reinserimento nel *route* originale è interdetto per un numero θ di iterazioni. Questo parametro θ è determinato in maniera casuale nell'intervallo $[5, 10]$. Inoltre, come evidenziato nel seguito, *Taburoute* penalizza i vertici molto mobili in luogo di quelli che hanno subito pochi spostamenti al fine di aumentare la diversificazione.

Un'ultima interessante caratteristica di *Taburoute* è l'adozione di una tecnica di *false start*: per individuare una soluzione iniziale da cui partire

si generano numerose soluzioni, sulle quali viene effettuata una breve ricerca. La migliore soluzione ricavata è considerata il punto di partenza.

Nel definire i passi dell'algoritmo, assumiamo W l'insieme di vertici candidati allo spostamento verso altri *route*, $q \leq |W|$ è il numero di vertici per i quali si effettua un tentativo di reinserimento e i sono il numero di iterazioni consecutive ammesse senza miglioramenti. I passi dell'algoritmo sono i seguenti:

1. *Inizializzazione* - Si generano $\left\lceil \sqrt{N}/2 \right\rceil$ soluzioni iniziali e si effettua *Tabu Search* con $W = V \backslash \{0\}$, $q = 5K$ e $i = 50$; questo valore di q assicura di selezionare un vertice in ogni *route* con probabilità del 90%.

2. *Miglioramento della soluzione* - Iniziando dalla miglior soluzione nota al passo 1, si effettua il *Tabu Search* con $W = V \backslash \{v_0\}$, $q = 5K$ e $i = 50N$.

3. *Intensificazione* - A partire dalla miglior soluzione ottenuta al punto 2, si pone $W = \lceil |V|/2 \rceil$ contenente i vertici che hanno subito più spostamenti nei passi 1 e 2, $q = |W|$ e $i = 50$ e si effettua il *Tabu Search*.

Le prove dell'algoritmo di *Taburoute* sulle istanze note ha portato al raggiungimento, in alcuni casi, dei migliori risultati conosciuti, mantenendo allo stesso tempo contenuti i tempi di esecuzione.

3.3.2 Algoritmo di Taillard

Le sostanziali analogie che questo algoritmo ha con l'algoritmo di *Taburoute* risiedono nell'analogo utilizzo dei *random tabu tags* e nella diversificazione mirata a penalizzare il movimento continuo degli stessi vertici.

La struttura del *neighborhood* è ottenuta attraverso la tecnica di λ-*interchange* utilzzata da Osman (vedi 3.1.1). A differenza del *Taburoute*, i reinserimenti non sono effettuati con l'utilizzo del meccanismo *GENI*, ma attraverso metodi tradizionali, che garantiscono il mantenimento di soluzioni ammissibili in ogni momento dell'esecuzione, durante la quale i *route* sono ottimizzati con buona frequenza.

L'originalità dell'algoritmo di Taillard consta nella suddivisione del problema in più sottoistanze distinte, caratteristica che privilegia l'esecuzione del problema in ambienti multiprocessore.

Se consideriamo istanze planari di VRP, nelle quali il deposito occupa una posizione centrale rispetto i clienti, risulta semplice suddividere il piano in settori circolari aventi come centro il deposito e, successivamente, i settori in regioni concentriche. Ogni regione può essere risolta in maniera indipendente, mantenendo comunque la necessità di periodici scambi di vertici tra sezioni adiacenti.
Se l'algoritmo si trova a risolvere istanze dove il deposito non è centrale rispetto i clienti, si può pensare di suddividere lo spazio secondo l'arborescenza minima che prende origine dal deposito stesso.

L'algoritmo di Taillard ha ottenuto ottimi risultati (Tabella 3.1).

3.3.3 Algoritmo di Xu e Kelly

Questo algoritmo è contraddistinto da una definizione particolarmente articolata del *neighborhood*. La sua struttura è infatti definita attraverso scambi di vertici tra due *route*, riposizionamenti globali di più vertici miglioramenti locali nei singoli *route*.

Il riposizionamento, in particolare, si attua attraverso l'impiego di un modello *network flow*, che permette di assegnare in maniera ottima un gruppo di vertici, spostandoli dai loro *route* di origine ad altri. L'ottimizzazione dei singoli *route* si ottiene applicando la procedura *3-opt* combinata con una realizzazione particolare di *Tabu Search*. Diversi parametri, modificabili dinamicamente durante l'esecuzione dell'algoritmo, permettono di regolare il funzionamento dello stesso. La variazione dei parametri prevede una sorta di reinizializzazione a partire dalle soluzioni ottime sino a quel momento identificate e memorizzate appositamente.

Proprio l'utilizzo e la difficoltosa impostazione a prori di questi parametri ha penalizzato l'utilizzo di questo algoritmo nei confronti di altri metaeuristici della classe *Tabu Search*, che pure ha ottenuto, su determinate istanze, i risultati migliori conosciuti.

3.3.4 Memoria adattativa

L'introduzione di una struttura dati contenente buone soluzioni, concettualmente una *memoria adattativa*, è riconducibile a Rochat e Taillard, i quali hanno proposto di mantenere dinamicamente le migliori soluzioni note di un problema in risoluzione. Da questa struttura sono poi estratte periodicamente delle soluzioni che, combinate tra loro, sapranno portare ad ulteriori nuove e buone soluzioni.

Per i problemi di VRP, la combinazione di soluzioni diverse prevede l'estrazione e la ricombinazione di interi *route*. Tale caratteristica pone

delle difficoltà evidenti: se vengono estratti dalla memoria i *route* migliori per essere ricombinati, questi devono essere necessariamente disgiunti, ovvero non devono avere vertici in comune. Da qui, la ricombinazione produrrà dei *route* diversi che copriranno porzioni, anche piuttosto ampie ma limitate, di vertici. Si impone perciò il reinserimento dei vertici esclusi attraverso l'utilizzo di euristici costruttivi.

Oggigiorno, su questo settore la ricerca è molto impegnata, in quanto la specifica di *memoria adattativa* permette di rendere estremamente efficaci numerosi metodi di ricerca. Gli stessi Rochat e Taillard hanno ottenuto, con questa strategia, alcuni dei migliori risultati noti sulle istanze più studiate.

Istanza	*Taburoute*	Taillard	Xu e Kelly	*GTS*	Migl. sol. nota
E051-05e	**524.61**	**524.61**	**524.61**	**524.61**	524.61
E101-08e	829.45	**826.14**	**826.14**	828.56	826.14
E101-10c	**819.56**	**819.56**	**819.56**	**819.56**	819.56
E151-12c	1036.16	**1028.42**	1029.56	1033.21	1028.42
D051-06c	**555.43**	**555.43**	**555.43**	**555.43**	555.43
D101-09c	**865.94**	**865.94**	881.38	869.48	865.94
D101-11c	**866.37**	**866.37**	915.24	**866.37**	866.37
D151-14c	1177.76	**1162.55**	-	1173.12	1162.55

Tabella 3.1: Confronto tra metaeuristici *Tabu Search*

3.3.5 *Granular Tabu Search*

Un'ulteriore tecnica che promette sviluppi interessanti è il *Granular Tabu Search* proposto da Toth e Vigo. In questo momento, tale tecnica detiene il miglior rapporto tra risultati ottenuti e tempi d'esecuzione.

L'intuizione, che precede lo sviluppo di questa strategia, consiste nell'osservare che nelle migliori soluzioni ottenute con altri metodi, la probabilità che un lato *lungo* rientri in esse è molto bassa. Da qui l'idea di escludere, a priori, la possibilià che lati più lunghi di una certa soglia (detta *soglia di granularità*) possano essere considerati dalla procedura.

La soglia, proposta dagli autori, è definita da due parametri: β, un termine di *sparsificazione* scelto di solito nell'intervallo $[1, 2]$, e $\bar{c}$, la lughezza media dei lati presenti in una soluzione ottenuta da un euristico veloce. La soglia è $\upsilon = \beta\bar{c}$; il parametro β varia a seconda che si ot-

tengano miglioramenti o meno, e tipicamente permette di considerare solamente il 10/20% dei lati.

Le soluzioni del *neighborhood* sono ottenute mediante un limitato numero di scambi di vertici, all'interno dei singoli *route* come tra *route* distinti; la procedura proposta dagli autori, infatti, prevede di esaminare tutti i possibili scambi in tempo $O(|E(v)|)$, dove $E(v) = \{(i,j) \in E : c_{ij} \leq v\} \cup I$, dove I è un ulteriore insieme di lati che può essere importante considerare, come, ad esempio, i lati incidenti il deposito o i lati appartenenti a buone soluzioni calcolate precedentemente.

L'implementazione del *GTS* comprende, inoltre, caratteristiche proprie dell'algoritmo *Taburoute*, garantendo perciò il calcolo di ottime soluzioni con tempi d'esecuzione tra i più bassi nella categoria *Tabu Search*.

3.4 Algoritmi genetici

Nel 1975 è stato proposto, da Holland, il paradigma alla base degli agoritmi genetici: si tratta di una tecnica che cerca di risolvere problemi imitando i processi tipici dell'evoluzione naturale. In via del tutto generale, l'idea di fondo consiste nel mantenere memoria di *stringhe* di bit, chiamate *cromosomi*, rappresentanti la codifica binaria di una soluzione al problema. L'evoluzione della popolazione è ottenuta applicando degli operatori che simulino i più importanti fenomeni naturali: *riproduzione* e *mutazione*.

Una descrizione generica del paradigma può essere data nelle seguenti modalità: innanzitutto sia definita una popolazione iniziale di *cromosomi* $X^1 = \{x_1^1, \ldots, x_N^1\}$, alla quale, in ogni iterazione $t = 1, \ldots, T$, sono applicate k volte (con $k \leq N/2$) le operazioni descritte dai passi 1 - 3, seguite dal passo 4:

1. *Riproduzione* - Si selezionano, dalla popolazione X^t, due cromosomi *generatori*, privilegiando statisticamente la scelta dei cromosomi migliori;

2. *Ricombinazione* - Si applica un operatore di *crossover* ai due generatori in modo da ottenere una nuova coppia di cromosomi *discendenti*;

3. *Mutazione* - Con piccola probabilità, si applica l'operatore di mutazione ai cromosomi discendenti;

4. *Rinnovo generazionale* - A partire da X^t, si crea la popolazione

X^{t+1} rimuovendo le $2k$ peggiori soluzioni e sostituendole con le $2k$ generate nelle k precedenti applicazioni dei passi 1-3.

Al termine delle T iterazioni, si prende come soluzione finale la migliore dell'ultima generazione.

Ciò che contraddistingue questo paradigma è l'assoluta generalità della sua specifica. Così come è stato descritto, infatti, esso può essere applicato a diverse tipologie di problema. Il punto chiave riguarda, in realtà, la definizione degli operatori di *crossover* e *mutazione*, che necessariamente si differenzieranno a seconda del problema affrontato. Inoltre, bisogna definire come una soluzione ad un determinato problema deve essere codificata in una stringa di bit, imponendo quindi una serie di regole per l'ammissibilità delle stringhe/soluzioni.

Nel caso specifico, per i problemi di VRP, la codifica in stringhe di bit dei *route* che compongono una soluzione non è la traduzione più adatta. Si preferisce codificare i *route* con delle sequenze di interi, dove la posizione dell'intero nella stringa indica l'ordine di visita, nel *route*, del vertice corrispondente. L'intero 0, rappresentante il deposito, può comparire più volte nella stringa ed è spesso utilizzato come carattere separatore tra i route.

Così definita la struttura dati, gli operatori devono essere costruiti in maniera specifica. Un operatore di *crossover* per il TSP, ma facilmente adattabile al VRP, è stato definito da Oliver, Smith e Holland e prende il nome di *order crossover*. Per quanto riguarda gli operatori di *mutazione*, sono stati studiati degli schemi di scambio o di rimozione e reinserimento (schemi *RAR - Remove-and-Reinsert*).

Come applicazioni pratiche, gli algoritmi genetici hanno dato i maggiori contributi sulla risoluzione di istanze più vincolate rispetto al semplice CVRP. Ottimi risultati sono stati ottenuti su *benchmark* di VRPTW, fatto che lascia sperare un'evoluzione di questa strategia anche nell'area dei problemi meno strutturati.

3.5 *Ant System*

Come lascia intuire il nome, questa strategia di risoluzione sfrutta la tecnica usata dalle formiche durante la ricerca del cibo. Questi insetti, infatti, marcano il cammino tramite la secrezione di una sostanza, il *feromone*, che è riconoscibile da tutti i suoi simili. La quantità di feromone presente su un percorso non è casuale, ma dipende direttamente dalla qualità della fonte di cibo raggiungibile e dalla lunghezza del percorso. Col passare del tempo, emergeranno i percorsi più interessanti, che portano alle migliori fonti di approvvigionamento, sui quali maggiore sarà la presenza di feromone.

Prendendo ispirazione da questo processo naturale, Colorni, Dorigo e Maniezzo, hanno proposto questa nuova categoria di metaeuristici, in cui formiche artificiali alla ricerca nello spazio delle soluzioni, simulano i veri insetti. I valori delle funzioni obiettivo rappresentano la qualità delle fonti di cibo e i valori memorizzati in una memoria adattativa (vedi 3.3.4) sono associati alle tracce di feromone.

In sostanza, ogni lato del grafo (v_i, v_j) 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 durante l'esecuzione della procedura. Ad ogni iterazione, delle formiche artificiali partono dai vertici per costruire dei *tour* con un metodo euristico che favorisca il collegamento di vertici vicini o di vertici in cui la presenza di feromone sia molto marcata. In un secondo momento l'aggiornamento della quantità di feromone avviene assumendo che una frazione $1 - p$ $(0 \leq p \leq 1)$ del feromone già presente sul lato selezionato nel *tour* L venga persa e assegnando una frazione $1/L$ del feromone portato dall'insetto che ha costruito il *tour*. La dispersione del feromone è necessaria perchè non emergano soluzioni di scarsa qualità, quali, ad esempio, quelle individuate alle prime iterazioni, penalizzando l'emersione di soluzioni migliori.

Pochi sono i contributi dati nell'applicazione degli *Ant System* al VRP. Dagli esperimenti condotti e dalla scarsità di esperienze nell'applicazione di tale tecnica, si può concludere che il metodo in esame può produrre talvolta risultati interessanti e incoraggianti, ma non può competere con altri metaeuristici senza essere in qualche modo ibridato con procedure d'ottimizzazione locale. Sembra comunque ragionevole attendersi miglioramenti e sviluppi interessanti in futuro.

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 rappresentare 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 acquisita nella computazione: grazie ai pesi associati alle connessioni, le reti neurali godono dell'importante caratteristica che è la capacità di induzione.

I primi tentativi di risolvere problemi di ottimizzazione combinatoria con delle reti neurali sono riconducibil all'opera di Hopfield e Tank. I loro modelli, assieme alle *reti elastiche (EN)* di Durbin e Willshaw e alle *mappe auto-regolanti (SOM, Self-Organizing Map)* di Kohonen, sono stati applicati con successo al TSP.

I modelli *EM* e *SOM*, solo ispirati alla definizione di reti neurali, prevedono sorte di *tour* deformabili, che si adattano gradualmente alla disposizione effettiva dei vertici dell'istanza. Modelli come questi possono facilmente essere applicati a istanze geometriche, ma difficilmente riescono ad integrare al loro interno vincoli più stretti, ad esempio quelli del CVRP o di altre varianti del VRP.

A questo proposito molte sono le soluzioni proposte, ma particolarmente interessante è quella sviluppata da Ghaziri, descritta brevemente come segue:

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

2. Si associa una probabilità di selezione ad ogni anello: tale probabilità dipende sia dalla distanza dell'anello dal vertice corrente, sia dal carico dell'anello stesso in termini di vertici temporaneamente assegnati (vedi punto 4).

3. Secondo la probabilità definita al punto 2, si seleziona un anello.

4. Si assegna il vertice corrente in maniera temporanea 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.

5. Si ripetono i precedenti passi sino a quando ogni vertice ha un'unità del modello sufficientemente vicina.

6. Infine, si assegna permanentemente ogni vertice alla sua unità più vicina, ottenendo così la soluzione finale.

L'aspetto interessante di questa procedura è proprio la definizione probabilistica dell'assegnamento dei vertici agli anelli. Questo metodo fa sì che gli anelli non si avvicinino troppo simultaneamente ai vertici le cui richieste superano le capacità dei veicoli, inoltre fissando alcune probabilità a zero, è possibile adattare il metodo a problemi ancor più vincolati, come ad esempio varianti del VRP con vincoli di tempo.

I risultati ottenuti mostrano come questo metodo produca discreti risultati, anche se non è pensabile un confronto alla pari con altri tipi di metaeuristici.

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.

Ringraziamenti

Un ringraziamento particolare va al relatore, Prof. Matteo Fischetti, per l'aiuto e la disponibilità incondizionata offertami durante la realizzazione di questo lavoro.
Un grazie sincero ai miei genitori e a mio fratello per il sostegno che mi ha permesso di raggiungere questo primo traguardo.

Bibliografia

[1] A. Amberg et al. *Multiple center capacitated arc routing problems: A tabu search algorithm using capacitated trees*, European Journal of Operational Research 124, pp. 360-376, 2000.

[2] N. Bianchessi, *Algoritmi di ricerca locale per il Vehicle Routing Problem with Simultaneous Pick-up and Delivery*, Università degli Studi di Milano, Nov. 2002.

[3] O. Bräysy, M. Gendreau *Tabu Search Heuristics for the Vehicle Routing Problem with Time Windows*, 2001

[4] 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. 315338.

[5] R. De Franceschi, *Algoritmi euristici per il Vehicle Routing Problem*, Università degli Studi di Padova, Dic 2003.

[6] R. De Franceschi, M. Fischetti, P.Toth *A new ILP-based refinement heuristic for Vehicle Routing Problems*, submitted to an international journal.

[7] A.Erera *ISyE6203: Transportation and Supply Chain Systems*, Georgia Institute of Technology, 2003

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

[9] http://neo.lcc.uma.es/radi-aeb/WebVRP/main.html