ALMA MATER STUDIORUM -
UNIVERSITÀ DI BOLOGNA
SEDE DI CESENA

SECONDA FACOLTÀ DI INGEGNERIA CON SEDE A CESENA

CORSO DI LAUREA SPECIALISTICA IN INGEGNERIA ELETTRONICA E DELLE TELECOMUNICAZIONI

TITOLO DELLA TESI

IMPLEMENTAZIONE E SPERIMENTAZIONE DI ALGORITMI PER IL ROUTING DI VEICOLI IN TEMPO REALE

Tesi in

METODI E MODELLI PER IL SUPPORTO ALLE DECISIONI LS

Relatore:
Chiar.mo Prof. Ing. DANIELE VIGO

Presentata da:
DANIELE D'ANDREAMATTEO

Correlatori:
Chiar.mo Prof. JAUME BARCELÓ
Ing. MARIA BATTARRA

Sessione III
Anno accademico 2010/2011

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.

Nell'ambito della logistica, gestire le informazioni in tempi brevi, ottimizzare le decisioni, ridurre la percentuale di mezzi che circolano vuoti e il numero totale di mezzi circolanti, con beneficio sulla viabilità, è di fondamentale importanza. La necessità di trovare un approccio sistemico, dove tutte le parti contribuiscano in modo sinergico, fa sì che anche *l'information technology*, da parte

sua, fornisca delle risposte precise a questi temi, che rappresentano un nodo cruciale nel contenimento dei costi di trasporto.

Una corretta impostazione del problema legato alle attività di recupero e riciclo di materiali di scarto, ad esempio, deve integrare il flusso di prodotti da distribuire, con quello degli scarti che devono essere raccolti, immagazzinati e riciclati. Solo in questo modo si può ottimizzare il processo rendendo così il riciclo più economico della sola distruzione, aiutando in tal modo la crescente attenzione verso le politiche ambientali.

La sfida è quella di generare flussi di merce estremamente veloci e per ottenere questo scopo, le società di servizi logistici cercano di soddisfare i propri bisogni funzionali in termini di:

- positioning e tracing dei veicoli;
- routing on line dei veicoli;
- sicurezza verso furti;
- telediagnosi dello stato del veicolo e delle merci trasportate.

In questo modo, ottimizzando le rotte dei veicoli, si ottiene una riduzione dei costi di gestione della flotta ed un migliore rapporto con i clienti, attraverso una risposta più rapida alle loro esigenze.

Con il termine *simulazione* s'intende l'attività del replicare per mezzo di opportuni modelli una realtà già esistente o da progettare, al fine di studiare, nel primo caso, gli effetti di possibili interventi o eventi in qualche modo prevedibili, o, nel secondo, di valutare diverse possibili scelte progettuali alternative.

L'uso di modelli come strumento di aiuto nei processi decisionali è antico e diffusissimo. Un tipico esempio è quello dei modelli a scala, usati soprattutto in fase di progettazione. Si tratta di modelli che replicano fedelmente, anche se a scala ridotta, la realtà che si vuole rappresentare. Tipici modelli di questo tipo sono i plastici che vengono utilizzati nella progettazione architettonica, o i modelli di strutture che vengono utilizzati per studiare gli effetti di sollecitazioni, ad esempio di tipo sismico.

Altri importantissimi modelli molto usati come *strumenti decisionali*, soprattutto con lo sviluppo e la diffusione della *Ricerca Operativa*, sono i modelli analitici. Si tratta di modelli in cui la realtà sotto esame viene rappresentata per mezzo di variabili e relazioni di tipo logico/matematico. A questa classe appartengono, fra gli altri, i modelli di programmazione lineare (più in generale di programmazione matematica) o i modelli di file d'attesa.

Si tratta di modelli di notevole potenza, che consentono in molti casi di determinare, con un costo contenuto, una o più soluzioni ottime (o comunque soluzioni molto buone) per il problema considerato. Tuttavia al crescere della complessità e della dimensione dei problemi tali modelli diventano di uso sempre più difficile e costoso, ed in molti casi, per le loro stesse caratteristiche, inapplicabili. La complessità di un processo decisionale ha diverse dimensioni: il numero delle variabili, il tipo di relazioni che legano fra loro le variabili, il numero di obiettivi, il numero di attori, cioè di persone che hanno la possibilità di prendere decisioni o di influire su esse, ed infine il grado di incertezza con cui le grandezze in gioco e le relazioni fra le variabili sono conosciute.

Aimsun è un software integrato di modellazione dei trasporti, sviluppato e commercializzato da TSS - sistemi di simulazione di trasporto con sede a Barcellona, Spagna.

Nel primo capitolo descriverò i concetti chiave per il Vehicle Routing Problem. Nel secondo capitolo introdurrò la simulazione: la base portante sarà l'interazione da sistema e modello, che saranno legate strettamente. Seguirà poi una breve introduzione sulla micro simulazione e sull'Aimsun. La tesi è stata elaborata presso l'UPC, Universitat Polytecnica de Catalunia, Dipartimento di Statistica e Ricerca Operativa, palazzina C, a Barcellona, in Spagna, durante i sei mesi di "tesi all'estero". Nel terzo capitolo implementerò in Aimsun l'algoritmo creato: il VRP con Time Window (finestre temporali). Nel quarto capitolo seguirà una simulazione realistica, settando i depositi, la flotta e i clienti.

Capitolo 1

Vehicle Routing Problem

In questo capitolo saranno illustrati i concetti chiave relativi al Vehicle Routing Problem. Per VRP s'intende la distribuzione di beni materiali tra un deposito o tra un insieme di depositi e i clienti. Tale problema è stato proposto da Dantzing e Ramser nel 1959. In particolare mi soffermerò sulle tre varianti del Veichele Routing Problem, VRP con vincoli di capacità (CVRP), VRP con Pickup e Delivery e VRP con Finestre temporali (TWVRP).

1.1 Cenni storici

La maggior parte dei problemi considerati dagli operatori della logistica, sono conosciuti da secoli, basti pensare ad esempio al classico problema del postino cinese formulato da Eulero nel 1736. Consiste nel trovare il percorso più corto in una rete di strade, in modo tale che siano visitate tutte richiudendosi sul punto di partenza (la sede della posta). Oppure quello dei 7 ponti di Könisberg (Fig. 1.1), dove si chiede se è possibile effettuare una passeggiata ritornando al punto di partenza, attraversando tutti i ponti una sola volta.

L'utilizzo di algoritmi di ottimizzazione consente di rendere il processo di pianificazione veloce e, soprattutto, economico. Per sviluppare algoritmi efficaci è necessario descrivere il problema in

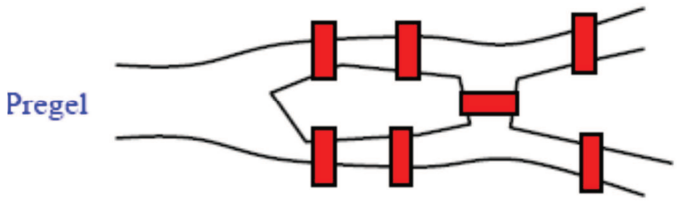


Figura 1.1 *Il problema dei 7 ponti*

modo formale attraverso un modello matematico, che deve rappresentare i suoi aspetti significativi, attraverso variabili di decisione, vincoli e funzioni di costo.

A tale proposito, la nascita della teoria dei grafi, consente di avere un modello astratto del problema.

Nello specifico, riferendoci all'esempio citato, il grafo equivalente alla mappa precedente è quello riportato nella seguente figura (Fig. 1.2).

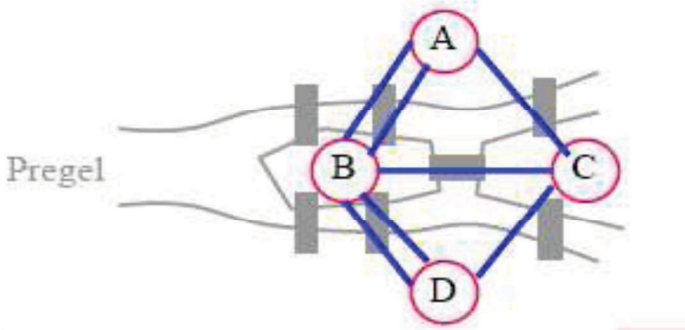


Figura 1.2 *Grafo equivalente per il problema dei 7 ponti*

Volendo rispondere al quesito, ossia se esista un circuito che attraversi tutti gli archi una ed una sola volta (circuito euleriano) si può dire che tale percorso esiste se e solo se in ogni nodo del grafo si ha un *numero pari* di archi incidenti (condizione euleriana). Quindi, il problema dei 7 ponti di Könisberg non ha soluzione.

1.2 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 rete stradale è generalmente descritta tramite un *grafo*, che può essere non orientato, ibrido e orientato (Figg.1.3, 1.4 e 1.5).

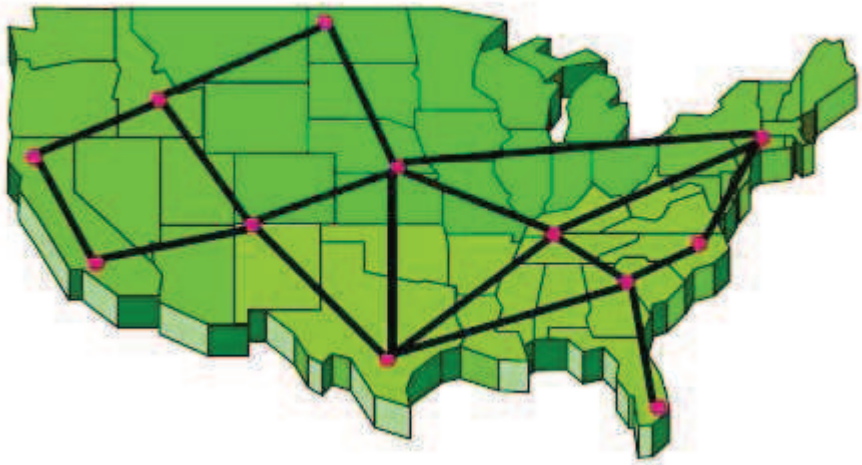


Figura 1.3 *Grafo non orientato*

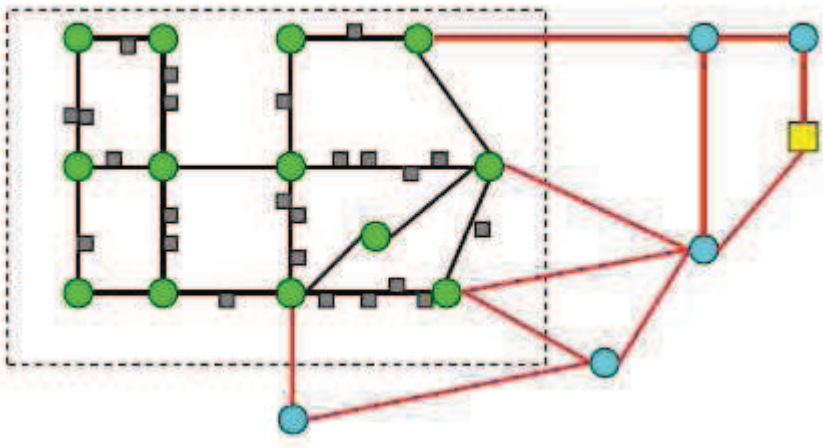


Figura 1.4 *Soluzioni ibride*

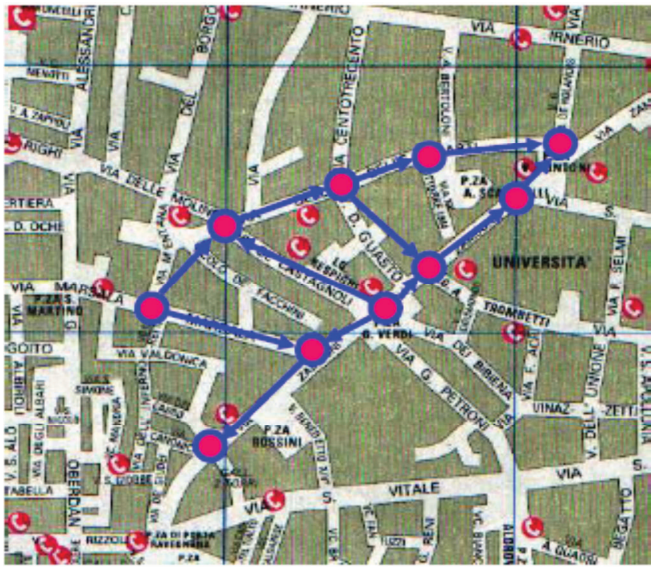


Figura 1.5 *Grafo orientato*

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 a 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;
- 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;
- 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;

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.

1.3 Descrizione del problema

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.

1.3.1 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).

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: $d(S) = \sum_{s \in S} d_s$. 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 ⊆ 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* $\lceil (S)/C \rceil$, essendo r(V \{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:

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

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:

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

al quale sono applicati i seguenti limiti:

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

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

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

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

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

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

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. 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 ≥ d(V) e K = 1 si riduce proprio ad un'istanza di TSP.

1.3.1.1 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*) ∈ 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.

1.3.2 VRP *Pickup* e *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 Oi e la destinazione Di coincidono per tutti i vertici - possono corrispondere, ad esempio, con il deposito - e non sono indicati esplicitamente.

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

1.3.3 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 perché 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 1.3.1), 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:

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

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

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 = +∞ per ogni vertice *i = 1, . . . , n* si riduce di fatto ad un'istanza CVRP, problema generalizzato da VRPTW che risulta quindi NP-difficile in senso stretto.

Ad ogni modo, nel terzo capitolo analizzerò nei dettagli il VRPTW usando qualche notazione diversa, dovuta al fatto che, poi, proporrò un modello matematico, creerò un'algoritmo e implementerò con *Aimsun.*

Capitolo 2

Simulazione e software *Aimsun*

In questo capitolo analizzerò, in generale, quali sono i concetti chiave della simulazione. Prima di tutto descriverò cos'è una simulazione ed esporrò i tipi di simulazione, passerò ai vantaggi e limiti della simulazione e ai vari modelli esistenti, per finire alla simulazione discreta e alla struttura di uno studio di simulazione. Nella seconda parte descriverò *Aimsun,* il pacchetto integrato di simulazione del traffico.

2.1 Introduzione alla simulazione

La simulazione è *l'imitazione delle operazioni eseguite nel tempo da un sistema o processo reale*. In poche parole, abbiamo definito cos'è la simulazione.

Lo scopo della simulazione l'ho possiamo dividere in più parti:

- generazione di una storia artificiale del sistema;
- studio e valutazione delle caratteristiche del sistema (analisi dettagliata);
- risposta alle domanda "*cosa accade se…*";
- *progetto*: analisi dei sistemi "ipotetici";
- confronto tra due o più sistemi;
- ottimizzazione: determinare valore ottimale di parametri;
- determinare i punti critici (bottlenecks);
- capacity planning;

- previsionale: predire le prestazioni del sistema nel futuro.

Le aree di applicazione sono le seguenti:

- sistemi di elaborazione;
- sistemi di comunicazione;
- *sistemi di trasporto e logistica*;
- sistemi di produzione e automazione;
- sistemi militari;
- sistemi sociali;
- sistemi naturali;
- sistemi economici.

2.2 Processi decisionali e modelli di simulazione

Il *processo decisionale*, cioè il processo attraverso cui, a partire dall'emergere di una situazione che richiede una scelta o una azione, si arriva alla scelta dell'azione da intraprendere e poi alla sua realizzazione, è oggetto di studio in settori notevolmente diversi che vanno dall'ingegneria all'informatica, dalla sociologia alla teoria della politica, dall'economia alle scienze gestionali.

Lo studio dei processi decisionali, la capacità di analizzarne e scomporne i meccanismi, e soprattutto la messa a punto di strumenti sia metodologici che tecnici di supporto è essenziale per pervenire a 'buone' decisioni. Spesso è il processo decisionale in se stesso che produce risultati significativi al di là delle decisioni ed azioni alle quali esso porta; questo per la sua caratteristica di essere un processo di apprendimento che in qualche modo cambia gli attori stessi in esso coinvolti.

In generale in un processo decisionale il punto di partenza è l'individuazione di una *realtà problematica* che richiede un cambiamento e quindi una decisione. La realtà così individuata

viene analizzata in modo da evidenziare al suo interno il *sistema* da studiare ai fini della o delle decisioni da prendere; vengono cioè scelti quegli elementi che ci sembrano più rilevanti, evidenziate le relazioni che li collegano, e definiti gli obiettivi da raggiungere. A questo punto si costruisce un *modello* formale che permetta di riprodurre (*simulare*) il sistema individuato, allo scopo di comprenderne il comportamento e di arrivare ad individuare le decisioni da prendere.

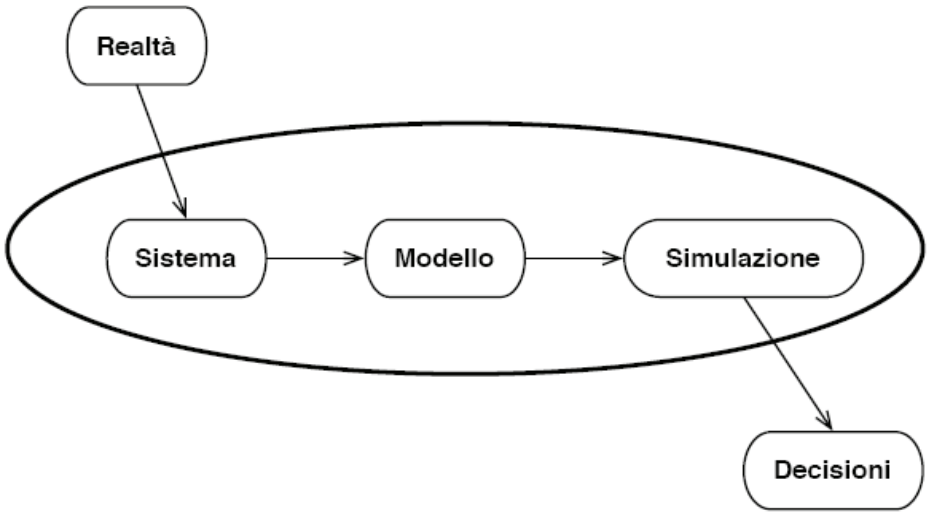


Figura 2.1 *Processo decisionale*

Il processo che abbiamo delineato parte quindi dalla realtà ed arriva alla o alle *decisioni* finali attraverso tre passi: l'individuazione del sistema da studiare, la costruzione del modello e la simulazione. Questo processo è sinteticamente rappresentato in figura 2.1, dove è stata evidenziata la parte che qui ci interessa, quella che va dalla definizione del sistema fino alla simulazione.
Esamineremo nel seguito più in dettaglio i passi del processo decisionale.

2.2.1 Il sistema

La realtà oggetto di indagine viene rappresentata attraverso un sistema, cioè un insieme di elementi interagenti fra loro. Quella di rappresentare la realtà come un sistema è una scelta, ed il risultato di tale rappresentazione è la conseguenza di una successione di scelte specifiche, tutte caratterizzate da un certo grado di arbitrarietà e quindi suscettibili di revisione nel corso del processo decisionale. La principale e più critica scelta riguarda i confini del sistema, cioè quali elementi della realtà debbano essere inseriti nel sistema che la rappresenta e quali invece lasciati fuori. Ad esempio, nello studio del traffico privato in un'area urbana bisognerà scegliere quale porzione della rete viaria considerare e dove disegnare i confini dell'area da studiare. Si tratta di scelte che riguardano aspetti fisici della realtà che sembra non pongano rilevanti problemi: alcune strade secondarie sono chiaramente poco rilevanti ai fini dei flussi principali di traffico, e appare naturale limitarsi a considerare l'area nella quale ci interessa conoscere la distribuzione del traffico. In realtà le scelte fatte anche su aspetti di questo tipo non sono neutre, e possono falsare i risultati ottenuti. La possibilità di usare alcune strade apparentemente secondarie, ad esempio, può avere effetti imprevisti sulla distribuzione del traffico. Inoltre può accadere che una parte dei veicoli in ingresso nell'area possa utilizzare più punti di accesso, e la scelta dipende dalla distribuzione del traffico dentro l'area stessa; questo porta a effetti sul traffico totale che possono non essere valutabili senza ampliare l'area sotto esame. Altre scelte riguardano quali variabili considerare e la definizione delle relazioni fra le variabili o elementi del sistema. Naturalmente in questo lavoro bisogna essere guidati dagli obiettivi che il nostro processo decisionale ha. Diversi obiettivi portano a rappresentazioni diverse della stessa realtà.

È necessario tenere sempre presente lo scarto che esiste tra il sistema e la realtà che esso rappresenta. Questo scarto può essere maggiore o minore, ma è comunque ineludibile. La realtà non `è

direttamente conoscibile se non attraverso una 'concettualizzazione' da parte dell'osservatore, e l'ottica sistemica è proprio lo strumento usiamo a questo scopo. Noi conosciamo la realtà attraverso il sistema con cui la rappresentiamo. Si tratta di una rappresentazione che dobbiamo essere sempre disponibili a rimettere in discussioni. Situazioni nuove ed impreviste ci potranno portare a rivedere il sistema che abbiamo definito, arricchendolo e modificandolo.

2.2.2 Il modello

Il modello costituisce il modo con cui noi formalizziamo il sistema che rappresenta la realtà in esame. I modelli possono essere di tipo diverso, e possono essere sia quantitativi che qualitativi. Ad esempio, modelli classici ed un tempo molto usati sono i cosiddetti *modelli a scala*. Si tratta di modelli fisici che rappresentano a scala ridotta un sistema. Tipici esempi sono il plastico di un quartiere o di un'intera città, utilizzato per scelte di tipo architettonico o urbanistico, oppure il modello a scala della struttura di un edificio che viene utilizzato per valutare la risposta della struttura a sollecitazioni ad esempio di tipo sismico. Questi sono modelli di tipo analogico; la riduzione della scala viene fatta in modo che le caratteristiche di interesse si mantengano, eventualmente riducendosi nella loro intensità secondo scala scelta. Questi modelli con la diffusione dei calcolatori elettronici sono sempre meno usati.

Un'altra classe di modelli è costituita dai *modelli analitici*. Si tratta di modelli in cui il sistema viene formalizzato attraverso un insieme di variabili e un insieme di relazioni matematiche che limitano e definiscono i valori che tali variabili possono assumere. Ad esempio una rete elettrica può essere rappresentata per mezzo di un opportuno sistema di equazioni, la cui soluzione fornisce i valori che variabili quali intensità di corrente e differenze di potenziale possono assumere. Molto spesso in questo tipo di modelli viene anche definito una funzione obiettivo da minimizzare o massimizzare. Si ricorre in questo caso ad algoritmi di

ottimizzazione. Modelli analitici sono quelli studiati nell'ambito della Programmazione Matematica (Bigi et al., 2003) oppure nell'ambito della teoria delle code.

I modelli a scala e quelli analitici sono, sia pure in forma diversa, modelli abbastanza 'rigidi' e caratterizzati da una limitata ricchezza espressiva. Possono essere usati per modellare sistemi 'relativamente' semplici[1]. Molto poco rigidi e capaci di rappresentare una grande varietà di diverse situazioni sono modelli qualitativi, quali ad esempio i diversi tipi di mappe cognitive che vengono utilizzate per rappresentare realtà problematiche nell'ambito della cosiddetta "Soft Operations Research"(Checkland, 1989; Rosenhead, 1989).

I *modelli di simulazione* sono modelli che si differenziano da quelli di tipo analitico per l'uso del calcolatore come strumento non solo di calcolo, come ad esempio nei modelli di programmazione matematica, ma anche di rappresentazione degli elementi che costituiscono la realtà in studio e delle relazioni fra di essi. La corrispondenza tra realtà e modello non è basata su una riduzione proporzionale delle dimensioni, ma è di tipo funzionale: ad ogni elemento del sistema reale corrisponde un oggetto informatico (un sottoprogramma, una struttura di dati, …) che ne svolge la funzione nel modello. Questi modelli sono particolarmente flessibili consentendo di rappresentare e di studiare sistemi molto complessi, e dei quali conosciamo alcune caratteristiche solo attraverso analisi di tipo statistico.

I modelli di simulazione sono in genere modelli dinamici, cioè includono la dimensione temporale, e hanno lo scopo di studiare l'andamento nel tempo di un sistema. Al contrario i modelli analitici sono spesso (anche se non sempre) statici: forniscono la

[1] In realtà i modelli di Programmazione Matematica consentono di affrontare problemi anche con milioni di variabili. Le difficoltà nascono soprattutto quando si è in presenza di forti nonlinearità e di situazioni caratterizzate da incertezza.

soluzione al problema studiato a partire da dati che descrivono il sistema in un dato istante o intervallo temporale[2].

2.2.3 La simulazione

La fase della simulazione vera e propria è quella conclusiva, che porterà poi alle decisioni finali. Innanzitutto il modello viene tradotto in un programma su calcolatore che viene fatto girare. In questo modo, analizzando il comportamento del modello e confrontandolo con i dati in possesso, è possibile verificare quanto il modello costruito rappresenti, correttamente rispetto ai nostri obiettivi[3], la realtà sotto studio. La correttezza può essere vista da due punti di vista diversi:

- *correttezza d'insieme* (black box validity): gli output che il modello produce riflettono accuratamente quelli del sistema reale;
- *correttezza delle singole componenti* del sistema (white box validity): le componenti del sistema sono consistenti con la realtà e/o la teoria esistente.

L'implementazione del modello può essere realizzata con diversi strumenti. È possibile usare linguaggi general purpose quali Pascal, C, C++, per i quali esistono delle librerie di routines orientate alla simulazione. Esistono anche diversi linguaggi specializzati, quali ad esempio SIMSCRIPT, MODSIM e GPSS. Un'interessante alternativa è quella di ricorrere ad applicazioni di tipo interattivo per la simulazione quali, fra gli altri, Arena, Witness, Extend e

[2] È bene avvertire il lettore che ogni tentativo di classificazione è necessariamente problematica. La realtà sfugge sempre ai nostri tentativi di racchiuderla in categorie predefinite, e questo vale anche per i modelli: esistono modelli analitici che tengono conto delle dinamiche temporali, e d'altra parte alcuni modelli di simulazione sono essenzialmente statici.

[3] Va sempre ricordato che un modello non è la realtà, e che di conseguenza non ha senso parlare di correttezza in sé del modello; la correttezza è relativa agli scopi per cui abbiamo costruito il modello.

Micro Saint. Tali applicazioni sono di facile uso e quindi molto adatte a costruire rapidamente modelli anche sofisticati, ma sono meno versatili e potenti dei linguaggi specializzati o di quelli general purpose. Per problemi di piccole dimensioni è anche possibile usare strumenti informatici di uso comune quali le spreadsheet. Tali strumenti possono essere utili quando si vuole rapidamente avere un'idea del funzionamento di una singola componente o di un sottosistema di un sistema complesso.

Una volta assicuratici della correttezza del modello inizia la parte conclusiva e fondamentale del lavoro, quella della sperimentazione che porterà alle decisioni finali. Questa fase richiede l'uso di strumenti statistici, sia per l'analisi dei dati di partenza sia per la valutazione dei risultati della simulazione. La sperimentazione deve essere realizzata in modo che l'influenza dei diversi fattori sui risultati ottenuti sia chiaramente evidenziata. Bisogna sempre tenere presente che, in ultima analisi, la simulazione è uno strumento conoscitivo.

2.3 Motivazioni ed adeguatezza

Ci sono diverse motivazioni per quanto riguarda la simulazione, in primis permette valutazioni di sistemi nelle stesse condizioni. Inoltre, la simulazione permette valutazioni di eventi e condizioni rare o rischiose e permette stime non misurabili su sistemi reali.

La simulazione, ora, non introduce overhead o alterazioni di sistema e permette una buona accuratezza. Ma quando la simulazione non è appropriata? Ci sono quattro opzioni:

- *simulazione vs metodi matematici*, per problemi risolvibili con metodi, ad esempio con soluzioni analitiche di modelli matematici;
- *simulazione vs sperimentazione diretta*, quando è più semplice ed efficiente effettuare una sperimentazione diretta

del sistema (esistente);
- *costo*, quando il costo (e tempo) di una simulazione che fornisca i risultati accurati diventa proibitivo;
- *convalida*, quando non si è in grado di convalidare il modello di simulazione.

Comunque, i *vantaggi* della simulazione possono essere molteplici. Innanzitutto, bisogna fare dei test di sistemi **prima** di investire sulle strutture. Poi, ci sono la possibilità di *espansione* e *compressione* del tempo simulato e di capire le *cause* di eventi. Le nuove politiche di gestione devono essere esplorate e valutate e, molto importante, è la *diagnostica* dei *problemi*. Le ultime due componenti sono l'identificazione di requisiti e, altra cosa importantissima, la comprensione del problema.

Di contro, gli svantaggi della simulazione possono essere divisi in due: il costo, cioè la costruzione dei modelli è lunga e non banale, e l'interpretazione dei risultati lunga e complessa. Vengono in aiuto i simulatori, con primitive e strutture dati, e l'ambiente di simulazione che includono strumenti per l'analisi dell'output.

2.4 Sistemi e modelli

I concetti di *modeling* possono essere molteplici:

- sistemi vs modelli;
- variabili di stato del sistema;
- eventi;
- entità e attributi;
- risorse;
- gestione di liste;
- attività e ritardi;
- modelli di simulazione a eventi discreti.

Dobbiamo scegliere il sistema da analizzare: abbiamo una vasta gamma di hardware, software e firmware e sceglieremo i componenti più idonei, compatibilmente ai costi accessibili. Inoltre dobbiamo fissare il criterio da usare per la valutazione delle prestazioni del sistema da analizzare. In ultimo le richieste fatte dagli utenti del sistema rappresentano il carico di lavoro (workload).

Dobbiamo fissare alcuni punti cardine: per *sistema* s'intende una collezione di componenti (elementi, entità) interdipendenti che interagiscono fra loro in accordo a specifiche predefinite. Altri punti sono la *creazione di un modello* (rappresenta la rappresentazione del sistema), e il fissare un *obbiettivo*, cioè l'apposito studio del comportamento delle relazioni fra le sue componenti.

Il processo di creazione ed uso del modello lo possiamo dividere in tre:

- definizione;
- parametrizzazione;
- valutazione.

Nella figura seguente vediamo come sistema e modello siano strettamente legati l'uno all'altro (Fig. 2.2):

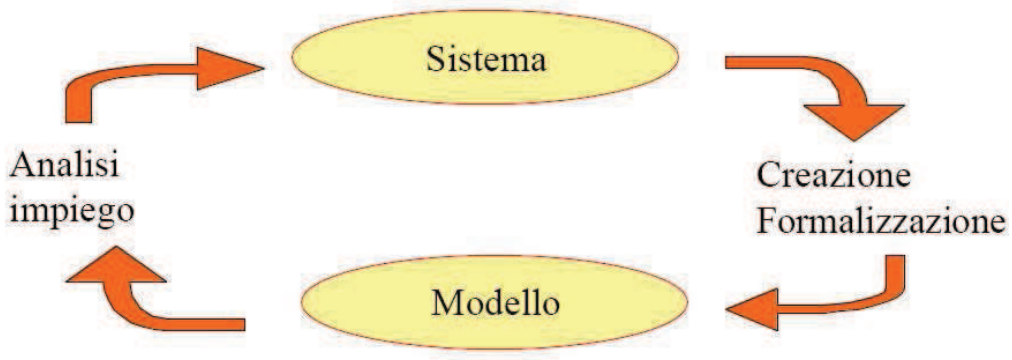


Figura 2.2 *Cooperazione modello - sistema*

Bisogna stabilire il livello di astrazione del modello, ossia quale sia il livello di dettaglio di rappresentazione. Servono, ancora, la definizione degli obbiettivi e, molto importante, l'utilizzo di modelli semplici e facilmente risolubili, specie nelle prime fasi di progetto di sistema.

La costruzione di un modello di sistema ha due parametri, la prima è la rappresentazione astratta, la seconda è la formalizzazione di osservazioni empiriche.

Il modello può avere dei vantaggi e dei limiti. I vantaggi sono i seguenti:

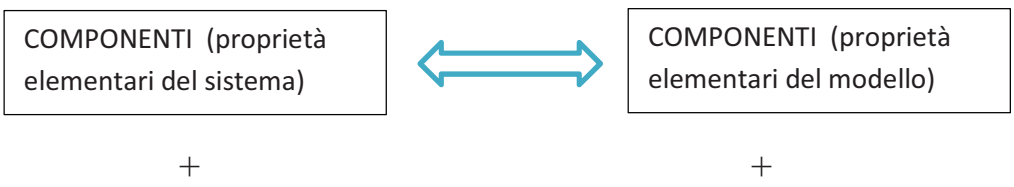
- l'organizzazione delle conoscenze e conoscenze empiriche;
- la comprensione del sistema;
- la rilevanza di componenti e/o interazioni;
- facilita l'analisi del sistema.

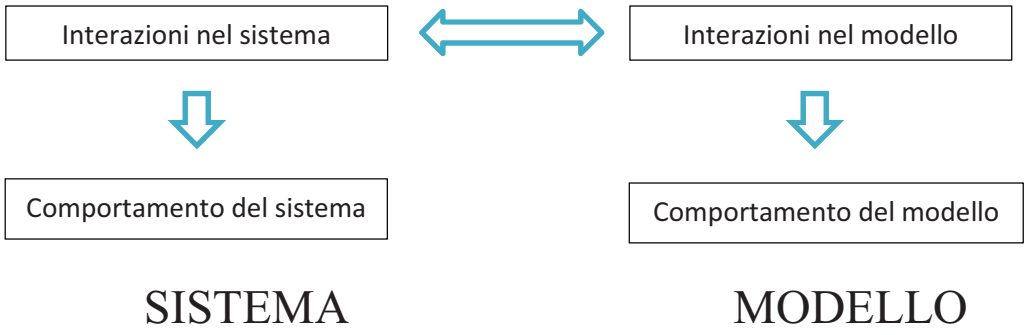
È buona norma avere un progetto per lo più *modificabile*, cioè è indispensabile averla in fase di progetto e valutare le scelte alternative prima di scartarle.

I rischi, di contro, possono essere due:

- il livello di astrazione non appropriato;
- la tendenza ad esplorare i risultati del modello oltre al suo campo di applicabilità.

Nello schema seguente è presentata la correlazione tra sistema e modello:





2.5 Modelli di simulazione

Il modello è inteso come astrazione o rappresentazione di un sistema reale. Ci sono varie caratteristiche che rendono il modello unico ma flessibile:

- l'analisi del sistema e l'isolamento delle caratteristiche da includere nel modello;
- la *complessità*, commisurata alle necessità dell'analisi;
- la *semplicità*, favorisce implementazione corretta e prestazioni del simulatore;
- la *fedeltà*, favorisce dati realmente descrittivi;
- l'approssimazione, trade-off con semplicità;
- tempo simulato vs "wall-clock time".

2.5.1 Le variabili

Le variabili di stato del sistema sono le seguenti:

- variabili che descrivono lo stato del sistema al tempo (simulato) t, e al livello di definizione sufficiente per l'analisi;
- permettono di interrompere e riprendere la simulazione;
- sono strutture dati del modello.

2.5.2 Gli eventi

Un *evento* è un cambiamento nello stato del sistema ed avviene al tempo simulato t. Gli eventi si dividono in interni (endogeni) e in esterni (esogeni). Per eventi interni intendiamo quelle variabili interne al modello, ad esempio l'inizio di un servizio di un job in coda. Per eventi esterni, invece, intendiamo quelle variabili esterne al modello, ad esempio l'arrivo di un nuovo utente in coda.

2.5.3 L'entità, gli attributi e le risorse

Le *entità* sono *oggetti* esplicitamente dal modello, possono essere dinamici (clienti) o statici (server) e, possono competere per ottenere le risorse ed essere accodati nelle corrispondenti code di attesa.

Gli *attributi* sono valori locali delle entità, ad esempio il tempo di arrivo di un cliente oppure la velocità di servizio del server.

Le risorse, infine, sono entità che forniscono *servizi passivi* o *attivi* ad entità, che possono richiedere una o più unità della risorsa. Le risorse richieste, se occupate, possono determinare un'attesa in coda e, le risorse possono fornire servizi in parallelo (server in parallelo).

2.5.4 Le liste

Le *liste* sono strutture usate per implementare le *code*. Possono essere mantenute ordinate, secondo i valori di attributi per ragioni di efficienza. In ultimo le liste hanno il compito di disciplinare l'attesa/servizio, ad esempio il "first in-first out" (FIFO) e il "last in-first out" (LIFO).

2.5.5 L'attività

L'*attività* è il periodo di tempo di durata pre-determinata caratterizzato da attività in corso, ovvero è una collezione di operazioni che trasforma lo stato di una componente. L'attività permette di fare lo scheduling di eventi di inizio/fine e, nella simulazione a eventi discreti (DES) le attività fanno avanzare il tempo simulato t.

Anch'essa importante è la *durata* dell'attività, caratterizzata dalla distribuzione statica. In ultimo, anch'esso importante, è il *ritardo* (delay), cioè la durata indefinita di un'attività, legata alle condizioni e all'evoluzione del sistema (attesa).

2.5.6 Il tempo e lo stato

Di vitale importanza per quanto riguarda i modelli di simulazione sono il *tempo* e lo *stato*. Ci sono modelli a tempo *continuo* e ci sono modelli a tempo *discreto*:

- tempo continuo (continous-time model): lo stato del sistema è definito sempre in ogni t;
- tempo discreto (discrete-time model): lo stato del sistema in istanti di tempo discreti, t, t+1,….

Ci sono modelli a stato continuo e modelli a stato discreto:

- stato continuo (continuous-state o -event model): le variabili di stato sono continue;
- stato discreto (discrete-state o -event model): le variabili di stato sono discrete e cambiano valori in corrispondenza di eventi discreti.

Per modelli a stato continuo possiamo citare, come esempio, il livello del liquido in un bacino idrico, mentre, per modelli a stato discreto possiamo citare, come esempio, il numero di persone in attesa alla posta.

2.6 Classificazione dei modelli

La classificazione dei modelli dipende, innanzitutto, dalle variabili di stato. I modelli si dividono in:

- modello *deterministico/probabilistico*,
  - deterministico: dipendenza deterministica "output (input k) = x",
  - probabilistico: almeno una variabile casuale "output(input k) = x1,x2,x3…";
- modello *statico/dinamico*,
  - statico: lo stato non dipende dal tempo;
  - dinamico: lo stato (almeno una variabile) dipende dal tempo;
- modello *lineare/non lineare*,
  - lineare: output(x) = k*x;
  - non lineare: output(x) ≠ k*x;
- modello *aperto/chiuso*,
  - chiuso: non esistono variabili esterne, ad esempio i nuovi job non entrano o escono dal sistema;
  - aperto: esistono variabili esterne, ad esempio esistono "sourse" e "sink" per i job;
- modello *stabile/instabile*,
  - stabile: il comportamento converge allo stato stazionario;
  - instabile: comportamento che cambia continuamente.

2.7 Tipi di modelli di sistema

I possibili modelli del sistema possono essere:

- modello *matematico*, *statistico*, e di *input/output*, descrivono input e output del sistema attraverso relazioni matematiche o statistiche;
- modello *descrittivo*;
- modello di simulazione a *eventi discreti*.

2.8 Tipi di simulazione

I possibili tipi di simulazione possono essere:

- ***emulazione***, simulazione che coinvolge componenti del sistema hardware e firmware. Esempi sono l'emulatore di terminale hw e l'emulatore di processore: emula un instruction set di un altro processore;
- ***Montecarlo simulation***, non esiste o non è rilevante l'asse del tempo, è utilizzata per modellare fenomeni che non cambiano nel tempo, richiede una generazione di numeri pseudo-casuali e, in ultimo, richiede anche valutazioni di espressioni non probabilistiche. Esempi sono il calcolo degli integrali, il calcolo di Pigreco…;
- ***trace driven simulation***, è una simulazione basata su una sequenza di eventi ordinati per tempo registrati dal sistema reale (traccia). I vantaggi sono molteplici, innanzitutto è credibile, non è basata su ipotesi e distribuzioni di input; è una semplice validazione e non c'è nessuna assunzione su workload. È possibile fare una buona analisi di sensitività, inoltre la variabilità è minore (varianza minore) che permette una stima accurata e/o veloce e, in ultimo, il confronto tra i

sistemi è certamente equo (fairness). Gli svantaggi, a sua volta, possono essere diversi, innanzitutto, di solito, la *complessità* del modello è maggiore; la difficoltà di ricavare una traccia rappresentativa dei carichi possibili (soprattutto per sistemi diversi e molto dinamici), chiamata *rappresentatività* e la *lunghezza*, la traccia è lunga e legata alla situazione in cui è registrata, da cui nasce il problema della scelta della parte di traccia da utilizzare. Altro svantaggio è la *convalida*, occorrono molte tracce diverse per motivare e dimostrare i risultati i risultati e, in ultimo, occorono tracce diverse per ogni diverso carico (workload);

- ***discrete-event simulation (DES)***, una simulazione che usa un modello a *tempo discreto*, può essere basata su stato continuo e stato discreto e le variabili di stato cambiano solo in corrispondenza ad eventi discreti , determinati a loro volta da attività e ritardi.

2.9 Tempo di simulazione

Il tempo si divide in:

- *reale*, ossia tempo del sistema da simulare;
- *simulato*, cioè tempo rappresentato nel modello con il clock, la variabile del modello il cui valore è il tempo simulato corrente;
- *di esecuzione*, tempo di elaborazione del programma di simulazione.

Il tempo di esecuzione dipende dalle componenti del modello e dalla complessità dei cambiamenti di stato rappresentati, e non dalla scala del tempo simulato.

La rappresentazione dell'evoluzione temporale (clock) può contrarre o espandere il tempo.

I meccanismi di avanzamento del tempo in simulazione possono essere o *per intervalli fissi* (*unit-time*) o *per eventi* (*event-driven*). Per intervalli fissi s'incrementa il clock di una quantità fissa Δ e si esamina il sistema per determinare gli eventi che devono aver luogo per i quali si effettuano le necessarie trasformazioni. Tratta tutti gli eventi con tempo di occorrenza $t_i \in (t, t + \Delta]$, è interessante la scelta dell'incremento Δ, gli eventi con diversi tempi di occorrenza possono essere trattati come eventi simultanei e ci sono possibili intervalli vuoti.

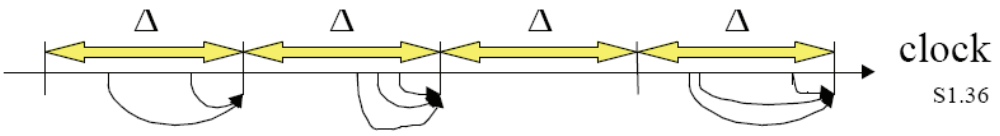


Figura 2.3 *Intervalli fissi del tempo*

Per i meccanismi di avanzamento del tempo in simulazione per eventi, si incrementa il clock fino al tempo di occorrenza del prossimo evento. Sono incrementi generalmente irregolari, qualche volta ci sono eventi simultanei solo se hanno lo stesso tempo di occorrenza e, molto importante, evita tempi di inattività.

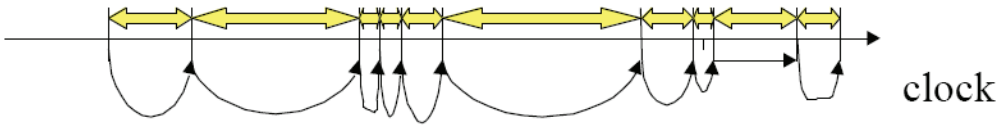


Figura 2.4 *Tempo per eventi*

2.10 Schemi di simulazione

Le strutture di modelli di simulazione sono essenzialmente quattro:

- *interazione tra processi*;
- *scheduling di eventi*;
- *scansione di attività*;

- *metodo a "tre fasi"*.

Il flusso di esecuzione di un processo in esecuzione emula il flusso di un oggetto (entità) attraverso il sistema: è la ***simulazione tra processi***. L'esecuzione procede finché il flusso non viene bloccato, ritardato, terminato o inizia una nuova attività. Esempi possono essere l'attesa in coda e il servizio (ritardo). Quando il flusso di un'entità viene bloccato , il tempo di simulazione avanza al tempo di inizio previsto dalla prima successiva entità in esecuzione.

La simulazione, invece, per ***scheduling di eventi*** si presta molto bene: infatti si avanza il tempo simulato al tempo del prossimo evento successivo (di solito il termine o l'inizio di un'attività). Viene "stilata" una lista ordinata di eventi e viene fissato uno scheduler di eventi. Il termine di un'attività coincide con la nuova allocazione di risorse rilasciate tra le entità in attesa e con lo scheduling di nuove attività causalmente determinate. Abbiamo una serie di prestazioni tipiche ad uno scheduler di eventi:

- mantiene la struttura di ordinata per tempo simulato (multi-linked) di eventi futuri (schedulati, cancellati, rinviati, bloccati);
- gestisce l'avanzamento del tempo simulato,
  - event-driven: clock ← tempo del prossimo evento;
  - unit-time: clock ← clock + Δ (sono accaduti eventi?);
- struttura dati Evento = (time, puntatore al codice della routine di evento);
- la routine di evento aggiorna le variabili di stato e aggiorna la lista di eventi (inserisce, cancella, o rinvia eventi);
- routine di inizializzazione: chiamate per prime, definiscono lo stato iniziale del sistema, le sequenze dei numeri pseudo-casuali e così via;
- routine di gestione degli eventi;

- report generator, cioè sono procedure che al termine della simulazione generano i dati di stima delle grandezze di interesse;
- routine di trace, sono procedure per notificare eventi o stime a run-time;
- gestione dinamica della memoria e garbage collection.

La simulazione per scansione attività è simile alla rule-based *programming* ed è chiamata a "due fasi", condizione ok -> esecuzione azione. Esiste un insieme di moduli in attesa di esecuzione, uno per attività e l'avanzamento del tempo lo si fa a intervalli fissi. Periodicamente si esegue un test sulle condizioni che determinano l'esecuzione di eventi, esecuzione di eventi -> aggiornamento delle variabili di stato.

L'ultima simulazione è la metodologia in "***tre fasi***" (anche qui l'avanzamento del tempo è a intervalli fissi):

- ***fase 1***: avanzamento del tempo simulato;
- ***fase 2***: rilascio delle risorse mantenute dalle attività che risultano terminate dopo l'avanzamento;
- ***fase 3***: esecuzione delle attività per le quali siano disponibili le risorse.

2.11 Pianificazione di uno studio di simulazione

La pianificazione di uno studio di simulazione viene presentata sotto forma di diagramma di flusso (Fig. 2.5), suddivisa in *dieci punti*.

Al ***punto 1*** abbiamo la ***formulazione del problema***: bisogna definire e verificare la corretta e completa formulazione del problema attraverso un dialogo tra cliente e analista e bisogna definire le assunzioni delle specifiche e delle definizioni del sistema adottate.

Al ***punto 2*** ci sono la ***definizione degli obiettivi*** e la preparazione di una proposta. Come obiettivi si devono definire le risposte attese dalla simulazione, gli scenari oggetto di indagine, i tempi necessari, le risorse necessarie, i costi e le fasi dell'analisi.

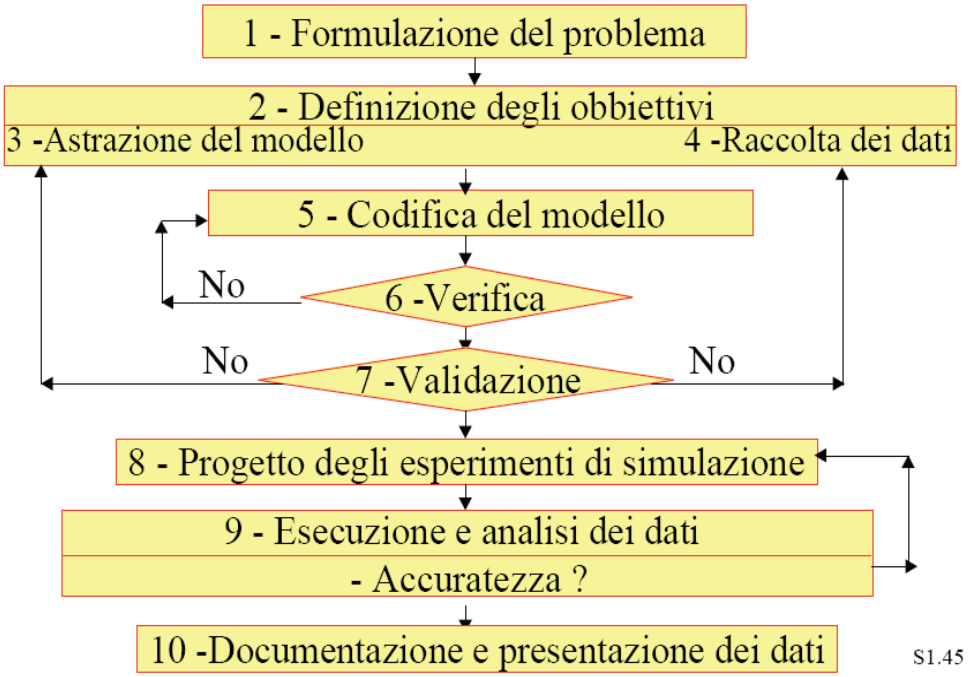


Figura 2.5 *Pianificazione di uno studio di simulazione il diagramma di flusso*

Nel ***punto 3*** abbiamo ***l'astrazione del modello concettuale***, con le relazioni logiche, matematiche e causali delle componenti e strutture del sistema. Dobbiamo poi scegliere il livello di astrazione del modello, i componenti, gli attributi, le caratteristiche funzionali e le relazioni tra di esse. In ultimo, bisogna scegliere la complessità strutturale e la solubilità. In fase progettuale si fa riferimento alla *metodologia Top-Down*: partire dalle cose semplici ed estendere il modello fino a raggiungere il grado di complessità descrittiva necessario. Un appunto, la complessità descrittiva non è necessaria all'analisi: aumenta il rischio di errori, costi e tempi. Necessario è il coinvolgimento del "cliente" nella costruzione del modello.

Al ***punto 4*** abbiamo la ***raccolta dei dati***: abbiamo la caratterizzazione del carico (workload characterization), l'analisi e l'indagine sui dati che caratterizzano il modello , le discipline di accesso e servizio, i tempi medi di percorrenza, attesa e servizio, le distribuzioni di probabilità e le tecniche di misurazione.

Al ***punto 5*** abbiamo la ***codifica del modello***: il modello concettuale ottenuto viene codificato in una forma interpretabile dal calcolatore, ottenendo un modello operazionale. Ci sono tre tecniche di codifica:

- approccio object-oriented;
- simulazione distribuita;
- ambienti di simulazione.

Ai ***punti 6*** e *7* abbiamo ***la verifica*** e la ***validazione del modello***: vi è la verifica della correttezza del modello operazionale (soluzione) del modello, la convalida del modello concettuale come adeguata rappresentazione del sistema e la convalida dell'adeguatezza di modelli, obiettivi e risultati. Possibile iterazione ai punti 2, 3, 4 e 5 con le relative modifiche.

Al ***punto 8*** c'è il ***progetto degli esperimenti di simulazione***: vengono definiti la lunghezza della simulazione, il numero di prove, la simulazione a termine o stazionaria e gli esperimenti pilota.

Al ***punto 9*** abbiamo l'***esecuzione*** e l'***analisi dei dati*** (analisi di sensitività): ci sono, quindi, la scelta del metodo di analisi, gli intervalli di confidenza e stimatori (feedback al punto 8), l'interpretazione dei risultati e un'analisi comparativa.

Infine, al ***punto 10*** abbiamo la ***documentazione*** e la ***presentazione dei dati***: devono essere presentati gli obiettivi, le assunzioni, il modello, le tecniche di analisi dei risultati e i risultati degli esperimenti.

2.12 Modello di simulazione del traffico: AIMSUN

Aimsun è un software integrato di modellazione dei trasporti, sviluppato e commercializzato da TSS - sistemi di simulazione di trasporto con sede a Barcellona, Spagna.

Il software Aimsun è usato da agenzie governative, comuni, università e consulenti in tutto il mondo per l'ingegneria del traffico, simulazione del traffico, pianificazione dei trasporti, logistica e studi per veicoli improvvisi e momentanei (ad esempio, un incidente). Viene utilizzato per migliorare le infrastrutture stradali, ridurre le emissioni, tagliare la congestione e gli ambienti di progettazione urbana per i veicoli e pedoni. Aimsun è attualmente uno dei leader di mercato nel software di simulazione dei trasporti, con oltre 2.200 utenti con licenza in oltre 60 paesi. È utilizzato anche in tempo reale, previsioni di traffico on-line come Aimsun online. Il prodotto è l'unico strumento attualmente sul mercato che integra in un unico pacchetto software tre livelli di simulazione del traffico veicolare più utilizzati al mondo: macro, meso e micro-simulazione. Questo permette la visualizzazione 2D e 3D di scenari di trasporto nelle principali città mondiali a livelli di dettaglio e di qualità.

L'***assegnamento macroscopico del traffico*** tratta le seguenti funzionalità: esse comprendono l'assegnazione del traffico su una rete a livello macroscopico, la regolazione di matrici origine-destinazione (OD) in modo da riflettere i dati reali, e la valutazione e la raccomandazione del rivelatore di layout e delle posizioni.

Le simulazioni mesoscopiche vengono utilizzate dai professionisti per modellare aspetti dinamici delle reti di grandi dimensioni. I principali concorrenti commerciali, Vissim e Paramics, hanno funzionalità limitate alla sola micro-simulazione, quindi necessitano di strumenti accessori per portare a termine analisi macroscopiche o mesoscopiche su una medesima rete; la necessità di ulteriori elementi di lavoro rende poco efficiente lo

sviluppo e la messa a punto di reti complesse in cui, dopo aver portato a termine un'assegnazione statica dei flussi veicolari sugli archi, rivestono un ruolo essenziale lo studio e la risoluzione delle criticità che vengono a formarsi all'interno della rete.

La ***nuova funzionalità*** di analisi ***mesoscopica***, introdotta dalla versione 6 di Aimsun, è la novità assoluta che i concorrenti non sono in grado di fornire; tale approccio permette di modellare l'aspetto dinamico di rete molto estese, utilizzando un motore di micro-simulazione semplificato, sia per il modello di lane-changing che di car-following, Le sezioni vengono riprodotte con collegamenti in cui si ha un deflusso libero, mentre le criticità, valutate ad ogni passo della simulazione, vengono concentrate nei nodi. Gli elementi utilizzati nella meso-simulazione consentono di ottenere risultati molto affidabili senza la necessità di accurate calibrazioni, come dimostrano i confronti con la micro-simulazione effettuati dagli sviluppatori di Aimsun 6.

I ***modelli di micro-simulazione*** in Aimsun rappresentano un valido strumento a disposizione dei tecnici e dei decisori nel settore dei trasporti per la valutazione degli effetti di scelte progettuali alternative. Tali modelli consentono, in modo particolare, analisi di dettaglio delle soluzioni pianificate a livello locale. Ad esempio è di grande supporto nella verifica di soluzioni di nodi complessi di intersezioni regolate con semaforizzazioni sincronizzate, attuate dal traffico tramite sensori, intersezioni a rotatoria ecc.

Grazie a tali strumenti è possibile fornire ai decisori:

- una chiara visualizzazione anche tridimensionale del territorio nel quale si inseriscono i progetti;
- gli elementi quantitativi utili alla valutazione del deflusso veicolare, pedonale, ciclistico;
- le stime di dettaglio sulle lunghezze delle code, dei relativi tempi, delle velocità medie e in sintesi delle prestazioni dei singoli componenti del sistema della viabilità;

- le stime di emissioni inquinanti atmosferiche e ambientali, consumi energetici e di carburante;
- visualizzare in modo realistico il movimento delle singole componenti del traffico, a partire dai pedoni, ai ciclisti, alle moto, ai veicoli di tutte le tipologie, ai sistemi di trasporto pubblico (bus, taxi, tram, treno).

Alcune tra le possibili applicazioni di un modello di micro simulazione sono quindi:

- la valutazione di nuove infrastrutture (strade, rotatorie, svincoli, ecc.);
- l'ottimizzazione e la verifica nella regolazione dei sistemi di controllo semaforico (controllati, attuati e sincronizzati);
- la valutazione degli effetti dell'introduzione di sistemi di circolazione (Zone a traffico regolato, sensi unici, corsie riservate, Zone a traffico limitato, ecc.);
- la valutazione di efficacia e di efficienza dei sistemi di trasporto pubblico;
- l'ottimizzazione e integrazione dei servizi di trasporto pubblico, della viabilità ordinaria, pedonale e ciclistica.

Aimsun è in grado di supportare il processo decisionale e il confronto delle alternative sia per l'accuratezza nei risultati (indicatori di traffico e ambientali), sia per la semplicità e forza espositiva di un ambiente grafico avanzato (anche tridimensionale). In particolare sono disponibili funzionalità fondamentali per i migliori risultati in termini rappresentativi, quali ad esempio:

- importazione di mappe digitali dai sistemi CAD e GIS;
- esportazione del modello in ambiente GIS e KML (Google Earth);

- interfaccia per il trattamento di tabelle OD con qualunque programma di Foglio Elettronico;
- trattamento delle tabelle OD per lo studio locale a partire da matrici di area;
- esportazione dei risultati in qualunque formato database per successive analisi personalizzate.

I linguaggi Python e **C++** sono utilizzati da Aimsun. In figura 2.6 abbiamo un esempio di micro simulazione:

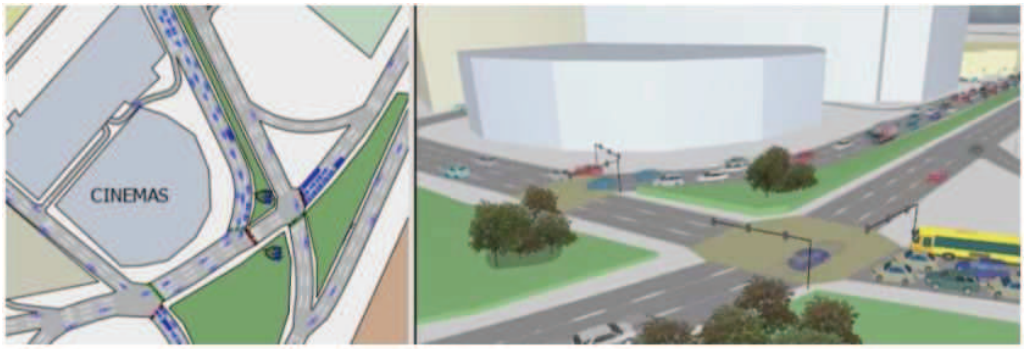


Figura 2.6 *Immagine in Aimsun 2D e 3D*

Capitolo 3

Modello VRP con TW e implementazione in *Aimsun*

In questo capitolo presento un algoritmo di Vehicle Routing Problem con finestre temporali: descrivo un modello matematico e presento l'algoritmo sotto forma di diagramma di flusso. Questa è una parte molto delicata e implemento il tutto mediante l'uso dell'ambiente Matlab.
Successivamente, devo implementare l'algoritmo creato mediante l'uso del software Aimsun con il programma C++: presenterò i principali metodi del sistema.

3.1 Un modello matematico per il VRP con TW

Andiamo a descrivere il modello matematico che, poi, ho utilizzato per l'implementazione del mio algoritmo.

Il VRPTW è dato da una flotta di veicoli rappresentato da un insieme **F**, un insieme di clienti **N**, con un grafico orientato che connette i clienti al deposito. Posta la dim(N)=**n**, il grafico contiene n+1 nodi; i clienti sono rappresentati dai vertici 1,2,...,n e il deposito ha vertice 0.

L'insieme degli archi **A** rappresenta la connessione tra il deposito e il cliente e tra cliente e cliente. Ad ogni arco (i, j) del grafico è associato un costo metrico c_{ij} che va a formare la matrice **C** e un tempo di percorrenza t_{ij} che va a formare la matrice **T**.

Ciascun veicolo della flotta ha una capacità q (la stessa per ogni veicolo).

A ciascun cliente $i \in N$ sono associate le time window $[a_i, b_i]$ che rappresenta l'intervallo in cui il servizio del cliente è attivo, una *domanda* d_i e una *service time* s_i. Queste time windows (finestre temporali) sono "hard", nel senso che un veicolo i deve cominciare prima del tempo b_i e nel caso in cui il veicolo arriva prima del tempo ai deve aspettare (senza penalità). Il deposito, inoltre, ha la time window $[a_0, b_0]$. I veicoli devono uscire dal deposito in a_0 e ritornare necessariamente in b_0.

Assumiamo c_{ij}, a_i, b_i, d_i, s_i non negativi, e t_{ij} positivo, per ogni $i \in N$ e per ogni arco $(i, j) \in A$.

Il modello codice due *variabili decisionali*, **x** e **y**. Per ogni arco $(i, j) \in A$, quando $i, j \in N$, $i \neq j$, per ogni veicolo $k \in F$, definiamo la componente x_{ij}^k di x:

$$x_{ij}^k = \begin{cases} 1, sse\ (i,j) \in A \text{ è } nel\ route\ che\ il\ veicolo\ serve, con\ k \in F \\ 0\ altrimenti \end{cases}$$

La componente y_i^k della variabile decisionale y è definita per ogni vertice $i \in N$ e per ogni veicolo $k \in F$, rappresenta il tempo in cui il veicolo k comincia a servire il cliente i. Nel caso in cui il veicolo k non serve il cliente i, y_i^k non ha significato.

L'obiettivo è costruire un insieme di percorsi a costo minimo, uno per ogni veicolo, con queste condizioni:

- ogni cliente è servito una sola volte esattamente;
- ogni percorso comincia dal deposito;
- le time windows sono rispettate;
- i vincoli della capacità sono rispettati.

Così il VRPTW può essere matematicamente descritto

$$\min \sum_{k \in F} \sum_{(i,j) \in N} c_{ij} x_{ij}^{k} \qquad (3.1)$$

al quale sono applicati i seguenti vincoli:

$$\sum_{k \epsilon F} \sum_{j \epsilon V} x_{ij}^{k} = 1 \ \ \forall \, i \ \in N \qquad (3.2)$$

$$\sum_{i \in N} d_i \sum_{j \in N} x_{ij}^{k} \ \leq \ q \ \forall k \in F \qquad (3.3)$$

$$\sum_{j \in N} x_{0j}^{k} = 1 \ \ \forall k \in F \qquad (3.4)$$

$$\sum_{i \in N} x_{ih}^{k} - \sum_{j \in N} x_{jh}^{k} \ = 0 \ \ \forall h \in N, \forall k \in F \qquad (3.5)$$

$$\sum_{i \in N} x_{i,n+1}^{k} = 1 \ \ \forall k \in F \qquad (3.6)$$

$$x_{ij}^{k}\left(y_i^{k} + s_i \ - t_{ij}\right) \leq 0 \ \ \forall (i,j) \in N, \forall k \in F \qquad (3.7)$$

$$a_i \ \leq y_i^{k} \leq b_i \ \ \ \forall i \in N, \forall k \in F \qquad (3.8)$$

$$x_{ij}^{k} \in \{0,1\} \ \ \forall i,j \in N, \forall k \in F \qquad (3.9)$$

La funzione obiettivo (3.1) ha un preciso obiettivo: minimizzare il costo metrico totale. Il vincolo (3.2) dice che per ogni cliente deve essere servito una sola volta. Il vincolo (3.3) afferma che un veicolo non può essere servito quando è minore la capacità. Le tre equazioni (3.4), (3.5) e (3.6) sono vincoli di flusso e assicurano, rispettivamente, che per ogni veicolo $k \in F$ parte dal deposito, dopo aver servito un cliente parte e si ferma in un altro cliente, e, infine, ritorna al deposito.

L'ineguaglianza (3.7) dice che un veicolo k che sta viaggiando dal cliente i al cliente j non può arrivare lì prima della somma data

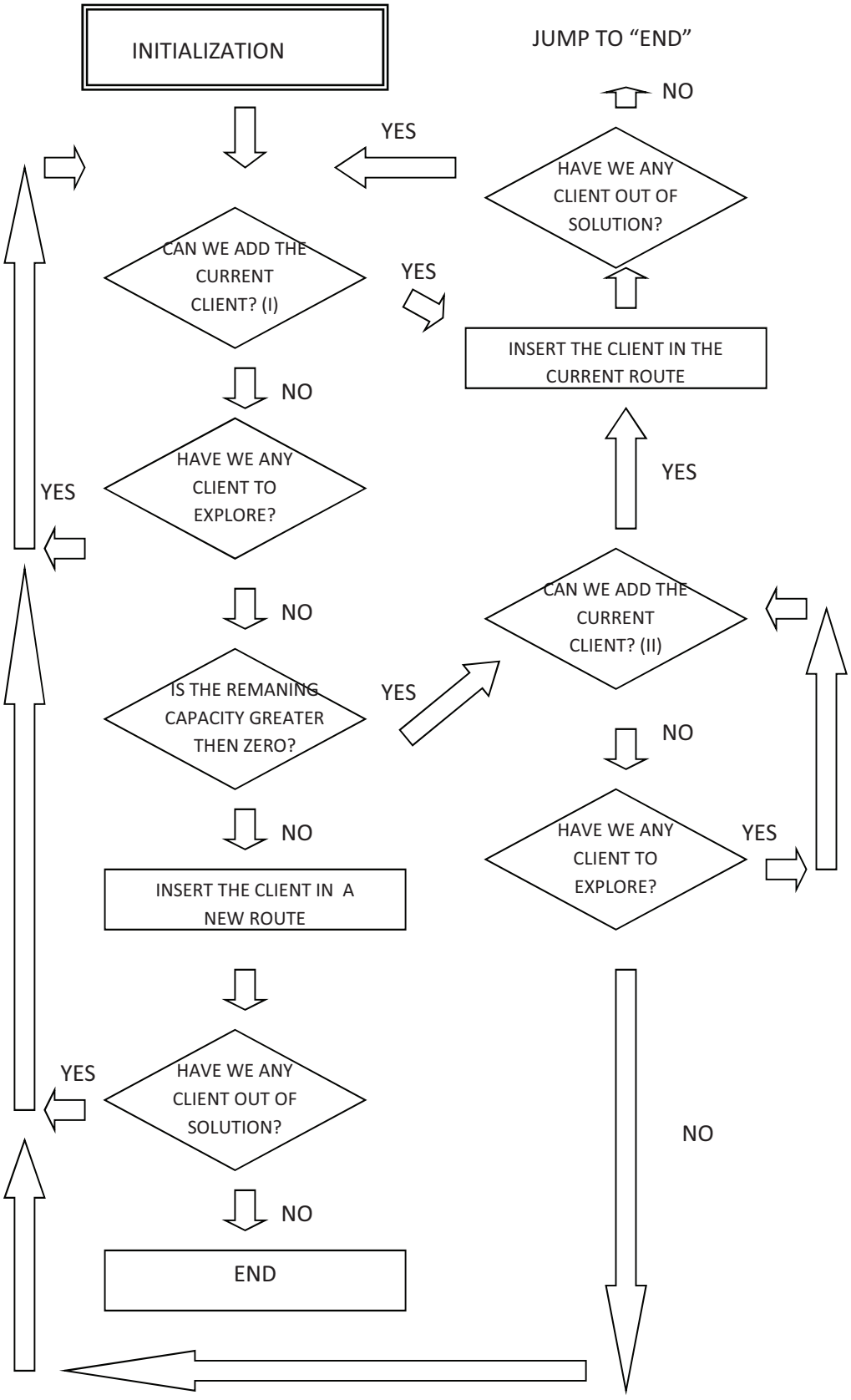
dalla partenza da *i* e dal tempo di viaggio t_{ij}. Infine, il vincolo (3.8) assicura che tutte le finestre temporali sono rispettate, mentre il vincolo (3.9) è dato dall'interezza sulla variabile decisionale x.

3.2 L'algoritmo

Il flow chart dell'algoritmo è visualizzata nella figura sottostante. La descrizione dettagliata di ogni blocco verrà data successivamente.
L'obiettivo dell'algoritmo è quello di minimizzare il costo metrico totale. Considero questi dati degli input:

1. *il numero di clienti (n);*
2. *il numero di veicoli nella flotta (k);*
3. *la capacità di ogni veicolo nella flotta (q);*
4. *la domanda di ogni cliente (d_i);*
5. *la matrice dei costi metrici (C);*
6. *la matrice dei tempi di viaggio (T);*
7. *la time windows di ogni cliente (con il tempo di partenza (a_i) e il tempo d'uscita (b_i));*
8. *il service time di ogni cliente (s_i).*

Ho chiamato ***inizializzazione*** il primo step. Dopo c'è il secondo step nella quale cerco di inserire un nuovo cliente nella route, se c'è compatibilità con le time windows e con la capacità. Se non c'è nessun cliente che può essere inserito, ho due strade: se la capacità del veicolo corrente della flotta è più grande di zero nella prima, la seconda nell'altro caso. Nel primo caso utilizzo una nuova condizione per inserire un cliente: se la *opening window* del cliente è dopo il tempo di arrivo del mio veicolo, inserisco questo cliente nella soluzione ma so che il veicolo dovrà aspettare prima che incominci il servizio. Nel secondo caso creo un nuovo percorso e inserisco il cliente.


INITIALIZATION
CAN WE ADD THE CURRENT CLIENT? (I)
YES
NO
HAVE WE ANY CLIENT TO EXPLORE?
YES
NO
IS THE REMANING CAPACITY GREATER THEN ZERO?
YES
NO
INSERT THE CLIENT IN A NEW ROUTE
HAVE WE ANY CLIENT OUT OF SOLUTION?
YES
NO
END
HAVE WE ANY CLIENT OUT OF SOLUTION?
YES
NO
JUMP TO "END"
INSERT THE CLIENT IN THE CURRENT ROUTE
YES
CAN WE ADD THE CURRENT CLIENT? (II)
NO
HAVE WE ANY CLIENT TO EXPLORE?
YES
NO

Quando ho inserito un nuovo cliente devo verificare se ho inserito tutti i clienti allo scopo di andare al **blocco fine** in cui ho la soluzione.

3.2.1 L'inizializzazione dell'algoritmo

Nell'inizializzazione voglio creare una nuova matrice in cui ogni riga di indice corrisponde al deposito (riga 0) o ad un cliente (gli altri) e gli elementi di colonne sono gli altri nodi dal più vicino al più lontano dal nodo considerato. Chiamo questa *matrice E*.
Naturalmente, per fare ciò, devo usare la matrice di costi metrici, C. prendo ogni riga, dispongo l'elemento in ordine crescente e inserisco la riga sulla nuova matrice riconoscendo che il cliente corrisponda al costo. Per ordinare l'elemento della riga uso questo algoritmo (considero ogni riga come un vettore):

1. metti l'indice *i* sul primo elemento del vettore;
2. fino a quando ci sono almeno due elementi dall'indice *i* alla fine del vettore, fai:
   2.1 trova il più piccolo elemento da *i*+1 alla fine del vettore: ricorda il vettore j dal più piccolo elemento trovato;
   2.2 se il più piccolo elemento trovato (indice *j*) è più piccolo dell'elemento di indice *i*, scambia i due elementi;
   2.3 incrementa *i*;
3. alla fine abbiamo un vettore ordinato.

Abbiamo la matrice C. La prima riga corrisponde al deposito, così trovo il primo cliente da inserire nella soluzione. Naturalmente, comincio a esplorare dal cliente più vicino al deposito (il secondo elemento della riga).
Nell'esempio viene trasformata dalla matrice C alle matrice E.

$$C = \begin{pmatrix} 0.0 & 1200.2 & 1578.8 & 1717.4 \\ 949.5 & 0.0 & 932.7 & 1071.3 \\ 779.8 & 544.7 & 0.0 & 215 \\ 647.8 & 412.7 & 791.4 & 0.0 \end{pmatrix} \Longrightarrow E = \begin{pmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 0 & 3 \\ 2 & 3 & 1 & 0 \\ 3 & 1 & 0 & 2 \end{pmatrix}$$

È necessario vedere se c'è compatibilità per non violare le time windows e rispettare il vincolo sulla capacità, che corrisponde alle diseguaglianze (3.3) e (3.7) del modello matematico. Introduco due nuove variabili chiamate *time* e *capacity*. All'inizio il time è inizializzato ad a_0, il tempo d'inizio del deposito, e la capacity è uguale a zero. Per vedere la fattibilità d'inserimento dobbiamo verificare se il tempo della variabile time più il tempo di viaggio tra il deposito e il cliente sia maggiore o uguale dell'inizio delle finestre temporali del nuovo cliente e minore o uguale alla chiusura. Infine dobbiamo verificare che la variabile capacity più d_j sia minore o uguale a q (con $j \in N$).

$$time + t_{0j} \geq a_j$$
$$time + t_{0j} \leq b_j$$
$$capacity + d_j \leq q$$

L'inizializzazione termina quando ho trovato un cliente da inserire nella soluzione.

In questo caso sommiamo la variabile time, il service time e t_{0j} (rispetto, in questo caso, il vincolo (3.7)), e la variabile capacity viene aggiornata con la d*j*. Introduco anche un vettore chiamato *bool* con dimensione pari al numero dei clienti in cui tutti gli elementi, inizialmente, uguali a zero. Al vertice *j*-esimo vado a settare la variabile a '1'. L'elemento bool è simile a una variabile di tipo booleano: l'elemento 'falso' corrisponde all'intero '0' mentre l'elemento 'true' corrisponde all'intero '1'. Con quel vettore mi

assicuro che ogni cliente sarà servito esattamente una volta (vincolo 3.2 del modello matematico).

$time = time + t_{0j} + s_j$
$capacity = d_j$
$bool[j] = 1$

Se il cliente attuale non può essere inserito nella soluzione, esploro il prossimo elemento della riga 0. Quando arrivo alla fine senza aver inserito una soluzione, inserisco nella soluzione l'elemento e_{01}. Ora il time contiene a_1 più il service time, la capacity è uguale a d_1.

$time = a_1 + s_1$
$capacity = d_1$
$bool[1] = 1$

3.2.2 L'esecuzione dell'algoritmo

Dopo l'inizializzazione, voglio trovare un nuovo cliente da inserire in soluzione. Adesso l'elemento da esplorare è la riga corrispondente a quella del vecchio cliente servito. Naturalmente, saltiamo il caso in cui il prossimo cliente è in realtà il deposito (vertice 0) e, consideriamo il prossimo elemento perché vogliamo avere anche il numero minimo di veicoli.

Nella figura 3.1 abbiamo la condizione *"Can we add the current client (I)?"*: questa condizione è basata nel verificare che la variabile *time* più il *tempo di viaggio* tra il vecchio cliente *i* e il nuovo *j* sia uguale o più grande dell'apertura della finestra temporale (time window) e minore o uguale alla chiusura b_j. Naturalmente devo verificare che la variabile *capacity* più d_j sia minore o uguale a q, e devo anche verificare che il cliente nuovo è già stato servito.

$time + t_{ij} \geq a_j$
$time + t_{ij} \leq b_j$
$capacity + d_j \leq q$
$bool[j] \neq 1$

Se il cliente nuovo può essere servito, aggiorno la variabile *time* in aggiunta al *tempo di viaggio* tra il vecchio cliente ed il nuovo e al *service time* del nuovo cliente. Dopo, aggiorno la variabile *capacity* con d_j e la variabile *bool*, e esploro il prossimo cliente nella riga corrispondente all'ultimo cliente inserito.

$time = time + t_{ij} + s_j$
$capacity = capacity + dj$
$bool[j] = 1$

Nel caso in cui il nuovo cliente non può essere servito, passo al prossimo cliente: se questi vincoli rispettano tutti i requisiti faccio la procedura standard, se no, cerco un nuovo cliente.

Se arrivo alla fine della riga senza nessun inserimento del cliente, mi sposto sulla *capacity*. Se $q - capacity$ è più grande di zero, andiamo nella condizione *"Can we add a client (II)?"*: questa nuova condizione sa ad esplorare la capacity come la (I), ma adesso la variabile *time* più il *tempo di viaggio* tra il vecchio cliente *i* e il nuovo *j* può essere minore di a_j (rilassiamo la condizione 3.8). Il prossimo cliente non deve essere già stato servito, naturalmente.

$tempo + t_{ij} < a_j$
$capacity + d_j \leq q$
$bool[j] \neq 1$

Naturalmente aggiorno la variabile *capacity*, *bool* e *time* (con il tempo di viaggio tra i due clienti, con il tempo di attesa calcolato, e

con s_j). Se *q – capacity* è minore di 0, creiamo una nuovo percorso: significa che c'è un nuovo veicolo parte dal deposito. Il cliente inserito è la prima riga che non è stata servita e le variabili *time* e *capacity* sono resettate e re inizializzate.

3.2.3 La terminazione dell'algoritmo

Quando ho inserito un cliente nella soluzione, aggiorno una variabile chiamata *Served_nodes*. Prima di continuare l'esecuzione dell'algoritmo, devo verificare se ho già servito tutti i clienti. In questo caso l'algoritmo termina e ho la soluzione definitiva.

```
%implementazione algoritmo: cammini necessari(k)
function[j]=algoritmo_mod2(C,T,A,B,S,W,Wmax,start,stop)
D=prova(C(:,1),10^5)%D è il vettore con i costi metrici ordinati
%faccio in modo da far partire il turno in modo da essere concomitante con
%l'inizio
   a=size(C(:,1))
   b=zeros(1,a(1));
   %disp(T(D(1),1));
   %disp(S(D(1)));
   %t=start+T(D(2),1)+S(D(2))
   %disp(t);
   %disp('w:');
   %w=Wmax-W(D(2))
   k=1;%indice di colonna
   %K(1,1)=D(2);
   b(1)=1;
   %b(2)=1;
   j=1;%indice di riga
   h=2;%indice cliente di partenza
   z=1;%numero di cammini
   stato=0;
   c=0;
   v=0;
   n=0;
   p=0;
   min=0;
   x=0;
```

```
while(c==0)
switch stato
   case 0
      disp('sono nello stato 0');
      p=0;
      o=2;
      min=A(D(2))-start-T(D(2),D(1));
      minind=2;
      iterator=2;
      while p==0
         if A(D(o))-start-T(D(o),D(1))<0,3
            K(1,1)=D(o);
            b(o)=1;
            w=Wmax-W(D(o));
            t=start+(A(D(o))-start)+T(D(o),D(1))+S(D(o));
            h=o;
            p=1;
         else
            if A(D(o))-start-T(D(o),D(1))<min
               min=A(D(o))-start-T(D(o),D(1));
               minind=D(o);
               iterator=o;
            end
            o=o+1;
         end
         if o==a(1)+1
           p=1;
           K(1,1)=minind;
           b(iterator)=1;
           w=Wmax-W(minind);
           t=start+(A(minind)-start)+T(D(minind),D(1))+S(D(minind));
           h=iterator;
         end
         h
         stato=1;
      end

   case 1
      disp('sono nello stato 1');
      for i=h:1:a(1)
         disp(i);
         disp('è un i');
         if i<a(1)
```

```
            switch(b(i))

                case 0
                    disp(t);
                    w
                    if t+T(D(i),D(h))>=A(D(i)) && t+T(D(i),D(h))<=B(D(i)) &&
t+T(D(i),D(h))+S(D(i))<stop
                        %disp('3it');
                        disp(w-W(D(i)));
                        if w-W(D(i))>=0
                            %disp('2it');
                            t=t+T(D(i),D(h))+S(D(i));%ora in cui parto, quanto ci
metto ad arrivare e quanto imiego er il servizio
                            k=k+1;
                            w=w-W(D(i));
                            K(j,k)=D(i);
                            b(i)=1;
                            h=i;
                            D(i)
                            disp('aggiunto');
                        end
                    end
                end
            else
                switch b(i)
                    case 0
                    disp('ci sono?');
                    t
                    w
                    if t+T(D(i),D(h))>=A(D(i)) && t+T(D(i),D(h))<=B(D(i)) &&
t+T(D(i),D(h))+S(D(i))<stop
                        %disp('4it');
                        disp('I if');
                        if w-W(D(i))>=0
                        %disp('2it');
                        t=t+T(D(i),D(h))+S(D(i));%ora in cui parto, quanto ci metto
ad arrivare e quanto imiego er il servizio
                        k=k+1;
                        w=w-W(D(i))
                        K(j,k)=D(i);
                        b(i)=1;
                        h=i;
                        D(i)
                        disp('aggiunto');
                        end
```

```
                if w>0
                stato=3;
                else
                stato=4;
                end

            elseif t+T(D(i),D(h))<A(D(i))
                %non rientro nella finestra temporale se sono
                %qui: però devo capire se sono fuori o dentro
                disp('I elseif');
                if w>0
                  stato=3;
                else
                   stato=4;
                end
            elseif t+T(D(i),D(h))>B(D(i))
                disp('II elseif');
                stato=4;
                K
                                                    stato
            else
              if w>0
                 stato=3;
              else
                   stato=4;
              end
            end


          case 1
           disp('ci so passat');
              if w>0
                 stato=3;
              else
                 stato=4;
              end


        end

    end

end
K
stato
```

```
    case 2
        K
        v
        w
        t
        h
        disp(t)
        disp('sono nello stato 2');
        for i=v:1:a(1) %vedo se è posibile ricercare una soluzione aspettando
            switch b(i)
                case 0
                    if  w-W(D(i))>=0
                        disp('ci passo nellif?')

                        if t+T(D(i),D(h))<A(D(i)) && t+T(D(i),D(h))+S(D(i))<stop
                            t=t+T(D(i),D(h))+(A(D(i))-t)+S(D(i));
                            k=k+1;
                            K(j,k)=D(i);
                            w=w-W(D(i));
                            b(i)=1;
                            h=i;
                        elseif t+T(D(i),D(h))>=A(D(i)) && t+T(D(i),D(h))<=B(D(i))
&& t+T(D(i),D(h))+S(D(i))<stop
                            t=t+T(D(i),D(h))+S(D(i));
                            k=k+1;
                            K(j,k)=D(i);
                            w=w-W(D(i));
                            b(i)=1;
                            h=i;
                        end


                        disp('ok');

                    %aspetta
                    end
                case 1
                    disp('non faccio niente');
            end
        end
        disp('debug');

        for i=a(1):-1:2
            disp(i);
            t
```

```
        if i>3
            switch b(i)
                case 0
                    disp('ci passo?');
                    disp(w-W(D(i)));
                    disp(t+T(D(i),D(h)));
                    disp(A(D(i)));
                    if  w-W(D(i))>=0
                        disp('ci passo nellif?')

                        if t+T(D(i),D(h))<A(D(i)) && t+T(D(i),D(h))+S(D(i))<stop
                            t=t+T(D(i),D(h))+(A(D(i))-t)+S(D(i));
                            k=k+1;
                            K(j,k)=D(i);
                            w=w-W(D(i));
                            b(i)=1;
                            h=i;
                        elseif t+T(D(i),D(h))>=A(D(i)) && t+T(D(i),D(h))<=B(D(i))
&& t+T(D(i),D(h))+S(D(i))<stop
                            t=t+T(D(i),D(h))+S(D(i));
                            k=k+1;
                            K(j,k)=D(i);
                            w=w-W(D(i));
                            b(i)=1;
                            h=i;
                        end


                        disp('ok');

                    %aspetta
                    end
            end
        else
            switch b(i)
                case 0
                    if  w-W(D(i))>=0
                        disp('ci passo nellif?')

                        if t+T(D(i),D(h))<A(D(i)) && t+T(D(i),D(h))+S(D(i))<stop
                            t=t+T(D(i),D(h))+(A(D(i))-t)+S(D(i));
                            k=k+1;
                            K(j,k)=D(i);
                            w=w-W(D(i));
                            b(i)=1;
```

```
                    h=i;
                    disp('devo aspettare');
                elseif t+T(D(i),D(h))>=A(D(i)) && t+T(D(i),D(h))<=B(D(i))
&& t+T(D(i),D(h))+S(D(i))<stop
                    t=t+T(D(i),D(h))+S(D(i));
                    k=k+1;
                    K(j,k)=D(i);
                    w=w-W(D(i));
                    b(i)=1;
                    h=i;
                else
                    stato=4;
                end

                disp('ok');
                stato=4;
            else
                stato=4;
                %aspetta
            end

                %stato=2;
        case 1
            stato=4;



    end
        %aspetta
    %   else

        %end



  end
%break;
end
K
t
stato
case 3
    %stato di check
    %I check: abbiamo finito di esplorare?
```

```
disp('sono nello stato 3');
n=0;
K
for s=1:1:a(1)
n=n+b(s)
end
if n==a(1)
   c=1;
   disp('soluzione trovata');
else
%II check: c'è ancora spazio per servire un altro cliente?
   o=2;
   z=0;
   while z==0
      if W(o)<=w
         z=1;
         stato=2;
         v=h;
      else
         o=o+1;
      end
      if o==a(1)
      z=1;
      end
   end
%III check: se non devo andare nello stato 2 devo tornare all'uno
perché la possibilità di soluzione trovata l'ho già scartata
   if stato~=2
      stato=1;
      j=j+1;
      k=1;
      o=2;
      p=0;
      while p==0
         if b(o)==0
            h=o;
            K(j,k)=D(o);
            b(o)=1;
            p=1;
            t=A(D(o))+T(D(1),D(o))+S(D(o));
            w=Wmax-W(D(o));
         else
            o=o+1;
         end
         if o==a(1)+1
```

```
                p=1; %superfluo, ma non si sa mai
                c=1;
              end
            end
          end
        end
case 4
        disp('sono nello stato 4');
        % I check: abbiamo finito di esplorare?
        n=0;
        for s=1:1:a(1)
        n=n+b(s)
        end
        if n==a(1)
```

3.3 Implementazione dell'algoritmo in Aimsun

Per implementare un algoritmo in Aimsun, sono necessari questi software:

- AIMSUN 6;
- Visual Studio 2005 + Service Pack 1;
- Qt Library;
- Qt-VS2005 patch;
- 'vsvar32.bat'. Questo file batch apre il Visual Studio e aggiunge i percorsi necessari per compilare la dll;

e questi codes files

- Log*AlgorithmName*.h;
- Log*AlgorithmName*.cpp;
- Log*AlgorithmName*Util.h;
- *AlgorithmName*.pro.

Per scrivere il codice è necessario la "AIMSUN Logistics SDK". Usiamo tre classi di Aimsun:

- Class LogVRPResult;
- Class LogVRPData;
- ClassAlgorithm.

L'algoritmo è stato implementato in **C++**. Il codice contiene un'istruzione che indica Aimsun a caricare il dll e registra l'algoritmo.

Extern "C" LOGSAMPLEROUTE *LogSampleAlgorithm : public LogAlgorithm*

Come si può vedere nella linea di codice, il nome dell'algoritmo è *SampleAlgorithm*. È importante vedere che ci sono due algoritmi: il primo ha l'obiettivo di scrivere un file in cui ci sono tutti i tempi di viaggio, il secondo contiene il vero e proprio algoritmo. Io descriverò il secondo, ma farò qualche riferimento al primo algoritmo nelle opportune parti.

3.3.1 *LogSamleAlgorithm.cpp*

Questa è la classe in cui l'algoritmo viene implementato.

Cominciamo a descrivere i metodi. La dichiarazione del prototipo di questi metodi è stata fatta nell'header file *LogSampleRoute.h*.

3.3.1.1 *Il costruttore*

Il costruttore, in questo caso, ha la funzione di inizializzare alcuni importanti attributi:

- *Name*: logistic plug-in che utilizza questi attributi per registrare internamente l'algoritmo;
- *externalName*: questo è il nome che verrà mostrato agli utenti e deve coincidere con il campo external name nel xml file;
- *Info*: contiene una piccola descrizione dell'algoritmo.

Questi attributi sono, rispettivamente, l'output dei metodi *getName()*, *getExternalName*() e *getDescription*().

3.3.1.2 *setData(LogVRPData *)*

Questo metodo inizializza la struttura dell'algoritmo e legge i dati da una variabile di tipo LogVRPData che viene fornito da Aimsun.

Come mostrato nella prima sezione, sono necessari alcuni input per l'algoritmo; nel senso che c'è un problema legato con la matrice di costi: sono necessarie sia la matrice dei tempi di viaggio sia la matrice dei costi metrici.

Nell'esecuzione dell'algoritmo che sto implementando in Aimsun, vorrei scegliere i due tipi di costo insieme, ma ce n'è solo una possibilità alla volta (figura 3.1).

Per ovviare al problema, uso un algoritmo di inserimento in cui l'unico obiettivo è di scrivere in un file la matrice dei tempi di viaggio. Le altre parti servono soltanto a far andare il codice in Aimsun. Naturalmente, nell'algoritmo principale ho inserito una parte di codice per prendere questi dati da un file scritto dal precedente algoritmo.

Come mostrato subito dopo, l'unica cosa da notare è che c'è un vettore e devi settare la dimensione per prendere questi costi del tempo di viaggio.

```
...
float sky[number_of_clients];
```

fscanf(fp,"%f%f%f...",&sky[0],&sky[1],...);

…

Con questo metodo si legge linea per linea.

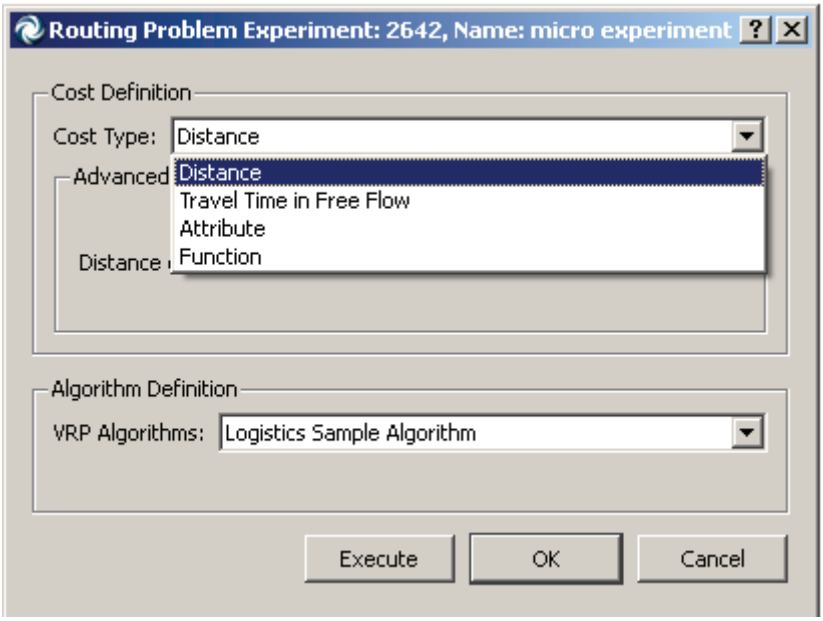


Figura 3.1 *La finestra per scegliere il costo*

Nell'esecuzione del secondo algoritmo ho scelto 'distance' come 'cost' per avere la matrice C. Ho salvato le due matrici in due *Qmap<key,T>*. È una template class della libreria Qt che memorizza (key,value) e fornisce facili accessi a value .

Dopo ho settato pure

- nodesDemand
- initialTime
- endTimes
- serviceTime
- vehiclesCapacity

Per salvare questi dati ho utilizzato un'altra template class di Qt, the *Qhash<key,T>* , che è lo stesso di Qmap ma fornisce accessi più veloci.

È importante notare che i dati del tempo sono presi dall'editor grafico in cui c'è il formato dell'orologio (figura 3.2). Così, per eseguire l'algoritmo dobbiamo tradurre le ore e i minuti in secondi. Fortunatamente, la libreria Qt per la logistica fornisce un metodo di nome toSeconds per dati di tipo LogVRPData.

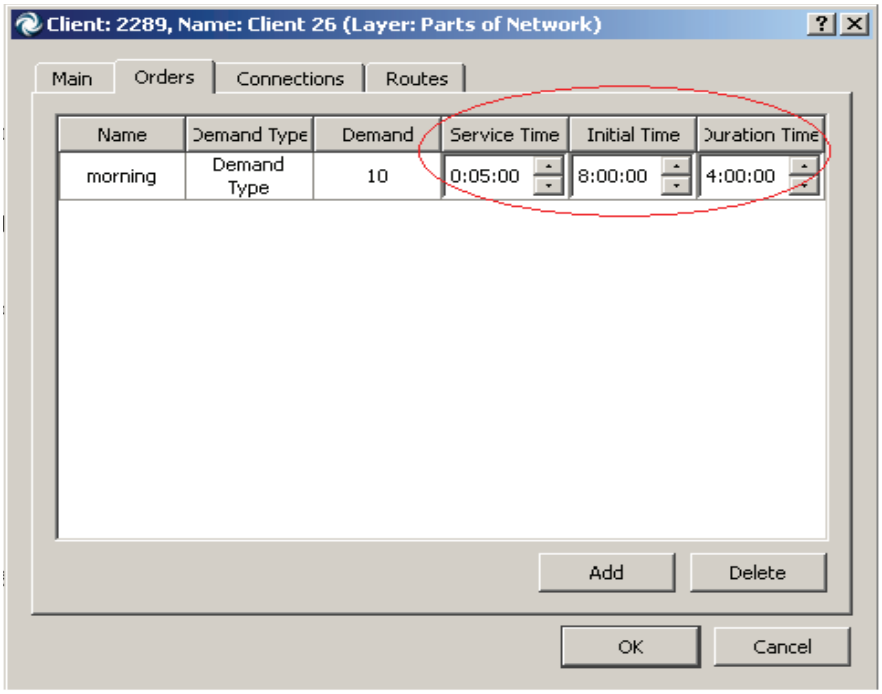


Figura 3.2 *La finestra di un cliente*

Dopo, come mostrato in figura 3.2, il tempo di end time viene calcolato come

initial time + duration time

Subito dopo c'è lo pseudocodice con il metodo annesso.

---

for *(iterator from the first vertex to the last)*

*{**if** vertex is the depot*
take from Aimsun and save the start time of the depot
take from Aimsun and save the end time of the depot
take from Aimsun and save the duration time of the depot
else
take from Aimsun and save the start time of the current client
take from Aimsun and save the end time of the current client
take from Aimsun and save the duration time of the current client
take from Aimsun and save the service time of the current client
take from Aimsun and save the demand of the current client
}

take from Aimsun and save the capacity of the fleet
take from Aimsun and save the metric costs
take from the file written previously the travel time costs

3.3.1.3 checkData(LogVRPData *,QString &)

Questo metodo controlla se la definizione del problema soddisfa i principali vincoli dell'algoritmo.
In questo caso, faccio due tipi di controllo:

- calcola il numero di depositi nella definizione;
- controlla se tutti i veicoli della flotta hanno la stessa capacità.

Il risultato di questo controllo è dato da un tipo booleano, che è falso se ci sono alcuni problemi.

***for** (iterator from the first vertex to the last){*
***if** vertex is a depot*

sum 1 to the number of depot
}
***if** number of depot is grater than 1*
output_1 : false
output_2 : "Too much depot"
***for**(iterator from the first vehicle of the fleet to the last)*
*{ **if** capacity of next vehicle is different from the capacity of the current vehicle*
output_1 false:
output_2 : "The capacity of the vehicles must be the same"
}

3.3.1.4 *execute(bool &)*

Questo metodo contiene l'implementazione dell'algoritmo. Usiamo diversi metodi di un'altra classe implementata nel progetto, chiamata *LogSampleRoute*.cpp, che è usata per verificare se un cliente può essere inserito o no nel percorso attuale. Come mostrato nelle prima parte, ci sono tipi di condizione, una

$time + t_{ij} > a_j$
$time + t_{ij} \leq b_j$

e l'altra

$time + t_{ij} < a_j$

Allo scopo di rispettare i vincoli, ci sono due differenti metodi.

Tornando a LogSampleAlgorithm, la prima parte di questo metodo è la parte in cui creo una nuova matrice, chiamato *Ordinato*, che corrisponde alla matrice E della sezione *"La progettazione dell'algoritmo"*, in cui ogni indice di riga corrisponde

al nodo e l'elemento di colonna sono gli altri nodi dal più vicino al più lontano.

Il corpo dell'algoritmo è scritto come una "macchina a stato" in cui cui abbiamo diversi stati.

3.3.1.4.1 Stato '0'

Lo stato '0' corrisponde alla prima inserzione del cliente. Lavora solo all'inizio. Prima c'è l'inserzione del deposito, perché per l'algoritmo statico Aimsun offre uno standard per il percorso: depot->client i->client j->.... Prendo il secondo elemento delle prima riga della matrice Ordinato e chiedo al metodo canServeClient(...) se questo nodo può essere inserito nella soluzione (canServeClient ha come uscita un valore booleano). Nel caso in cui la risposta è negativa, il prossimo nodo da considerare è il terzo della prima riga e così via, nell'altro caso (la risposta è vera) inseriamo il nodo in soluzione e il controllo salta allo stato '1'.

Se, invece, non ci sono nodi da inserire con questa condizione, il secondo elemento della prima riga della matrice diventa automaticamente il primo cliente nella soluzione e il controllo salta allo stato '1'.

Naturalmente, per ogni cliente inserito ci sono anche gli aggiornamenti delle variabili *Ordinato_bool[i]* che ha la stessa funzione del vettore *bool* descritto nella prima sezione, e la variabile *inserted_Nodes* che contiene il numero di nodi che sono nella soluzione. Per ogni inserzione usiamo anche un metodo chiamato *addClient(..)* che deve inserire l'arco nel percorso attuale.

3.3.1.4.2 Stato '1'

In questo stato c'è la ricerca di un altro nodo da inserire nella soluzione. Utilizza canServeClient e la riga di Ordinato che ricerca un nuovo cliente corrispondente all'ultimo nodo inserito.

Naturalmente, quando un cliente è inserito, aggiorna inserted_Nodes e Ordinato_bool[i].

Finché ci sono inserzioni di nodi il controllo del programma è nello stato '1': quando la riga di Ordinato arriva alla fine senza nessuna inserzione abbiamo due possibilità legate alla capacità. Se *capacity_fleet* meno capacity è più grande di zero vado allo stato '3', altrimenti vado allo stato '2'.

3.3.1.4.3 Stato '2'

Vado in questo stato se la capacity_fleet meno capacity non è maggiore di zero. A questo punto devo costruire un nuovo percorso, un nuovo veicolo della flotta e devo scegliere quale nodo inserire nella soluzione. All'inizio comincio ad inserire il deposito, naturalmente, e poi il processo è lo stesso dello stato '0' per la prima parte. Per la seconda parte, si inserisce il primo cliente non inserito dalla prima riga di ordinato, se non abbiamo trovato nessun cliente da inserire nella prima parte. Dopo saltiamo allo stato '1'.

3.3.1.4.4 Stato '3'

Sappiamo che il mio veicolo è in grado di fornire ancora dei beni, e so che non ci sono nodi che rispettano i vincoli di CanServeClient. Come mostrato nella prima sezione, rilassiamo il vincolo allo scopo di inserire un cliente nella soluzione anche se il veicolo arriva prima dell'apertura della finestra. Se inserisco un cliente passo allo stato '1', se non lo inserisco vado allo stato '2'.

3.3.1.4.5 L'end

Quando l'algoritmo è finito dobbiamo uscire dalla “macchina a stato”. Questa cosa è realizzata, implicitamente, quando la variabile insert_Nodes diventa uguale al numero di clienti.

---

while *number of inserted clients is different from the number of clients {*
switch *the value of the variable "stato"{*
case '0'*: //initialization*
while *I don't find a client to insert{*
if *I can serve the current client*
{insert the depot in the solution
insert the client in the solution
variable stato becomes 1
increment the number of inserted clients
sum to the variable capacity the demand of the current client
signal that this client has been served
else
explore the successive client
if *I can't insert any client*
variable stato becomes 1
insert the depot in the solution
insert the first client on the row 0 of the matrix Ordinato in the solution
increment the number of inserted clients
signal that this client has been served
sum to the variable capacity the demand of the current client
}
break*;*
case '1*':* ***while*** *I don't find a client to insert{*
if *I can serve the current client*
insert the client in the solution
variable stato remains 1
increment the number of inserted clients
signal that this client has been served
sum to the variable capacity the demand of the current client
else

explore the successive client
***if** I can't insert any client*
***if** current vehicle can deliver other goods*
variable stato becomes 3
exit from while
else
variable stato becomes 2
exit from while
}
***break**;*
***case '2':** // it creates a new route*
***while** I don't find a client to insert{*
***if** I can serve the current client*
{insert the client in the solution
variable stato becomes 1
increment the number of inserted clients
sum to the variable capacity the demand of the current client
signal that this client has been served
else
explore the successive client
***if** I can't insert any client*
exit from while
}
***while** I don't find a client to insert {*
***if** the current client hasn't be already served*
variable stato becomes 1
insert the depot in the solution
insert the current client in the solution
increment the number of inserted clients
signal that this client has been served
sum to the variable capacity the demand of the current client
else
explore the successive client
}

break;

case '3 ': *//there's* ***the relaxation of the constraint of the time window***
while *I don't find a client to insert{*
if *I can serve the current client*
insert the client in the solution
variable stato remains 1
increment the number of inserted clients
signal that this client has been served
sum to the variable capacity the demand of the current client
else
explore the successive client
if *I can't insert any client*
variable stato becomes 2
exit from while
}
break*;*
}
}

3.3.1.5 generateResult()

Questo metodo genera un'istanza LogVRPResult contenente tutti i clienti che verranno visitati dai veicoli della flotta. Itera in tutti i percorsi risultanti, setta il veicolo id e il percorso del percorso e, poi, aggiunge tutti i clienti nell'ordine di visita.

---

for *(iterator from the first route to the last one)*
{sets in Aimsun the vehicle id
sets in Aimsun the route cost

adds in Aimsun all client in visiting order}

Capitolo 4

Simulazione e validazione

In questo capitolo farò, su di un contesto reale, una simulazione vera e propria in Aimsun: ci saranno macchine, bus, traffico, semafori e così via. Il principio cardine sarà il *tempo*. La simulazione è basata sull'algoritmo descritto nel capitolo precedente. Assegnerò i clienti, creerò una flotta di veicoli e il deposito.
Nella prima simulazione ci sarà una violazione a un cliente, ma nella seconda simulazione, grazie a modifiche pratiche e, soprattutto, sostanziali, validerò la soluzione in maniera definitiva.

4.1 "City Logistic"

La logistica, come definita dal Council of Logistics Management, nel 2001, è quel processo che pianifica, implementa, controlla l'efficienza, stocca le merci, effettua i servizi e tutte le informazioni dal punto d'origine al punto di arrivo in modo da soddisfare i clienti. Tuttavia, quando si svolgono delle attività logistiche nelle aree urbane, queste mostrano delle caratteristiche tipiche che le rendono diverse dalle attività di logistica in generale, ed è la ragione per cui qualora il trasporto merci avvenga nelle aree urbane, in particolare il flusso dei trasporti associati alla fornitura delle merci nei centri urbani, lo stesso è denominato "logistica urbana" o, in inglese "city logistic".

4.2 L'implementazione della "Aimsun Logistic"

L'implementazione della logistica Aimsun è un Plug-in di Aimsun come sistema di supporto alle decisioni per la city logistic. Aimsun è un ambiente software integrato appositamente progettato e realizzato per supportare il traffico e strumenti di analisi del trasporto.

Il grafico $G = (N, A)$ per le applicazioni di Vehicle Routing, come illustrato in figura 4.1, deriva dal grafico Link-Node della rete stradale, costituito da un insieme di nodi N = {0,1,2…n} dove il nodo 0 è il deposito e i nodi i, i=1…n sono i clienti, e l'insieme dei link A=$\{(i,j)|i,j \in N\}$ corrispondono a percorsi praticabili sul nodo di rete stradale che connettono il nodo *i* con il nodo *j* a formare il c_{ij} associato al link $(i, j) \in A$ calcolato o dal tempo di viaggio dal nodo *i* al nodo *j* o al trasporto metrico.

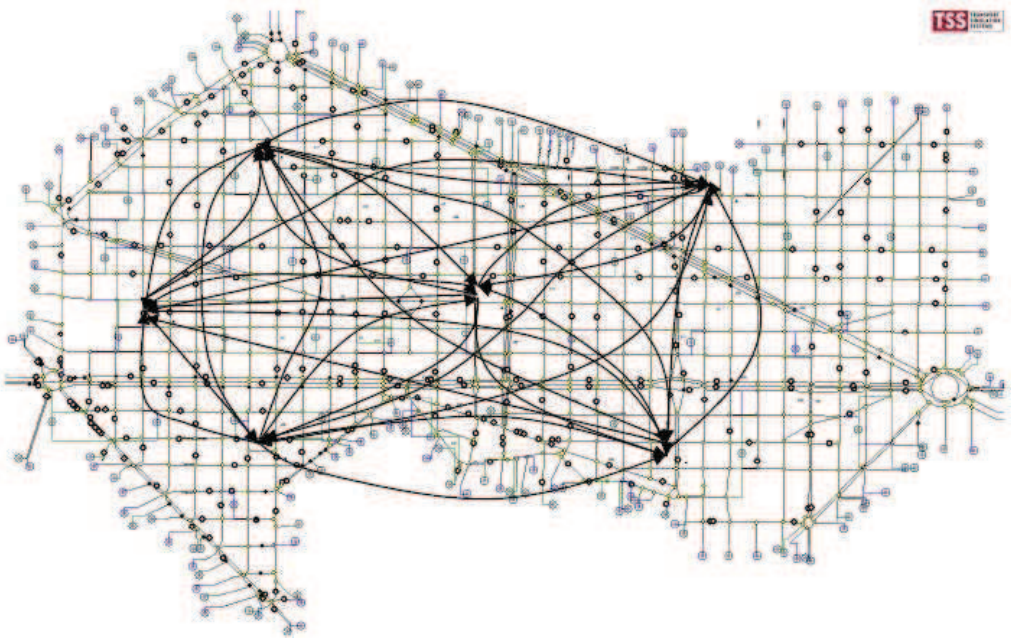


Figura 4.1 *Esempio di grafico* $G = (N.A)$ *per le applicazione VR*

Ciò significa che il processo di traduzione, come implementato nel sistema di supporto decisionale basato su Aimsun, consiste in due parti:

- traduzione della mappa digitale (modello di rete Aimsun) della rete urbana in termini di modello di rete link-Node come un grafo diretto includendo la rappresentazione dei nodi "speciali" dei depositi e dei clienti;
- traduzione del modello di rete in termini di grafo di Vehicle Routing i cui nodi sono depositi e clienti e le cui maglie sono percorsi praticabili nel grafo link-node orientato tra il deposito e i clienti e tra clienti e clienti.

Il generico diagramma concettuale di un "routing-simulation" integrato quando è implementato con Aimsun può essere definito in figura 4.2, che mostra come il grafo $G = (N, A)$ venga messo a disposizione dell'analista, che calcola la migliore soluzione attuale in base alle informazioni disponibili sui tempi di percorrenza del percorso, determina le nuove rotte e gli orari dei veicoli e invia queste informazioni al modello di simulazione che muoverà la flotta dei veicoli di conseguenza.

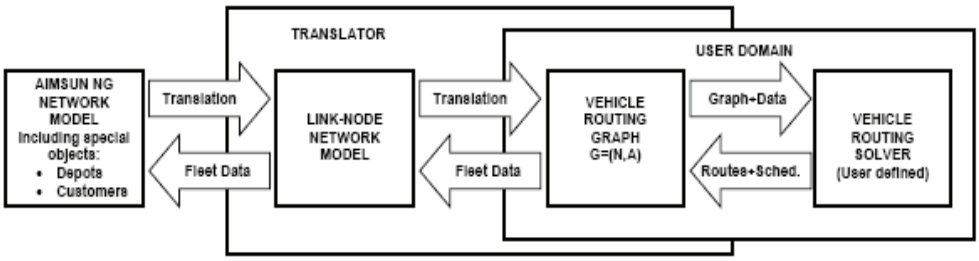


Figura 4.2 *Diagramma concettuale del"Routing-Simulation" basato su Aimsun*

4.3 "Aimsun Logistics"

La figura 4.3 descrive il diagramma "logico" in cui Aimsun Logistics lavora.

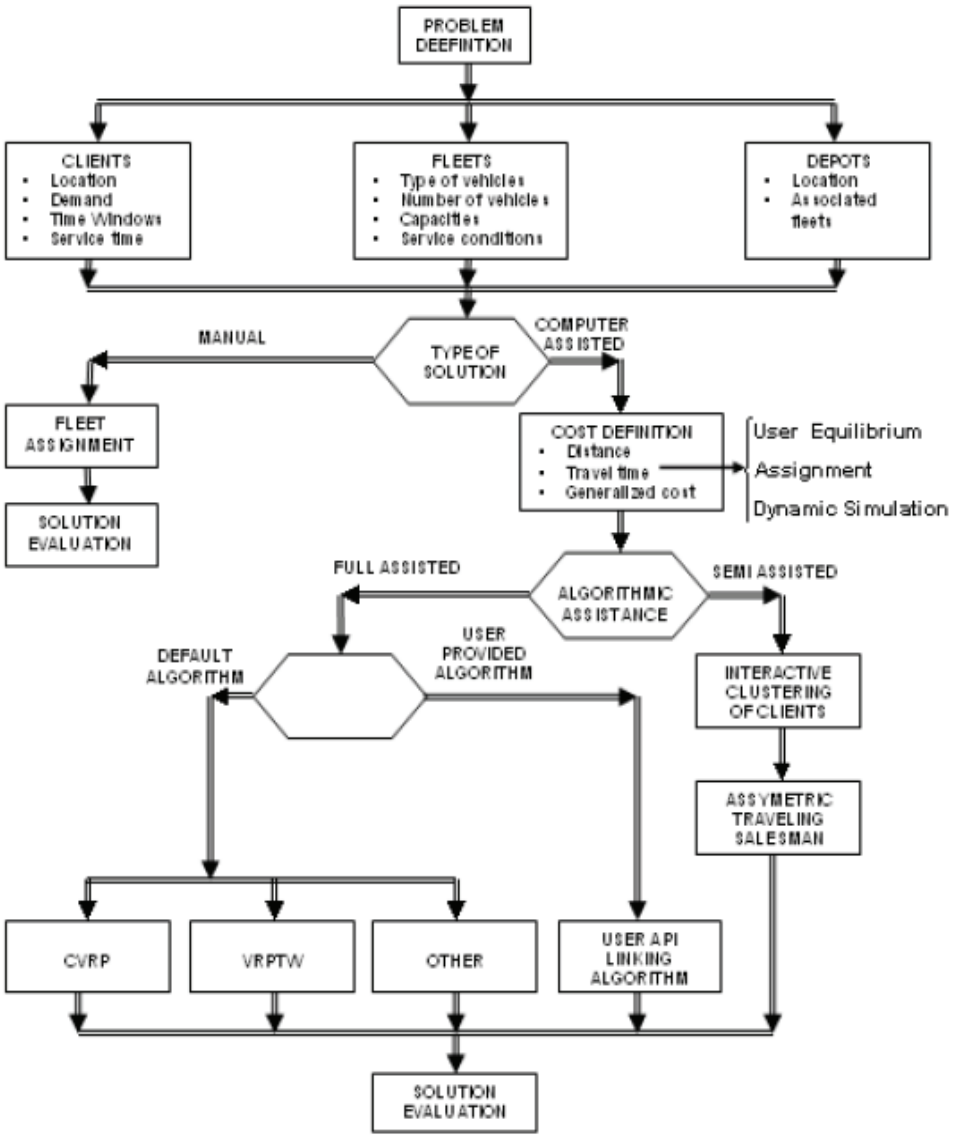


Figura 4.3 *Il diagramma logico di "Aimsun Logistic"*

La definizione del problema ha tre componenti:

- clienti;
- flotta;
- deposito.

Definiamo i clienti e i suoi attributi:

- locazione;
- domanda;
- la time window;
- tempo di servizio (service time).

Per continuare devo definire la flotta e le sue caratteristiche,

- tipo di veicolo che compone una flotta;
- il numero di veicoli per ogni flotta;
- capacità del veicolo;
- condizioni di servizio (ad esempio l'inizio e la fine del tempo di servizio);

e, infine, devo definire i depositi

- posizione;
- flotte associate, cioè che le flotte operano da ciascun deposito.

Una volta che il problema è definito, Aimsun Logistica permette di selezionare il tipo di processo di soluzione che intende utilizzare. Il sistema, attualmente, supporta due possiblità:

- soluzione manuale, l'utente assegna i clienti ai veicoli manualmente e definisce i percorsi;
- assistita da computer, l'utente seleziona il tipo di supporto algoritmico che vuole risolvere il problema.

In caso di selezione di opzione Aimsun assistita da computer gli utenti devono prima definire i costi su cui la soluzione verrà basata, ci sono tre opzioni:

- costi basati sulla distanza forniti direttamente da Aimsun basata, a sua volta, sulla lunghezza tra deposito e cliente e tra cliente e cliente;
- costi basati sui tempi di viaggio;
- costi generalizzati, utilizzando la capacità dall'Aimsun editing per definire funzioni di costo.

4.4 Interfaccia grafica utente di Aimsun Logistics

La logistica Aimsun offre molte caratteristiche utili da utilizzare nell'interfaccia grafica utente. Vediamo nei dettagli la creazione del modello e la simulazione.

4.4.1 Definizione di un cliente

Cliccando sul pannello "new client" (nuovo cliente) mostrato in figura 4.4, e, dopo, cliccando su 2D (2 dimensioni), creerò un nuovo cliente del modello.

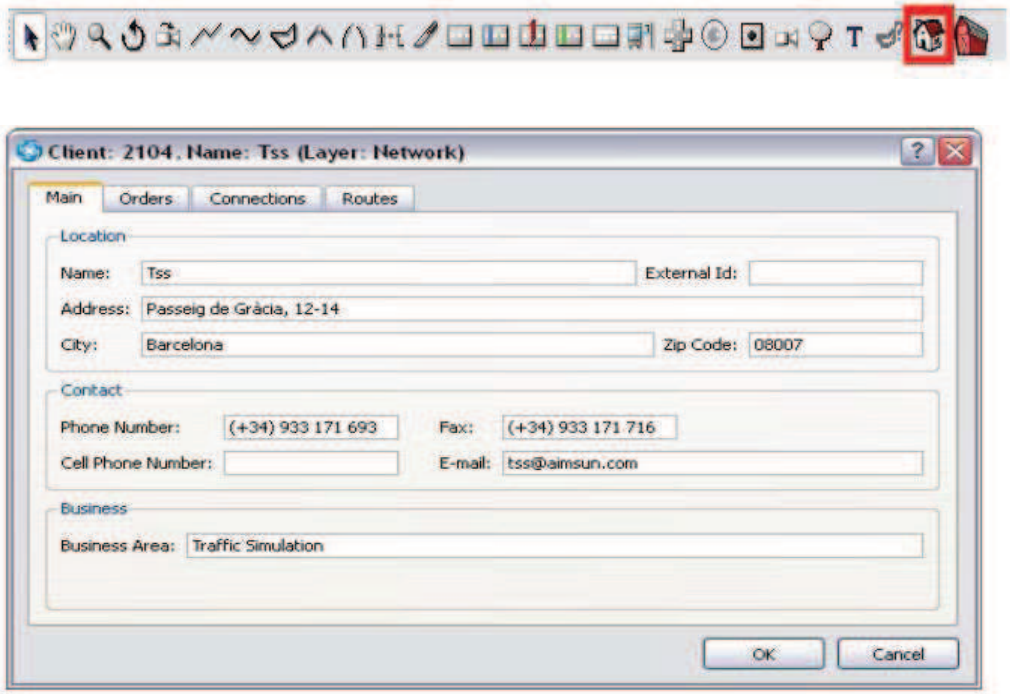


Figura 4.4 *Attributi del cliente*

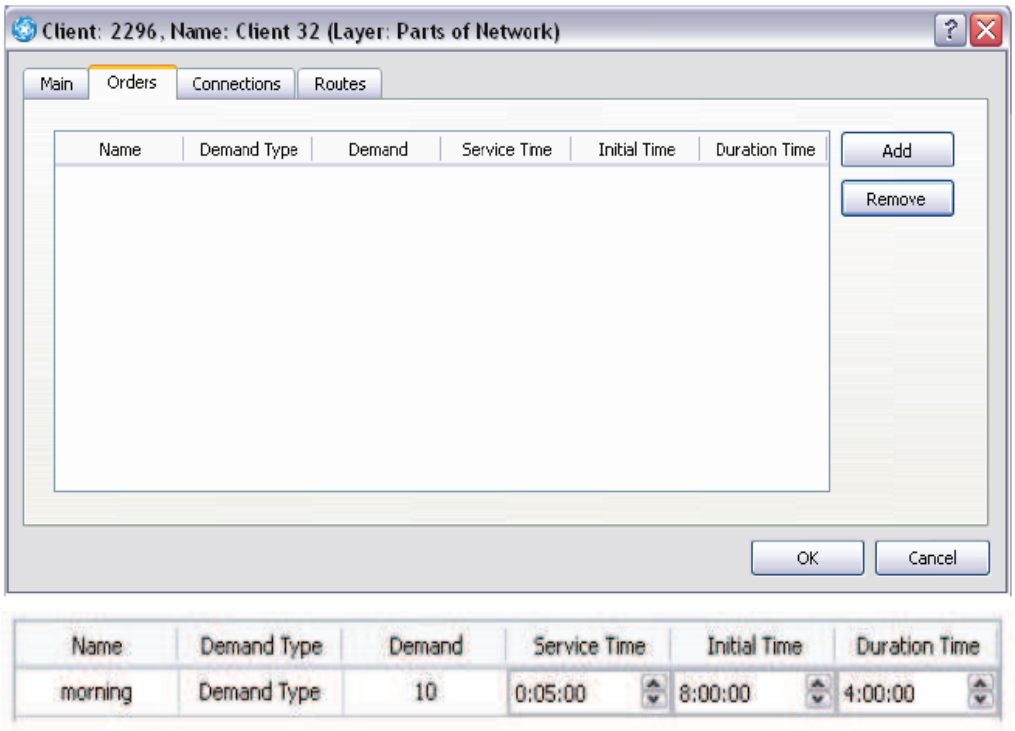


Figura 4.5 *Ordine del cliente*

Questo editor (Fig.4.5) permette all'utente di aggiungere o rimuovere l'ordine da un qualsiasi cliente. Assegno il nome "*morning*", il tipo di domanda "*demand type*", la domanda "*10*", la

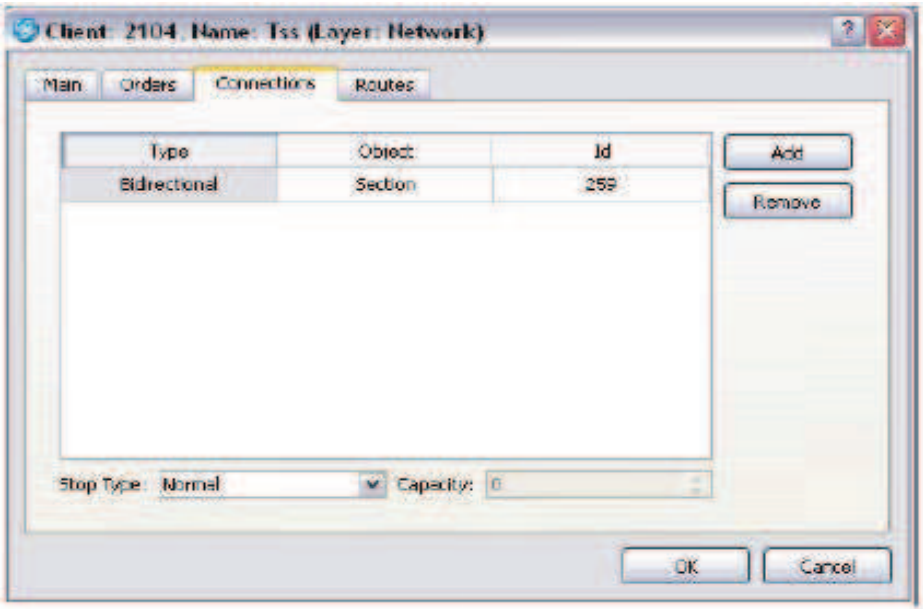


Figura 4.6 *Connessione del cliente*

service time di "*5*" minuti, il tempo iniziale "*8*" e la durata "4" ore.

Nella Figura 4.6 ogni cliente può avere una sola connessione ad una sessione e il suo tipo è bidirezionale: lo setto con il tasto "Add".

4.4.2 Il deposito

Per creare un deposito, clicco sopra "new depot" e seleziono la vista 2D.

Scrivo su "Name" *South Depot*, su "fleets" *Inter Fleet*, e su "connections" al nuovo deposito, come mostrato in figura 4.7.

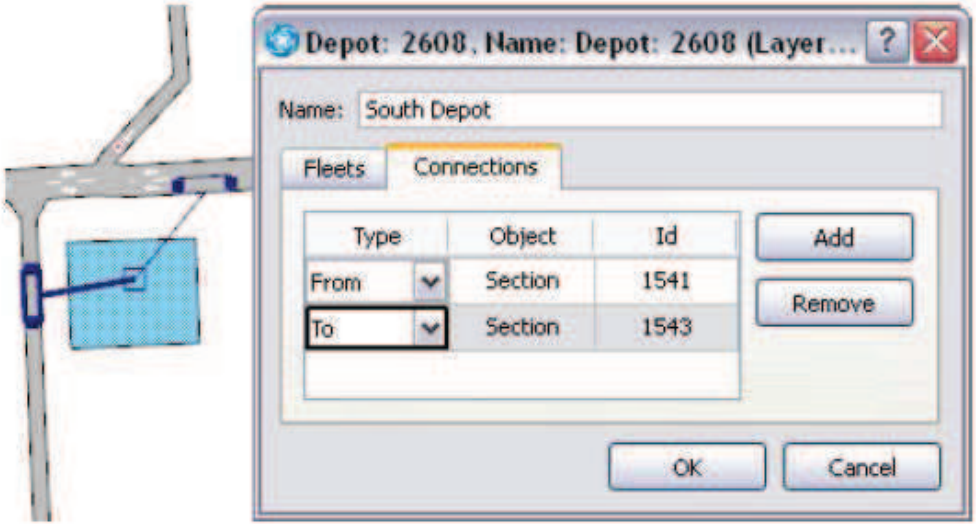


Figura 4.7 *Collegamento del deposito*

4.4.3 La flotta

Per creare nuove flotte selezioniamo dal menu *Site/New…/Fleet*. Selezioniamo, ancora, il tipo di veicolo, ed è un furgone (*van*), e la capacità è di *100*.

Definiamo la "*time properties*", con il Name *Morning Delivery*,

con l'initial time "*8*" e con la duration ancora "*8*". Possiamo vedere, in figura 4.8, tutte le selezioni fatte.

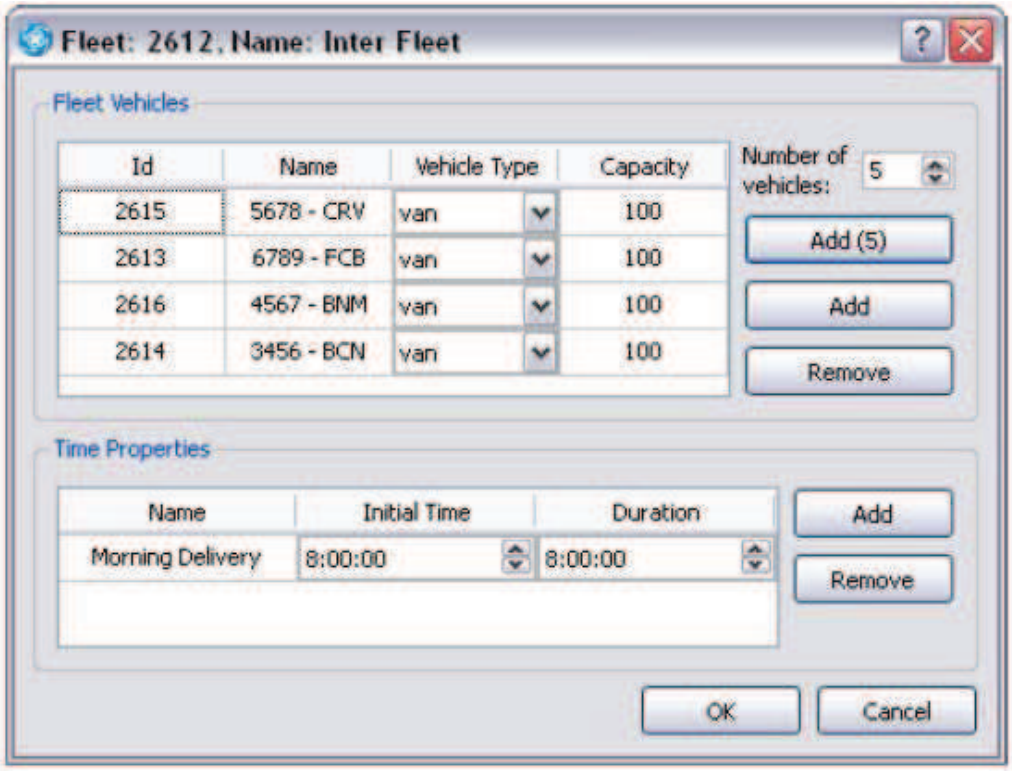


Figura 4.8 *La finestra per dichiarare una flotta*

Possiamo vedere, sotto forma di tabella, il riassunto: deposito (South Depot) e clienti (10,15,16,18,19,30).

Node	initial time	end time	service time	demand
South Depot	8.00	16.00	-	-
10	8.00	12.00	5	10
15	8.00	8.20	5	10
16	8.00	12.00	5	10
18	8.00	12.00	5	10
19	8.00	12.00	5	10
30	8.00	12.00	5	10

A questo punto abbiamo tutti gli elementi necessari, e presentiamo la mappa, mostrata dalla figura 4.9.

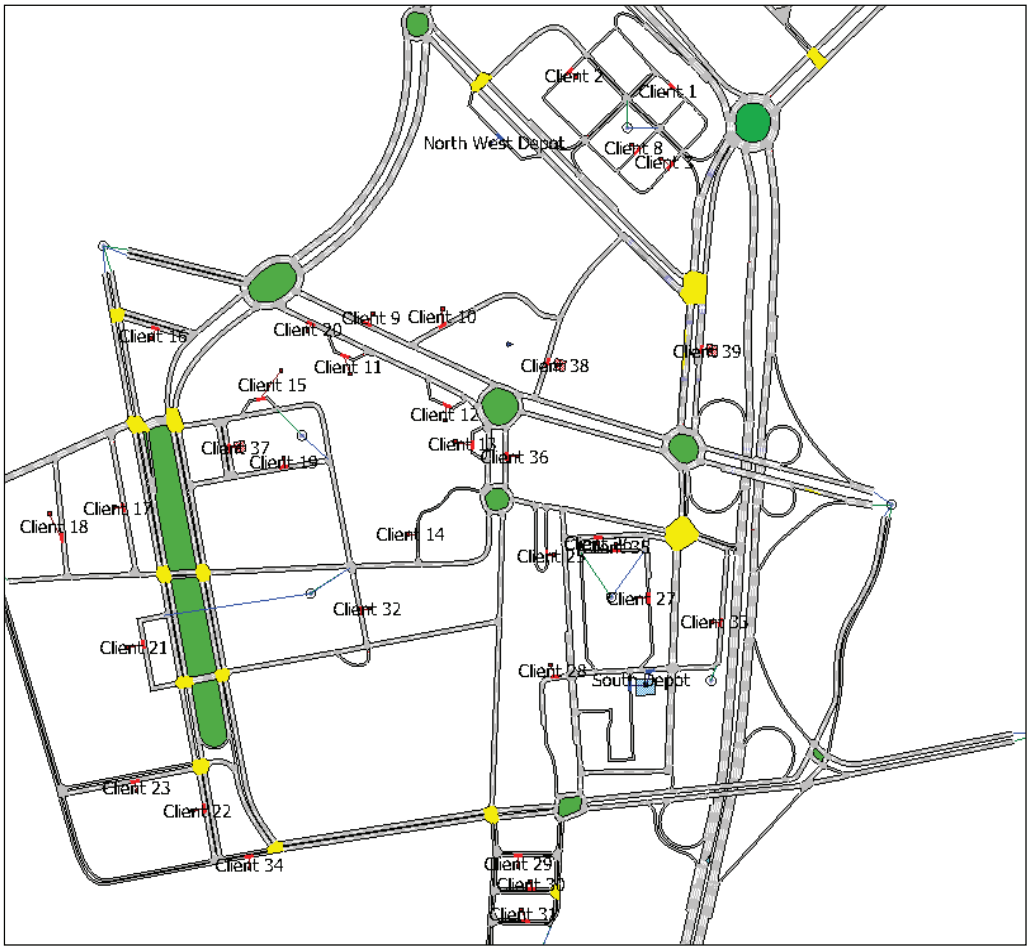


Figura 4.9 *La mappa per la simulazione*

4.5 L'esecuzione dell'algoritmo

Per eseguire un algoritmo, e, in particolare, l'algoritmo basato sul capitolo 3, dobbiamo definire *Routing Problem Scenario* in cui ci sono deposito, flotta e clienti. Creo un nuovo scenario con la figura 4.10, e assegno i clienti 10,15,16,18,19,30, com'era già stato detto.

Il secondo step consiste nel creare un nuovo esperimento di Routing Problem. Nella figura 4.11 c'è l'immagine associata al routing problem.

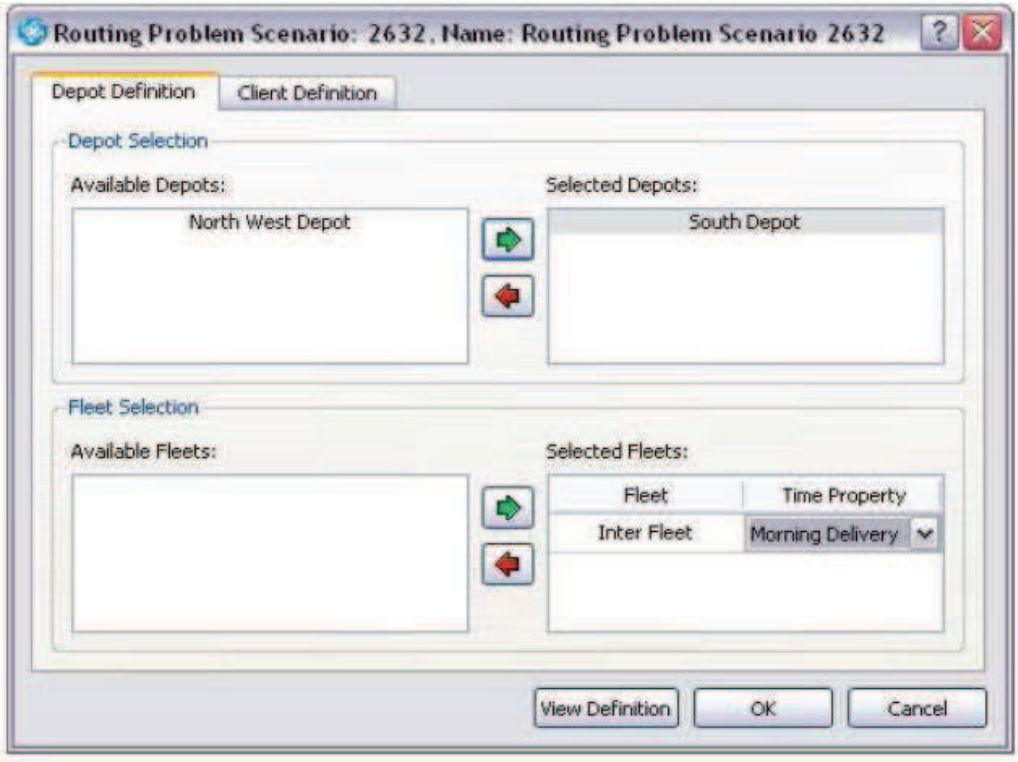


Figura 4.10 *Il Routing Problem Scenario*

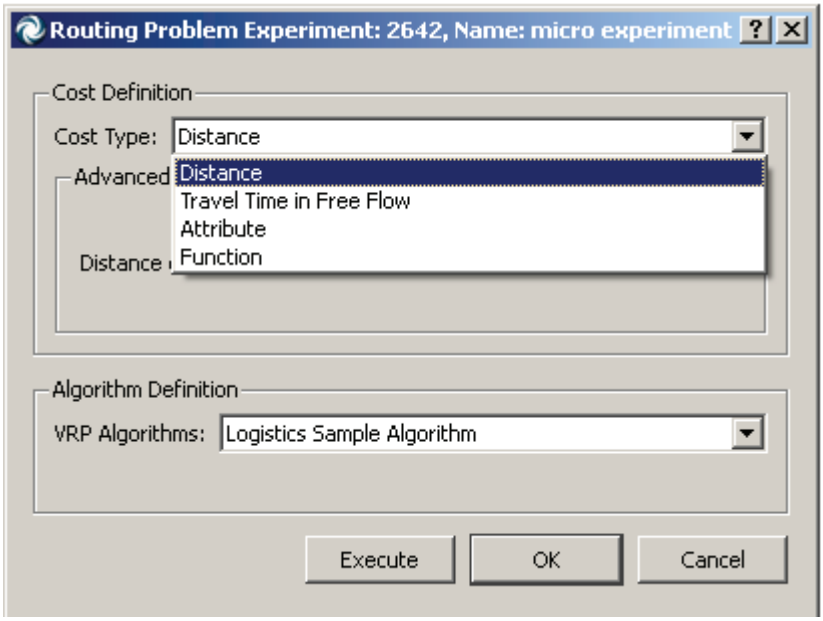


Figura 4.11 *Routing Problem Scenario*

È necessario selezionare *Logistic Sample Algorithm* come algoritmo e *Travel Times in free flow* come costo nella finestra di Aimsun, e che corrisponde a un esperimento della dichiarazione Routing Problem Scenario.

Fatto l'algoritmo I, si mette in Aimsun l'algoritmo II e si compiono i medesimi passi. Come risultato abbiamo una *Fleet Assignment* come mostrato nella figura 4.12.

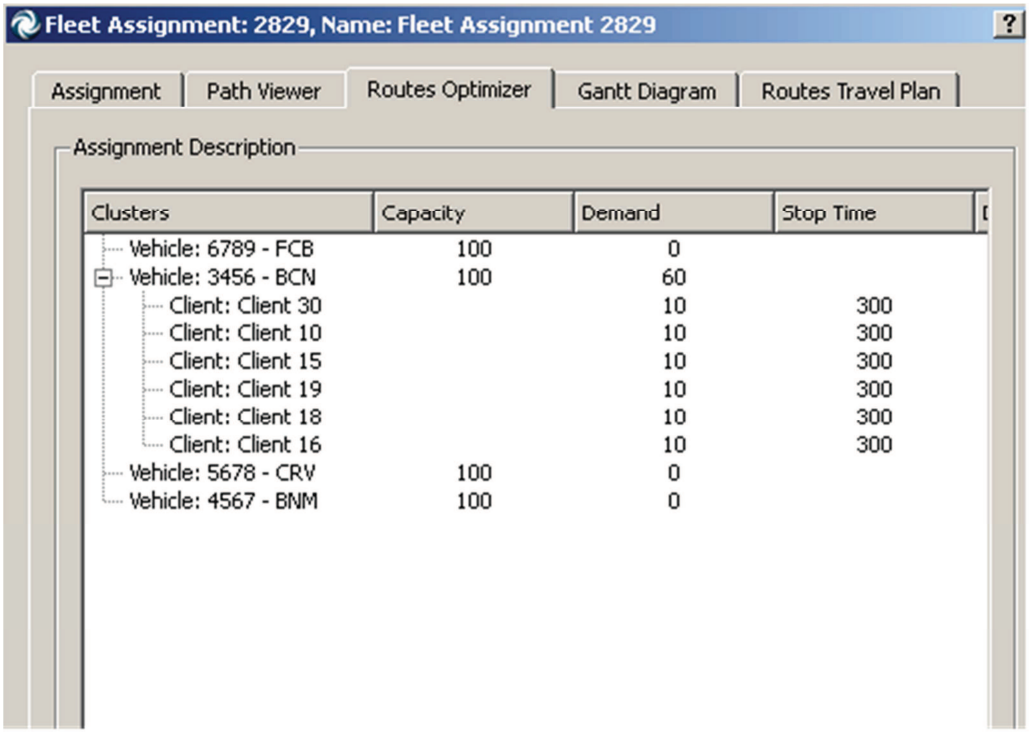


Figura 4.12 *Il fleet Assignement Plan*

4.6 La simulazione

Per fare la simulazione, creiamo un *Fleet Assignment Plan* con il Fleet Assignment fatto dall'esecuzione dell'algoritmo. Dopo, facciamo una replicazione di un esperimento di uno Scenario che ha il Fleet Assignment Plan fatto come input.

Figura 4.13 *Simulazione della rete iniziale*

Durante la simulazione noto che la finestra del cliente 15, causa traffico intenso, non è rispettata (in questo problema la simulazione fallisce).

Come conseguenza voglio servire prima il cliente 15. Per fare questa cosa vado a modificare la matrice dei costi metrici. Considero la riga del cliente 30 che è il nodo corrente servito: "forzo" ad andare direttamente dal deposito al cliente 15, risistemando la matrice di costi metrici. Il $c_{0,30}$ viene scambiato con il $c_{0,15}$ e, così, abbiamo la prima soluzione.

Ora, però, cambieranno i clienti serviti in soluzione: prima ci sarà, dopo il 15, il cliente 19, poi a seguire il 18, 16, 10 e, per finire, il cliente 30.

Figura 4.14 *Simulazione della rete finale e validazione*

Nel vedere la simulazione non ci sono più violazioni. Con questo processo abbiamo validato la soluzione.

Conclusioni

Gli obiettivi di questa tesi sono stati rispettati.

Preventivamente ho sistemato la matrice dei costi metrici dal più breve tratto di strada fino al più lungo, a partire dal deposito.

Ho sviluppato un algoritmo euristico di tipo Vehicle Routing Problem con Time Window, andando dapprima ad analizzare le finestre temporali in maniera "hard": il cliente ha un range di tempo stabilito a priori.

Trovata una soluzione si inserisce in soluzione e si passa al cliente successivo e così via. Se, invece, non si inserisce nessun cliente in soluzione, si va ad inserire il primo cliente, cioè il cliente che nel tratto di strada è il più breve a partire dal deposito. Si passa, poi, nella seconda fase: abbiamo due alternative: scelgo la prima se la capacità del veicolo della flotta attuale è più grande di zero, la seconda nell'altro caso.

Nel primo caso utilizzo una nuova condizione per inserire un cliente: se la opening window del cliente è dopo il tempo di arrivo del mio veicolo, inserisco questo cliente nella soluzione ma so che il veicolo dovrà aspettare prima che incominci il servizio. Nel secondo caso creo un nuovo percorso e inserisco il cliente.

Quando ho inserito un nuovo cliente devo verificare se ho inserito tutti i clienti allo scopo di andare al blocco fine in cui ho la soluzione.

Dopo, ho implementato in Aimsun l'algoritmo VRPTW. Sono stati effettuati due tipi di algoritmi diversi: come risultato, nel primo è stato registrato, su di un file, i tempi di viaggio; nel secondo è stato effettuato il vero e proprio algoritmo.

Infine, sempre mediante l'Aimsun, ho "settato" i depositi, la flotta e i clienti e ho simulato in un ambiente reale: la simulazione però, inizialmente fallisce. Ho dovuto modificare il tragitto dei clienti e, finalmente, validare la soluzione.

Questa simulazione è stata, soprattutto, robusta. Cambiando le finestre temporali, la soluzione è stata sempre validata.

Il passo successivo sarà quello di creare una sorta di processo automatico in grado di fornire una soluzione quanto più accurata e aderente alla realtà.

Bibliografia

AA.VV., *Aimsun Logistics Manual*, version 6.0, October 2007, 2006-2007 TSS (Transport Simulation Systems).

AA.VV., *Metodi di micro simulazione del traffico AIMSUN*, consultato dal sito *www.polinomia.it/strumenti/**aimsun**.html*.

Aldà M., *Algoritmi euristici per il Vehicle Routing Problem*, Università di Padova, A.A. 2003/2004.

Barceló J., *An Overview Of Models To Assist In The Design And Evaluation Of City Logistics Projects*, Technical Report, Department of Statistics and Operations Research, Universitat Politècnica de Catalunya, 2003.

Barcelò J., *Working notes on Vehicle Routing and Scheduling Models and City Logistics*, Department of Statistics and Operations Research, Universitat Politècnica de Catalunya.

Barcelò J., Grzybowska H., Pardo S., *Vehicle Routing and Scheduling Models, Simulation and City Logistics*, Department of Statistics and Operations Research, Universitat Politècnica de Catalunya, 2007.

Bigi G., Frangioni G., Gallo G., Pallottino S., Scutellà M.G.. *Appunti di Ricerca Operativa*. SEU – Servizio Editoriale Universitario Pisano, 2003.

Christofides N., Mingozzi A. e Toth P., *The vehicle routing problem*, in Christofides N., Mingozzi A., Toth P. e Sandi C. (a cura di), Wiley Combinatorial Optimization, Chichester, 1979.

Dantzig G.B., Ramser R. II, *The truck dispatching problem*, Management Science 6 (1959).

Gallo G., *Note di simulazione*, Dispense Corso di Simulazione, Università di Pisa, A.A. 2005/2006.

Peditto R., *Modelli ed Algoritmi per problemi di Vehicle Routing*, Università di Genova, A.A. 2007/2008.

Toth P., Vigo D., *The Vehicle Routing Problem*, in P.Toth , D.Vigo (a cura di), An Overview of Vehicle Routing Problems, Siam, 2002.