POLITECNICO DI MILANO

Facoltà di Ingegneria dei Sistemi

Corso di Laurea in Ingegneria Matematica

Ottimizzazione dei percorsi dei controllori della sosta irregolare: metodi Monte Carlo e algoritmo Genetico

Anno Accademico 2010-2011

Enrica ROMANO, matr. 739307

Silvia ROTA, matr. 734357

Relatori: Prof. Alberto COLORNI

Prof. Maurizio BRUGLIERI

Indice

Elenco delle figure

Abstract

Lo scopo del seguente lavoro è stato quello di risolvere in maniera ottimale ed efficiente un problema di ottimizzazione dei percorsi dei controllori della sosta irregolare.
L'argomento in questione è stato trattato precedentemente nell'articolo [1], riportato in bibliografia, punto di partenza per tale progetto.
Il problema è stato affrontato mediante l'utilizzo di metodi euristici e metaeuristici, che a differenza dei metodi esatti illustrati in [1], hanno assicurato il raggiungimento di buone soluzioni in tempi ridotti.
I modelli presentati sono stati testati e confrontati su un'area della città di Milano, e su un certo numero di istanze casuali.

Introduzione

In questa tesi viene affrontato il **problema della sosta irregolare** e dell'organizzazione efficace di un sistema di controllo basato sul personale addetto alle contravvenzioni: in particolare si affronta il problema di ottimizzare i percorsi dei cosiddetti ausiliari della sosta, tenendo conto dei vincoli di tempo e della configurazione territoriale.

Il parcheggio irregolare è un serio problema nella maggioranza delle città italiane, anche perché nella maggior parte dei casi l'applicazione delle sanzioni non è efficace. Il controllo avviene attraverso gli ausiliari della sosta, che - partendo da una posizione iniziale (deposito) - percorrono un certo numero di strade, controllano le macchine e sanzionano quelle parcheggiate irregolarmente. Gli ausiliari hanno a disposizione un tempo di servizio limitato, al termine del quale tornano al deposito. I comuni non hanno risorse sufficienti per assumere un numero adeguato di ausiliari; inoltre quasi sempre il programma di lavoro degli ausiliari (percorsi e orari) non è stabilito con metodi quantitativi.

La nostra tesi si colloca in questo contesto ed è la prosecuzione di altri lavori sul tema. Siamo partite da una precedente tesi [3] e da un articolo [1] sull'argomento, avendo in comune con questi due riferimenti l'obiettivo di **progettare i percorsi degli ausiliari** al fine di massimizzare il numero di auto sanzionate. Nei lavori citati il risultato non era stato soddisfacente, dal momento che gli algoritmi proposti avevano tempi di calcolo proibitivi e funzionavano quindi solo su istanze di dimensioni limitate. Il punto centrale della nostra tesi è stato proprio quello di **realizzare algoritmi efficienti**, pur se euristici, confrontandoli poi con i precedenti in termini di tempo di calcolo e risultati raggiunti: il confronto ha mostrato la bontà delle scelte da noi fatte.

Il lavoro di tesi è organizzato come segue.
Nel **primo capitolo** abbiamo descritto le caratteristiche del problema, la collocazione all'interno della letteratura, la sua formalizzazione; sono inoltre riportati (come parametri del problema) alcuni dati utilizzati nei lavori precedenti.

Nel **secondo capitolo** vengono presentati brevemente gli strumenti teorici utilizzati in [3] e in generale l'approccio al problema della Ricerca Operativa.

Il **terzo capitolo** è dedicato alla formulazione matematica del problema. Quanto descritto fin qui ha lo scopo di chiarire quale sia il materiale pregresso.

Ma, come detto, la strada seguita in [3] con l'utilizzo di un metodo esatto ha prodotto risultati non molto efficaci per l'elevato costo computazionale. Il problema analizzato è infatti NP-difficile[1], il che significa che i tempi di calcolo crescono enormemente con l'aumentare delle dimensioni del problema. Per tali ragioni in questo lavoro abbiamo studiato e implementato metodi differenti, euristici e meta-euristici, ottenendo grazie a ciò buone soluzioni in tempi decisamente ridotti rispetto a quelli che sarebbero stati impiegati dal metodo esatto.

Nel **quarto capitolo** è descritto il metodo Monte Carlo nella sua accezione più generale e successivamente la sua applicazione al problema: è stata formulata una euristica basata sul metodo Monte Carlo semplice e un'altra sul metodo guidato. Nello stesso capitolo è descritta l'implementazione dei due metodi.

Nel **quinto capitolo** viene studiato e proposto un metodo meta-euristico, basato su una categoria generale di euristiche: gli algoritmi genetici. Anche in questo caso, prima viene mostrata una versione generale e teorica, poi l'applicazione alla risoluzione del problema, infine la sua implementazione.

Nel **sesto capitolo** viene presentata un'analisi dei vari metodi esposti fatta utilizzando un grafo-griglia, appositamente costruito, tramite il quale è stato possibile tarare i parametri dei vari metodi.

In conclusione, nel **settimo capitolo** vengono riportate le campagne sperimentali che sono state eseguite su differenti istanze: due istanze reali e 60 istanze casuali ottenute tramite un opportuno software di generazione.

Nell'**ultimo capitolo** esponiamo le conclusioni a cui siamo giunte e indichiamo possibili miglioramenti.

[1] Le informazioni basilari sulla complessità computazionale sono riportate nel capitolo 2.

Capitolo 1

Descrizione del problema

1.1 Il problema

Data una rete stradale, si devono determinare i percorsi da far attuare ad una squadra di $|W|$[1] ausiliari del traffico, in modo da massimizzare i profitti derivanti dalle sanzioni apportate agli autoveicoli in sosta irregolare.

La caratteristica del problema è **la curva del profitto**: esso infatti non ha un valore costante, bensì il valore si annulla nel momento in cui un controllore lo raccoglie, e ricresce fino ad un valore massimo in modo lineare, per un tempo T detto tempo di turnover [2].

Tale problema, come vedremo in seguito, è risultato NP-difficile.

La fase di definizione e concettualizzazione del problema considera come contesto di riferimento la rete stradale urbana e i relativi spazi riservati alla sosta (siano essi contigui ai margini della carreggiata o raggruppati in appositi piazzali di parcheggio); ogni tratto stradale è definito in termini di caratteristiche funzionali e geometriche (numero di stalli disponibili, senso di marcia, intersezioni, transiti consentiti); per ogni tratto della rete, inoltre, si considera una stima del tempo medio di ricambio degli autoveicoli in sosta. Sulla rete insistono gli spostamenti dei controllori, per ciascuno dei quali, noti i tempi di servizio e le durate degli spostamenti per ogni tratto controllato, si vuole definire il percorso tale da rendere massimo il profitto derivante dalle sanzioni dei veicoli in sosta irregolare.

L'obiettivo, quindi, è la ricerca dei **percorsi ottimi**, siano essi ciclici o lineari, tali cioè da massimizzare la differenza tra i profitti derivanti dalle sanzioni e i costi associati agli spostamenti dei controllori.

Nell'ambito dello studio, rientrano nella definizione di sosta irregolare i veicoli:

[1] In generale indicheremo sempre con W l'insieme dei controllori e con $|W| = k$ la numerosità di tale insieme.

[2] si veda figura 1.1

- in divieto di sosta;
- in stalli riservati (strisce blu), ma con tagliando scaduto;
- stazionanti in stalli riservati ai residenti (strisce gialle) o ai portatori di handicap.

1.2 Un confronto con la letteratura

Il problema da risolvere è facilmente rappresentabile, come già detto, tramite la Teoria dei grafi. Studiando accuratamente la letteratura riguardante questo genere di problemi è possibile verificare che esso appartiene all'ampissima famiglia di Vehicle Routing Problem (VRP). Con VRP si intende una famiglia di problemi che ha per oggetto lo studio di tecniche per la pianificazione dei percorsi di una flotta di veicoli che svolgono un servizio di distribuzione di beni materiali tra un sistema di depositi ed un insieme di clienti. Si tratta quindi di problemi in cui è necessario strutturare un sistema di percorsi, veicoli e clienti che, nel rispetto dei vincoli imposti dal servizio, minimizzi i costi totali di quest'ultimo. Talvolta il problema può riguardare anche la pianificazione degli orari, in tal caso si parla di Vehicle Routing and Scheduling Problem.
Molteplici sono le applicazioni pratiche riconducibili ad un problema di instradamento: la raccolta dei rifiuti, la consegna ed il prelievo delle merci, la pulizia delle strade mediante veicoli, la raccolta della posta nelle cassette, il servizio di scuolabus.
Sempre più spesso la pianificazione dei sistemi di distribuzione è realizzata mediante software basati su tecniche di ricerca operativa e di programmazione lineare, conseguenza, questa, del grande successo ottenuto da tali metodi nelle applicazioni pratiche.
La maggior parte dei problemi di Vehicle Routing sono problemi NP-difficili.
In riferimento all'elevata complessità del problema, gli approcci risolutivi sono, come già detto, di due tipi:

- procedure risolutive esatte, che cercano la soluzione ottima in tempi esponenziali;
- procedure risolutive euristiche, che ricercano una soluzione approssimata ma soddisfacente, in relazione ad opportuni criteri, con tempi molto inferiori.

Molto spesso si preferisce realizzare buoni algoritmi euristici, piuttosto che algoritmi esatti, per cercare di trovare il giusto compromesso tra la qualità della soluzione trovata ed il tempo di calcolo speso per ottenerla.
La risoluzione dei problemi di routing prevede la costruzione di un modello a grafo. Tali problemi possono essere modellati come problemi di Node Routing o come problemi di Arc Routing: in questi ultimi l'importanza fondamentale è rivestita dagli archi, che costituiscono la rete in cui il servizio viene svolto, come nel nostro caso; in quelli di Node Routing l'importanza fondamentale

risiede nei nodi, che in genere sono intesi come le locazioni dei clienti da servire, mentre gli archi sono sostanzialmente elementi dei percorsi che connettono i nodi.

In particolare è possibile definire il problema come un Arc Routing Problem, ovvero facente parte di quei problemi che riguardano la visita degli archi del grafo che richiedono un certo tipo di servizio. In generale questo tipo di problemi ha come obiettivo quello di minimizzare i costi legati agli spostamenti, sotto i vincoli legati alla lunghezza massima del percorso.

I problemi di Arc Routing sono definiti su un grafo $G = (V, A \cup E)$, dove $V = v_1, \cdots, v_n$ è un insieme di *vertici*, A è un insieme di *archi orientati* a_{ij} con $i \neq j$ e E è un insieme di *lati non orientati*, e_{ij}, con $i \neq j$. Se si indicano con A' e i sottoinsiemi di A ed E che devono essere attraversati, con c_{ij} e d_{ij} i costi relativi all'attraversamento di a_{ij} e e_{ij} dav_i a v_j. L'obiettivo è quello di trovare un percorso chiuso di costo minimo su G, che attraversa i sottoinsiemi di archi o lati richiesti. Tra i vari ARP troviamo ad esempio il famoso problema del Postino cinese. [3] La differenza tra i normali ARP e il problema in oggetto è che in questo caso l'obiettivo non è di minimizzare bensì di massimizzare i profitti derivanti dalle sanzioni.

Uno dei più famosi problemi su grafo che prendono in considerazione la collezione di profitti è il problema del commesso viaggiatore, TSP, che consiste nel determinare il percorso di costo minimo tale da visitare tutti i nodi e tornare al punto di partenza, ovvero determinare il minimo *ciclo Hamiltoniano* del grafo.[4]

Come già detto, esiste una generalizzazione del TSP con profitto, i Profitable Arc Routing Problems, in cui non è necessario visitare tutti i vertici del grafo, a ciascuno dei quali è associato un profitto, e il cui obiettivo è quello di massimizzare il profitto associato a ciscun vertice e minimizzare i costi associati a ciascun arco.

L'algoritmo che viene proposto per risolvere questi specifici problemi è il *branch and price*.

L'obiettivo in questo genere di problemi consiste anche nella minimizzazione dei tempi di spostamento e esiste un vincolo sulla lunghezza o sul tempo complessivo per terminare il ciclo.

1.3 Problema dei percorsi ottimi dei controllori dei parcheggi

Il problema da risolvere è molto simile all'Arc Routing Problem con profitto (Profitable Arc Routing Problem, PARP): dato un grafo sul quale sono definiti sugli archi i profitti (nel nostro caso

[3] Il problema del Postino cinese è un noto problema di Teoria dei grafi formulato dal matematico cinese Mei-Ku Kwan nel 1962; esso consiste nella creazione di un cammino ciclico di lunghezza minima in un grafo non orientato che ne attraversi tutti i suoi archi.

[4] Nella teoria dei grafi un ciclo è detto hamiltoniano se esso tocca una e una sola volta tutti i vertici del grafo.

le multe raccolte) e i costi di percorrenza, si vuole trovare un insieme di cicli che massimizzi la raccolta dei profitti meno i costi di viaggio, e soggetto a vincoli:

- dei limiti sul numero di volte in cui è possibile ripercorrere lo stesso arco ottenendo un profitto;
- la massima lunghezza dei cicli (nel nostro caso i km totali che possono percorrere gli ausiliari durante il loro orario di servizio).

  Il problema però differisce dal PARP perché il numero di volte in cui è possibile ripercorrere lo stesso arco ottenendo un profitto non è noto a priori e dipende dalla finestra temporale in cui l'arco è visitato poiché dipende dal tempo necessario affinchè si rinnovino le macchine parcheggiate nell'arco stradale considerato .

Come detto, il problema in questione, che si può definire **Parking Warden Tour Problem (PWTP)**, può essere categorizzato come un Arc Routing Problem (ARP). Numerosi problemi reali possono essere descritti come ARP, tutte le attività relative a consegna o raccolta di beni, ad esempio.
La differenza principale tra i problemi appena descritti e il PWTP riguarda i profitti: nel caso oggetto del presente lavoro la disponibilità dei profitti dipende dal tempo trascorso tra due passaggi successivi per il medesimo lato. Quindi il profitto non è noto a priori ma dipende esso stesso dalla soluzione del problema, in particolare dal turnover.

Il PWTP è un problema NP-difficile, quindi non si conoscono algoritmi che portino ad una

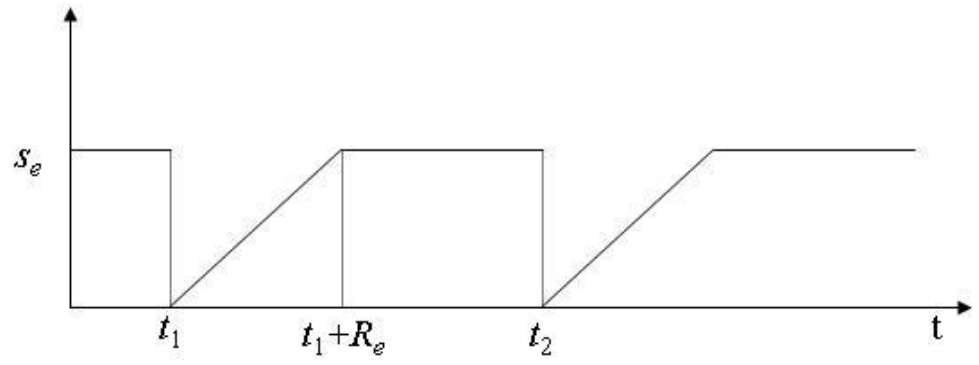

Figura 1.1: Andamento del profitto al variare del tempo passato dal momento del passaggio dell'ultimo controllore

soluzione esatta in tempo polinomiale.

1.4 Considerazioni sul turnover

Il turnover è il ricambio di autoveicoli in sosta. Se viene controllata una determinata via, al passaggio successivo sulla stessa, la situazione può presentare variazioni: quanto più la variazione è marcata, tanto più conviene che si ripassi per la via perché è maggiore la probabilità di trovare un'auto multabile, l'andamento del turnover è rappresentato in figura 1.1.
La probabilità, dato il fatto di aver controllato la via, di trovare un'auto multabile è > 0 se e solo se la composizione delle auto è mutata. Se non è mutata, la probabilità di trovare nuove auto multabili è nulla.
Quindi, definiti gli eventi:
$A =$ il controllore multa un veicolo
$B =$ ci sono stati spostamenti
avremo le seguenti probabilità:

$$P[A \mid B] \leq 0;$$

$$P[A \mid \bar{B}] = 0$$

1.5 Il grafo

Per la descrizione grafica del problema, ma anche per la sua risoluzione, è stato necessario un notevole lavoro di raccolta di dati di vario genere. Il grafo di cui si dispone è la rete stradale di Milano, che è però propria dei veicoli, è perciò stato fatto un lavoro per definire quali siano in tale rete i tratti camminabili e quali, tra questi, siano dotati di parcheggio. In generale lungo la rete stradale i parcheggi possono trovarsi:

- su un solo lato
- su entrambi i lati.

Inoltre è stato necessario andare a valutare la tipologia di parcheggi presenti in ciascuna strada, se per residenti, solo operativa o mista. Nelle tabelle riportate in Appendice di [3] è possibile reperire tutti i dati necessari.
Il grafo deve rappresentare, come già detto, il percorso camminabile, poiché è necessario andare a definire i percorsi che possono essere fatti dai controllori. Inoltre esso deve tenere conto dei vari modi in cui il controllore può ispezionare una strada, se su un lato solo o su entrambi i lati della carreggiata.

Per questo, per distinguere queste possibilità, è stato operato uno sdoppiamento dei lati, sia quelli lungo i quali avviene il parcheggio delle autovetture, sia quelli atti a congiungere i due lati della strada, e conseguentemente degli archi corrispondenti. Nella procedura di sdoppiamento dei lati

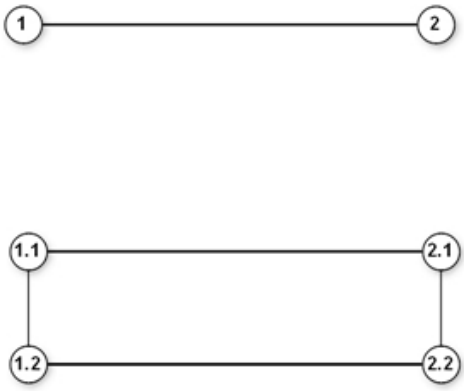

Figura 1.2: Esempio di sdoppiamento di un lato e vertici relativi

Figura 1.3: Esempio di sdoppiamento di un incrocio

è necessario considerare esclusivamente i lati percorribili a piedi.

1.6 Il modello

La definizione del problema ha portato alla stesura di un modello matematico che, in quanto tale, è in grado di fornire una soluzione esatta del problema di riferimento.
Per definire correttamente il modello matematico che descrive il problema di interesse, è necessario inquadrare le varie caratteristiche del problema.

Dati

- la squadra di controllori: il numero di agenti adibiti al controllo della sosta, di cui è necessario conoscere le specifiche di servizio (ore di servizio, estensione della pausa pranzo, etc.), possibilmente arricchite da indicazioni in merito alle abitudini del lavoratori (se sono soliti percorrere parti del tragitto insieme, se fanno pausa pranzo insieme, se sono soliti fermarsi con regolarità) e il tempo che impiegano generalmente a multare un autoveicolo;
- la rete urbana: i tempi di turn-over, cioè di ricambio degli autoveicoli in sosta e la capacità in termini di stalli/posti disponibili, sui singoli tratti della rete e presso i parcheggi;
- la durata dei percorsi;
- la stima del numero di autovetture parcheggiate irregolarmente;
- i profitti derivanti dalla singola sanzione ed il numero di sanzioni su un arco di tempo ragionevole, sulla rete o su una sua parte.

Si vuole scegliere il percorso di ciascun controllore con l'obiettivo di massimizzare le entrate derivanti dalle sanzioni.

Capitolo 2

L'approccio della Ricerca Operativa ai problemi decisionali

Con il termine Ricerca Operativa (traduzione dell'espressione inglese Operations research) si indica la disciplina che ha per oggetto la ricerca del modo migliore per condurre un'operazione. Si tratta di una disciplina nata in ambito militare durante la Seconda Guerra mondiale ma, a partire dall'immediato dopoguerra, i suoi metodi hanno iniziato ad essere utilizzati anche in altri contesti, quali la logistica, la produzione, i trasporti, le telecomunicazioni e la finanza.

A differenza di altre discipline scientifiche che si occupano di descrivere le soluzioni di un problema e di studiarne le proprietà, la ricerca operativa non ha alcuna finalità descrittiva e non si occupa dei problemi con un'unica soluzione. Essa, infatti focalizza la sua attenzione sullo studio e sullo sviluppo di metodi e di strumenti quantitativi per la soluzione di problemi decisionali e di ottimizzazione. Con il termine problema decisionale si intende un problema per il quale esistono diverse alternative o soluzioni e per il quale è possibile definire un criterio di valutazione o obiettivo che consenta di confrontare le diverse soluzioni possibili al fine di individuare la migliore.

L'approccio alla risoluzione di problemi decisionali può essere rappresentato con un diagramma di flusso in cinque fasi, come in figura 2.1
La fase 1 (problema) consiste appunto nell'individuazione del problema, al fine di capire se si tratta effettivamente di un problema decisionale e quali sono i suoi elementi variabili e quali invece sono invariabili, sui quali dunque non si può prendere alcuna decisione. In questa prima fase è necessario procedere con la raccolta dei dati, alcuni dei quali sono da subito disponibili, mentre per altri, legati a fenomeni che devono ancora verificarsi, bisogna effettuare delle stime basate sugli andamenti della serie storica o su previsioni. Può anche succedere che alcuni dati, benchè noti, non possano essere divulgati, ad esempio per ragioni di privacy o se coperti da segreto aziendale per

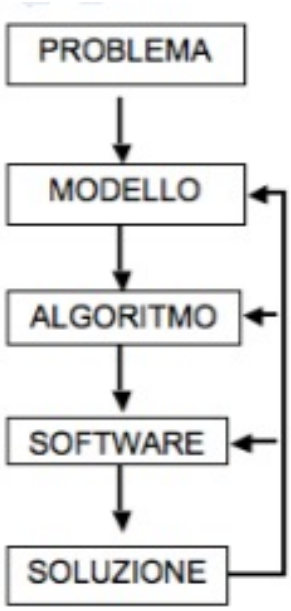

Figura 2.1: Diagramma di flusso di un processo decisionale

evitare di avvantaggiare la concorrenza. In tali ultime ipotesi, si devono costruire dei generatori casuali di dati, che forniscano istanze verosimili del problema.
La fase 2 (modello) consiste nella costruzione del modello matematico, ossia nell'astrazione e schematizzazione del problema di cui si tratta.
Nella fase 3 (algoritmo) dev'essere scelta una procedura per risolvere il modello matematico precedentemente sviluppato, e questa procedura si chiama appunto algoritmo. Attraverso di esso i dati in entrata vengono elaborati e trasformati in dati in uscita in grado di risolvere il problema oggetto di studio.
Nella fase 4 l'algoritmo viene implementato mediante un apposito software o linguaggio di programmazione.
La fase 5 (soluzione) consiste nell'analizzare i risultati ottenuti, per valutare la bontà del modello utilizzato ed eventuali errori commessi nelle fase precedenti. In tale ultima ipotesi si dovrà tornare alla fase nella quale è stato commesso l'errore e ripartire da quel punto.

Più precisamente nella formulazione di un modello di programmazione matematica sono tre i punti di fondamentale importanza:

- individuazione delle decisioni che interessano il problema in esame;
- determinazione dell'obiettivo da ottimizzare;
- definizione delle soluzioni ammissibili.

2.1 Programmazione Lineare

Uno dei modelli di programmazione è quello della programmazione lineare (PL). Definito un generico problema di PL:

$$\min f(x)$$

$$\text{s.v.} \qquad \underline{x} \in X \subseteq \Re^n$$

$$X = \{\underline{x} \in \Re : g_i(\underline{x} \geq 0, i \in \{1, \cdots, n\})\}$$

$$g_i \; lineari \; \forall i = 1, \cdots, n$$

esso può anche essere definito come segue:

$\min f(x) = c_1 x_1 + \cdots + c_n x_n$

$s.v.$

$\mathrm{a}_{11} x_1 + \cdots + a_{1n} x_n \leq b_1$

$\vdots$

$a_{m1} x1 + \cdots + a_{mn} x_n \leq b_m$

dove:

- le a_{ij}, b_i e c_j sono costanti date;
- le x_i sono le variabili di decisione, $x \in \Re^n$: grandezze numeriche il cui valore individua una soluzione del problema;
- la $f(x)$ è la cosidetta funzione obiettivo, $f(x) : X \to \Re$
- le restrizioni sono date dai vincoli, che definiscono la regione ammissibile, ovvero $X \subseteq \Re^n$: distingue le soluzioni ammissibili da quelle non ammissibili.

X è la regione delle soluzioni ammissibili, ovvero l'insieme dei punti di $\Re^n$ che rispettano i vincoli del problema. In un problema di programmazione matematica lineare, la regione delle soluzioni ammissibili è costituita da un politopo.[1]

È possibile riscrivere il problema in forma matriciale come segue:

$\min z = \underline{c^T x}$

$A\underline{x} \leq \underline{b}$

$\underline{x} \geq 0$

[1] Un politopo è l'equivalente n dimensionale dell'oggetto geometrico che in 2 dimensioni è detto poligono e in 3 poliedro. In particolare in questo caso vengono trattati i politopi convessi.

dove:

$\underline{c} = c_1 \cdots c_n$

$\underline{x} = x_1 \cdots x_n$

$\underline{b} = b_1 \cdots b_m$

e quindi $x*$ è la soluzione ottima se $\underline{c^T x*} \leq \underline{c^T x} \forall \underline{x} \in X$.

L'interesse della ricerca operativa è trovare, attraverso la programmazione lineare, una soluzione globalmente ottima, ovvero:

$x*$ tale che $f(x*) \leq f(x)\ \forall x \in X$ nel caso in cui l'obiettivo sia quello di minimizzare la funzione obiettivo, $f(x*) \leq f(x)\ \forall x \in X$ nel caso in cui l'obiettivo sia quello di massimizzare la funzione obiettivo.

Non sempre è possibile ottenere una soluzione, ci si può infatti trovare nelle seguenti situazioni:

1. la soluzione sia non ammissibile, $X = 0$;
2. la soluzione sia illimitata;
3. si ottiene un numero molto alto di soluzioni ottime.

Affinchè un problema sia descrivibile attraverso la programmazione matematica, devono essere verificate le seguenti proprietà:

1. proporzionalità: il contributo dovuto ad ogni variabile deve essere proporzionale alla variabile stessa, quindi deve essere della forma $costante \times variabile$;
2. additività: il contributo dovuto a tutte le variabili deve essere equivalente alla somma dei contributi dovuti a ciascuna variabile;
3. parametri: tutti i parametri presenti nella descrizione del modello devono essere costanti.

Le prime due derivano dalla necessità che il modello sia lineare, ovvero che siano lineari la funzione obiettivo e tutte le variabili decisionali; l'ultima deriva dal concetto di divisibilità, per cui è possibile che le variabili assumano valore frazionario.

2.2 Programmazione Lineare Intera

Una sezione particolare della programmazione matematica lineare è costituita da quella intera (PLI o MILP). I problemi che sono oggetto della MILP sono tutti quelli che hanno, tra i vincoli, l'interezza delle variabili e della funzione obiettivo.

$\min z = \underline{c^T x}$

$A\underline{x} \leq \underline{b}$

$\underline{x} \geq 0$ interi

con a_{ij}, c_j, b_i interi

Il problema equivalente, in cui non vi è il vincolo di interezza, è detto problema di *rilassamento continuo*; è evidente che $X_{PL} \supseteq X_{PLI}$. Quindi il problema di programmazione lineare fornisce un limite inferiore al valore ottimo del problema di programmazione lineare intera. La regione delle soluzioni ammissibili di un problema di PLI è costituita da un reticolo. Esistono molteplici formulazioni di uno stesso problema di questo tipo, al contrario del caso continuo; la *formulazione ideale* è quella in cui i vertici del politopo del rilassamento continuo sono interi: in questa formulazione i problemi intero e continuo hanno di fatto la stessa soluzione.

La teoria ci assicura che esiste sempre una formulazione ideale, ma essa può contenere un numero elevatissimo di vincoli; quindi benchè essa renda il problema computazionalmente più complesso, è sempre possibile scrivere un problema di MILP nella formulazione ideale, e quindi avere come soluzione quella del rilassamento continuo corrispondente.

La maggior parte dei problemi di programmazione intera sono NP-difficili; al contrario, nel continuo il noto metodo del simplesso è in grado di arrivare alla soluzione ottima in tempo polinomiale.

Per i problemi di MILP esistono diverse categorie di metodi per la risoluzione:

- enumerazione implicita;
- piani di taglio;
- euristiche;
- metaeuristiche.

Le prime due forniscono ottimi globali, le ultime ottimi locali.

I metodi di enumerazione implicita esplorano tutte le soluzioni ammissibili, esplicitamente e implicitamente: tra questi vi sono il metodo detto di Branch and Bound, e la programmazione dinamica.

2.3 La complessità computazionale

Una grandezza da tenere in considerazione nei problemi di ottimizzazione è la complessità computazionale, ovvero la grandezza che misura la quantità di risorse necessarie ad un algoritmo per giungere alla soluzione del problema.

Generalmente la risorsa che viene considerata per valutare la complessità di un problema è il tempo e il parametro rispetto al quale lo si valuta è la dimensione del problema: più precisamente si valuta quanto aumenta il tempo di calcolo all'aumentare delle dimensioni dell'istanza del problema.

Possono inoltre essere utilizzati differenti criteri di valutazione:

- il caso ottimo: i dati sono i migliori possibili per l'algoritmo, cioè quelli che richiedono meno elaborazioni per essere trattati;
- il caso peggiore: i dati richiedono il massimo numero di passi per l'algoritmo;
- il caso medio: fornisce un reale indicatore della complessità dell'algoritmo perché considera i dati medi.

Dalla misurazione delle risorse si possono costruire le classi di complessità. Definiamo le seguenti classi di complessità computazionale, relativamente a problemi di tipo decisionale, ovvero problemi la cui soluzione è una risposta del tipo sì/no:

- classe dei problemi P: problemi per cui esiste un algoritmo che per ogni istanza determina in tempo polinomiale la risposta;
- classe dei problemi NP: problemi per cui esiste un algoritmo che per ogni istanza verifica in tempo polinomiale se la risposta è sì;
- classe dei problemi NP-difficili: problemi non meno difficili dei problemi NP-completi.

Definizione: un problema decisionale R si dice NP-difficile se ogni problema $Q \in NP$ è riconducibile polinomialmente a R.
Definizione: un problema decisionale si dice NP-completo se appartiene alla classe di problemi NP ed è NP-difficile.
È evidente che la complessità computazionale è una caratteristica fondamentale di un algoritmo: se all'aumentare delle dimensioni dell'istanza il tempo di calcolo necessario al calcolatore per fornire la soluzione del problema cresce eccessivamente, questo può essere fonte di problemi, laddove si voglia utilizzare l'algoritmo anche per istanze molto grandi. Per tale motivo, nel caso si debba risolvere un problema per il quale non si conosce un algoritmo in grado di fornire una soluzione in tempo polinomiale rispetto all'istanza, si preferisce utilizzare altri metodi, magari non esatti ma più rapidi nel calcolare una soluzione accettabile. Un esempio sono i metodi euristici e metaeuristici.

2.4 Le euristiche

Accanto a tutti quelli che vengono chiamati metodi esatti, ovvero tutti quegli algoritmi che forniscono la soluzione ottima del problema che viene loro sottoposto, vi sono gli algoritmi euristici. Un algoritmo euristico risolve un problema di ottimizzazione, utilizzando in genere regole di buon senso, e fornisce una soluzione ammissibile, non necessariamente ottima.
Esistono diversi tipi di euristiche:

- **Euristiche costruttive**: costruiscono una soluzione del problema

- **Euristiche di miglioramento**: iterativamente a partire da una soluzione del problema cercano di determinarne una migliore nell'intorno della data.

In genere le soluzioni fornite dalle euristiche costruttive e quelle utilizzate dalle euristiche di miglioramento sono soluzioni ammissibili. Per certi problemi può risultare estremamente complesso determinare una prima soluzione ammissibile: in questi casi si può rilassare lagrangianamente alcuni vincoli, i.e., determinare una prima soluzione che soddisfi almeno parte dei vincoli e penalizzare il fatto che altri vincoli non siano rispettati. Quindi a partire dalla soluzione non ammissibile ottenuta, si cerca iterativamente di determinarne una che rispetti maggiormente i vincoli nell'intorno di quella già in possesso.
Affinchè un'euristica sia efficace, essa deve sfruttare le caratteristiche strutturali del problema da risolvere. Per questo motivo non è possibile definire un'euristica generale che vada bene per qualsiasi problema.
Esistono però degli approcci generali a cui si può fare riferimento per sviluppare euristiche specifiche. Tali approcci vengono detti metaeuristiche.

2.5 Le metaeuristiche

Le metaeuristiche sono algoritmi approssimati, utilizzati in applicazioni di ottimizzazione combinatoria e problemi di soddisfacimento di vincoli. La loro principale caratteristica risiede nel fatto che, al contrario degli algoritmi completi, essi non forniscono in tempo finito una soluzione ottima, nel caso di problemi di ottimizzazione, o una soluzione ammissibile, se esiste, nel caso di problemi di soddisfacimento di vincoli.
Tuttavia, essi sono caratterizzati da una notevole efficienza e in molte applicazioni risultano superiori in termini di prestazioni, rispetto agli algoritmi completi. Inoltre, nella maggior parte dei casi l'obiettivo da raggiungere nelle applicazioni non è una soluzione ottima, ma una soluzione di alta qualità di valore prossimo all'ottimo, o semplicemente una soluzione migliore di quella di cui si dispone in un determinato istante, in un tempo ragionevole. In queste situazioni gli algoritmi incompleti hanno tipicamente prestazioni superiori a quelli completi, essendo in grado di fornire soluzioni di buona qualità in tempi brevi rispetto agli enormi tempi di calcolo richiesti dagli algoritmi completi per garantire l'ottimalità della soluzione, quando la dimensione del problema diventa grande.
Tra le strategie definite sono da ricordare:

- tabu search;
- simulated annealing;

- algoritmi di ricerca quali gli algoritmi genetici e ant colony optimization.

Le metaeuristiche hanno conosciuto ampio sviluppo negli ultimi decenni, sia dal punto di vista teorico che da quello applicativo.
Vediamo alcuni di questi algoritmi:

Tabu Search: la tabu search è una metaeuristica di ricerca locale che permette passi in cui con criteri deterministici avvengono dei peggioramenti.
Vengono inoltre definiti dei criteri di accettazione della soluzione generata basati non necessariamente solo sul valore della funzione obiettivo.
Poiché si accettano anche soluzioni che inducono peggioramenti della funzione obiettivo, per evitare di ritornare su soluzioni già visitate, ad ogni passo viene aggiornata una lista di operazioni vietate (tabù), che impediscono di rieffettuare passi già fatti;

Simulated Annealing: la simulated annealing è un algoritmo per la soluzione di problemi combinatori NP-difficili basato sulla metafora dell'annichilazione dei metalli: si raggiunge uno stato di minima energia non troppo velocemente, in modo da evitare che si raggiunga una struttura cristallina finale di energia non minima, che rappresenta nel modello un minimo locale; la simulated annealing è una metaeuristica di ricerca locale che permette passi in cui avvengono dei peggioramenti con probabilità che decresce nel tempo;

Algoritmi Genetici, Ant Colony: sono algoritmi basati su fenomeni naturali quali la selezione naturale, l'organizzazione di colonie di formiche, etc. Essi hanno ottenuto qualche successo su problemi specifici, ma non sembrano in generale essere superiori a quelli basati su tabù search ed in seconda istanza sulla simulated annealing;

Scatter Search (SS): è un algoritmo che sfrutta soluzioni prese casualmente combinato con un algoritmo di ricerca locale.

2.6 Teoria dei grafi

Molti problemi decisionali possono essere formulati utilizzando il linguaggio della teoria dei grafi, come ad esempio problemi di trasporto, organizzazione degli orari o localizzazione di servizi.
Benchè si presenti tipicamente in forma grafica, la teoria dei grafi può anche essere descritta tramite una formulazione matematica: programmazione lineare, lineare intera o non lineare.
Inoltre esistono numerosi algoritmi che permettono di risolvere specifici problemi sul grafo.

Definizioni di base

Grafo orientato

- Un *grafo orientato* $G = (N, A)$ consiste in un insieme di N nodi e un insieme A di archi i cui elementi sono coppie ordinate di nodi. Una *rete* è un grafo in cui ai nodi e/o agli archi si associano valori numerici.
- Un arco diretto (i, j) ha due estremi: la *coda* i e la *testa* j. L'arco è *incidente* in i e j, è un *arco uscente* di i e un *arco entrante* di j. Se (i, j), allora j è *adiacente* a i.
- Il *grado* di un nodo è il numero di archi entranti e uscenti del nodo.

Cammini e cicli orientati

- Un *cammino orientato* in un grafo $G = (N, A)$ è una sequenza di nodi e archi $i_1 - a_1 - i_2 a_2 - \cdots - a_{r-1} - i_r$, senza ripetizioni, e tale che $a_k = (i_k, i_{k+1})$, per $k = 1, \cdots, r - 1$.
- Un *ciclo orientato* è un cammino orientato in cui il primo e l'ultimo nodo coincidono.
- Un grafo è *aciclico* se non contiene cicli.
- Un grafo è *fortemente connesso* se contiene almeno un cammino diretto tra ciascuna coppia di nodi.

Grafo non orientato

- Un *grafo non orientato* $G = (V, E)$ consiste un un insieme V di vertici e un insieme E di lati i cui elementi sono coppie non ordinate di nodi.
- Il *grado* di un vertice è il numero di lati ad esso incidenti.

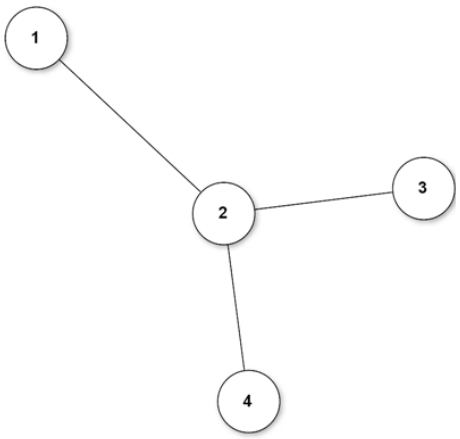

Figura 2.2: Un grafo non orientato

Cammini e cicli

- Un *cammino* in un grafo non orientato è una sequenza di vertici e lati $i_1 - e_1 - i_2 - e_2 - \cdots - i_r$, senza ripetizioni di vertici e tale che $e_k = \{i_k, i_{k+1}\}$ per $k = 1, \cdots, r-1$.
- Un *ciclo* in un grafo non orientato è un cammino in cui il primo e l'ultimo vertice coincidono.

È possibile dunque formulare attraverso la Teoria dei Grafi dei problemi di Ricerca Operativa, ad esempio problemi di flusso massimo lungo il grafo, nel caso in cui agli archi siano associati dei valori numerici, o problemi di cammini minimi, per i quali la teoria dispone di molteplici algoritmi.

Capitolo 3

Il modello di programmazione lineare intera

Il Parking Warden Tour Problem viene modellizzato su un grafo $G = (V, E)$ non orientato, con V insieme di nodi $|V| = n$ e E insieme dei lati, $|E| = m$.

Parametri

$W \equiv$, $W = 1 \cdots k$ insieme di controllori nei parcheggi;

$T \equiv$ tempo di servizio di un controllore;

$c_e \equiv$ tempo di viaggio a piedi sul lato e;

$\sigma_e \equiv$ tempo totale per ispezionare i veicoli in sosta sul lato e;

$q \equiv$ tempo per sanzionare un veicolo;

$p \equiv$ profitto per ciascun veicolo in sosta irregolare;

$s_e \equiv$ stima del numero di veicoli in sosta irregolare su e;

$R_e \equiv$ stima del tempo di tournover sul lato e.

Questi parametri sono tutti i dati a disposizione al fine di ottenere l'ottimizzazione dei percorsi dei $|W|$ controllori della sosta.

Per la formulazione del modello, si considerano inoltre le seguenti assunzioni:

- i guardiani partono tutti dal deposito;
- i guardiani terminano tutti il loro percorso al deposito.

Chiaramente non è necessario che tutti i controllori partano e tornino contemporaneamente, inoltre essi dispongono di un tempo di servizio T. È necessario inoltre fare alcune ipotesi:

1. un tragitto può essere percorso senza ispezionare le auto e raccogliendo gli introiti corrispondenti;

2. il percorso non può essere interrotto;

3. il rendimento di un percorso va a zero dopo l'ispezione.

Variabili

Le variabili del problema vengono distinte in tre tipologie:

- variabili di instradamento;
- variabili di profitto;
- variabili di tempo.

Variabili di instradamento

Le variabili di instradamento sono variabili binarie e indicano la visita da parte di un controllore di un nodo v o di un lato e.

$$x_{ekw} = \begin{cases} 1 & \text{se e è il k-esimo lato visitato dal controllore w} \\ 0 & \text{altrimenti} \end{cases}$$

$\forall$e$\in E$, $\forall$k=1,$\cdots, K$, $\forall$w=1,$\cdots, W$

$$z_{ekw} = \begin{cases} 1 & \text{se e è il k-esimo lato visitato dal controllore w con profitto} \\ 0 & \text{altrimenti} \end{cases}$$

$\forall$e$\in E$, $\forall$k=1,$\cdots, K$, $\forall$w=1,$\cdots, W$

$$v_{ikw} = \begin{cases} 1 & \text{se } i \text{ è il } k-esimo \text{ nodo visitato dal controllore } w \\ 0 & \text{altrimenti} \end{cases}$$

$\forall$i$\in V$, $\forall$k=1,$\cdots, K$, $\forall$w=1,$\cdots, W$

con

$$K = \frac{T}{\min_e * c_e}$$

Variabili di profitto

π_{kw} $\forall$k=1,$\cdots, K$, $\forall$w=1,$\cdots, W$

Variabili di tempo

t_{kw} con $0 \leq t_{kw} \leq T$
$\forall$k=1,$\cdots$, K, $\forall$w=1,$\cdots$, W

Variabile continua che misura il tempo trascorso da quando un controllore w ha lasciato il deposito.

$$y_{k'k''w'w''} = \begin{cases} 1 & \text{se il } k' - esimo \text{ lato visitato da } w' \text{ coincide con il } k'' - esimo \text{ lato visitato da } w'' \text{ e } w' < w'' \\ 0 & \text{altrimenti} \end{cases}$$

$\forall$k',k"=1,$\cdots$, K, $\forall$w',w"=1,$\cdots$, $W : w''\ w'$

Modello - Funzione Obiettivo

$$\max p \sum_{w=1}^{W} \sum_{k=1}^{K} \pi_{kw} \tag{3.1}$$

Nella funzione obiettivo compaiono solamente le variabili π_{kw}; esse infatti assumono il valore del profitto in relazione al tempo passato tra due passaggi successivi per il medesimo lato e in relazione al numero di veicoli in sosta irregolare. In 3.1, l'assegnazione del valore a π_{kw} avviene tramite i vincoli da (4.13) a (4.19). Viene massimizzata la sommatoria tra tutti i controllori w e tra tutti i passaggi k di ogni controllore.

Modello - I vincoli

$$\sum_{e \in \delta(0)} x_{e1w} = 1 \qquad \forall w = 1, \ldots, W \tag{3.2}$$

$$\sum_{e \in E} x_{ekw} \leq 1 \qquad \forall k = 2, \ldots, K,\ \forall w = 1, \ldots, W \tag{3.3}$$

$$v_{01w} = 1 \qquad \forall w = 1, \ldots, W \tag{3.4}$$

$$\sum_{i \in V} v_{ikw} \leq 1 \qquad \forall k = 2, \ldots, K+1,\ \forall w = 1, \ldots, W \tag{3.5}$$

$$\sum_{i:\{i,j\} in E} v_{ik+1w} \geq v_{jkw} \qquad \forall j \in V,\ \forall k = 1, \ldots, K,\ \forall w = 1, \ldots, W \tag{3.6}$$

$$\sum_{k=2}^{K} v_{0kw} = 1 \qquad \forall w = 1, \ldots, W \tag{3.7}$$

$$\sum_{i\in N}\sum_{k'=k+1}^{K+1} v_{ik'w} \leq (K-k+1)(1-v_{0kw}) \qquad \forall k=2,\ldots,K, \forall w=1,\ldots,W \quad (3.8)$$

$$x_{ekw} \geq v_{ikw} + v_{jk+1w} + v_{ik+1w} + v_{jkw} - 1 \qquad \forall e=\{i,j\}\in E, \forall k=1,\ldots,K, \forall w=1,\ldots,W$$

$$x_{ekw} \leq v_{ikw} + v_{jkw} \qquad \forall e=\{i,j\}\in E, \forall k=1,\ldots,K, \forall w=1,\ldots,W \quad (3.9)$$

$$x_{ekw} \leq v_{jk+1w} + v_{jk+1w} \qquad \forall e=\{i,j\}\in E, \forall k=1,\ldots,K, \forall w=1,\ldots,W \quad (3.10)$$

$$z_{ekw} \leq x_{ekw} \qquad \forall e=\{i,j\}\in E, \forall k=1,\ldots,K, \forall w=1,\ldots,W \quad (3.11)$$

$$\sum_{k=1}^{K}\sum_{e\in E}(c_e x_{ekw} + q\pi_{kw} + \sigma_e z_{kw}) \leq T \qquad \forall w=1,\ldots,W \quad (3.12)$$

$$t_{1w} = 0 \qquad \forall w=1,\ldots,W \quad (3.13)$$

$$t_{kw} \geq t_{k-1w} + \sum_{e\in E}(c_e x_{ek-1w} + q\pi_{wk} + \sigma_e z_{ek-1w})$$

$$\forall k=2,\ldots,K+1, \forall w=1,\ldots,W \quad (3.14)$$

$$\pi_{kw} \leq \sum_{e\in E} s_e z_{ekw} \qquad \forall k=1,\ldots,K, \forall w=1,\ldots,W \quad (3.15)$$

$$\pi_{k''w} \leq s_e \frac{t_{k''w} - t_{k'w}}{R_e} + S(2 - z_{ek''w} - z_{ek'w} + \sum_{k=k'+1}^{k''-1} z_{ekw})$$

$$\forall e\in E, \forall k', k''=1,\ldots,K : k''>k', \forall w=1,\ldots,W : w'<w'' \quad (3.16)$$

$$\pi_{k''w''} \leq s_e \frac{t_{k''w''} - t_{k'w'}}{R_e} + S(1 + \frac{T}{R_e}(3 - z_{ek''w''} - z_{ek'w'} - y_{k'k''w'w''})$$

$$\forall e\in E, \forall k', k''=1,\ldots,K : k''>k', \forall w', w''=1,\ldots,W : w'<w'' \quad (3.17)$$

$$\pi_{k'w'} \leq s_e \frac{t_{k'w'} - t_{k''w''}}{R_e} + S(1 + \frac{T}{R_e})(2 - z_{ek''w''} - z_{ek'w'} + y_{k'k''w'w''})$$

$$\forall e\in E, \forall k', k''=1,\ldots,K : k''>k', \forall w', w''=1,\ldots,W : w'<w'' \quad (3.18)$$

$$x_{ekw} \geq 0 \qquad \forall e \in E, \forall k = 1, \ldots, K, \forall w = 1, \ldots, W \quad (3.19)$$

$$y_{k'k''w'w''} \in \{0,1\} \qquad \forall k', k'' = 1, \ldots, K, \forall w', w'' = 1, \ldots, W : w' < w'' \quad (3.20)$$

$$v_{ikw} \in \{0,1\} \qquad \forall k = 1, \ldots, K+1, \forall w = 1, \ldots, W \quad (3.21)$$

$$z_{ekw} \in \{0,1\} \qquad \forall e \in E, \forall k = 1, \ldots, K, \forall w = 1, \ldots, W \quad (3.22)$$

$$0 \leq t_{kw} \leq T \qquad \forall k = 1, \ldots, K+1, \forall w = 1, \ldots, W \quad (3.23)$$

$$y_{k'k''w'w''} \leq 3 + \frac{t_{k''w''} - t_{k'w'}}{T} - z_{ek'w'} - z_{ek''w''}$$

$$\forall e \in E, \forall k', k'' = 1, \ldots, K, \forall w', w'' = 1, \ldots, W : w' < w'' \quad (3.24)$$

$$y_{k'k''w'w''} \geq -2 + \frac{t_{k''w''} - t_{k'w'}}{T} + z_{ek'w'} + z_{ek''w''}$$

$$\forall e \in E, \forall k', k'' = 1, \ldots, K : k'' > k', \forall w', w'' = 1, \ldots, W : w' < w'' \quad (3.25)$$

Il complesso sistema vincolare rappresenta il cuore del modello.

- Il vincolo 3.2 obbliga tutti i guardiani ad uscire dal deposito;
- il vincolo 3.3 limita superiormente le variabili x;
- il vincolo 3.4 è l'analogo del vincolo 3.2 ma sui nodi v;
- il vincolo 3.5 rispetto al vincolo 3.3.

Il fatto che le variabili x e v siano limitate a 1 si spiega per il fatto che le sole variabili definite binarie nel modello sono le z. Le altre sono continue, ma tramite i vincoli i e per effetto delle z, con il vincolo 3.11, assumono solamente valori 1 e 0. Così facendo, si limita il numero di variabili binarie nel modello, a patto però di aggiungere vincoli.

- I vincoli 3.6, 3.7, 3.8, 3.9 e 3.10 garantiscono la continuità ai nodi.
- Il vincolo 3.12 impedisce che il percorso di un controllore possa eccedere il tempo di servizio, considerando il contributo degli attraversamenti di tutti i lati visitati, tramite la variabile x_{ekw} e il parametro c_e, dei soli lati visitati con profitto, tramite la variabile z_{ekw} e il parametro σ_e e delle perdite di tempo per controllare i veicoli in sosta, tramite la variabile π_{kw} e il parametro q;

- Il vincolo 3.13 inizializza a 0 i tempi di ogni ausiliario;
- Il vincolo 3.14 rappresenta un'equazione di continuità rispetto agli istanti di visita del lato e da parte di w, imponendo che l'istante di visita successiva sarà almeno uguale all'istante precedente, nel caso in cui il controllore non si sposti, a partire chiaramente dal secondo lato visitato dopo quello prossimo al deposito;
- Il vincolo 3.15 impone che i veicoli multati π_{kw} siano al più uguali alla somma di tutti i veicoli in sosta irregolare controllati come $k - esimi$ da w;
- I vincoli da 3.16 a 3.24 stabiliscono che al secondo passaggio di un qualsiasi controllore su un arco e, le auto che questi potrà multare saranno pari al numero di auto in sosta irregolare su quel lato se è passato almeno un tempo pari al tempo di turnover, altrimenti saranno solo una frazione delle auto in sosta irregolare s_e. Si suppone, infatti, che il numero di auto in sosta irregolare che possono essere multate, a seguito del passaggio di un controllore, sia 0 e che esso cresca linearmente col passare del tempo, fino a rigenerarsi dopo un tempo pari al tempo di turnover R_e.

  In particolare, il vincolo 3.16 riguarda il passaggio per il medesimo lato e dello stesso guardiano w, dopo un certo tempo $t_{k''w} - t_{k'w}$; i vincoli 3.17 e 3.18 riguardano invece il passaggio sul medesimo lato, in due istanti diversi, da parte di due controllori diversi, w' e w''. È importante notare che tali vincoli riguardano solo il passaggio per un lato e per raccogliere profitto.
- I vincoli 3.24 e 3.25, infine, assegnano alle variabili $y_{k'k''w'w''}$ il valore 0 o 1 a seconda delle precedenze di passaggio per un lato e di due controllori w' e w'', come descritto in precedenza.

I vincoli da 3.16 a 3.18, sono costruiti con la tecnica del *Big - M*: a seconda del valore assegnato alle variabibli z e y, alcuni vincoli risultano ridondanti e altri risultano attivi. Il Big - M, nel caso in esame $S = \max_{e \in E} s_e$. **Complessità computazionale**

Le numerose variabili intere presenti nel modello MILP apportano un notevole contributo alla complessità computazionale, tale da rendere il problema non computazionalmente risolubile per istanze di grandi dimensioni, come quelle reali in tempi ragionevoli. Per tale ragione non si è proseguiti per la strada della risoluzione del problema in modo esatto, ma è stato risolto tramite un *algoritmo euristico*.

3.1 L'algoritmo euristico per il PWTP

Il problema principale per cui non si è in grado di risolvere il problema in modo esatto in tempi ragionevoli, è il numero $k = |W|$ di controllori. Pertanto l'idea alla base dell'euristica utilizzata è

quella di risolvere il problema con un solo controllore e considerare $|W|$ il numero di istanze del problema. Quindi vengono risolti $|W|$ problemi PWTP con un solo controllore, a cui si aggiunge il vincolo di non poter utilizzare i lati utilizzati nelle precedenti soluzioni.
La soluzione di ogni sottoproblema è soluzione ottima. La prima soluzione che viene trovata, che fornirà il percorso che massimizza in assoluto i profitti, moltiplicata per il numero di controllori, fornirà un *upper-bound* per il nostro problema.
In generale l'approccio euristico consente di ottenere un *upper-bound* della soluzione ottima con tempi di calcolo inferiori rispetto a quelli del calcolo esatto.
Tale approccio è alla base dell'articolo [1]. Vediamo in modo più chiaro l'euristica ivi descritta: poiché il motivo che rende il problema non risolubile in maniera esatta tramite il modello di programmazione lineare è principalmente il fatto che al crescere delle cardinalità delle istanze il tempo di calcolo cresce enormemente, si procede come segue:

- si calcola la soluzione esatta per un controllore;
- gli archi percorsi dal controllore vengono eliminati da quelli percorribili con profitto;
- si procede a calcolare la soluzione esatta per un secondo controllore sugli archi restanti
- si itera in questo modo il calcolo.

In tal modo si evita il problema del turnover e il valore del profitto totale, relativo a k controllori, è semplicemente la somma dei profitti dei k percorsi. I percorsi generati devono avere le seguenti caratteristiche:

- partire e terminare al deposito;
- il tempo impiegato deve essere inferiore al tempo di servizio dei controllori;
- gli archi possono essere percorsi raccogliendo il profitto oppure no;
- un arco in cui è stato raccolto il profitto viene eliminato dall'insieme degli archi su cui possono raccogliere profitto i controllori successivi;
- un controllore può ripassare su archi già attraversati, può attraversare un arco già visitato e da cui ha già raccolto il profitto, ma può raccogliere nuovamente il profitto esclusivamente nel caso in cui nell'istante in cui lo attraversa sia già trascorso il tempo di turnover.
- un controllore può star fermo in un nodo nell'attesa che passi il tempo di turnover su un lato particolarmente profittevole.

Nell'articolo sono riportate le soluzioni dell'euristica applicata a un certo numero di istanze, in particolare è indicato il gap percentuale di profitto ottenuto rispetto all'upper bound dato dal prodotto tra la soluzione del primo controllore e il numero di controllori.
L'euristica descritta in questo capitolo ha però alcuni problemi: innanzitutto l'idea di risolvere $|W|$ istanze di PWTP, ciascuna con un solo controllore ma con i vincoli suddetti, potrebbe essere una approssimazione eccessiva tale da limitare le soluzioni ammissibili e quindi non permettere di trovare soluzioni che potrebbero invece fornire un buon valore in termini di funzione obiettivo. Inoltre, non è certamente realistico non permettere a diversi controllori di attraversare gli stessi archi: per quanto per le caratteristiche del problema sarebbe intuitivamente meglio non ripassare per un arco da cui è già stato prelevato il profitto, non si può escludere a priori che riattraversare un arco visitato possa portare un aumento del profitto. Infine, nonostante il metodo sia euristico è possibile verificare che al crescere della cardinalità dell'insieme dei controllori il tempo di calcolo delle soluzioni si dilata notevolmente.
Per tali motivi si è cercato di creare dei metodi più efficienti per la risoluzione di questo problema. In particolare è stato ideato un metodo euristico che prende spunto dal metodo stocastico Monte Carlo e un metodo metaeuristico che a partire dalle soluzioni ottenute con l'euristica si comporta come un Algoritmo Genetico (AG).[1]

[1] Il metodo Monte Carlo e la metaeuristica AG sono descritti in maniera approfondita nei paragrafi successivi.

Capitolo 4

Metodo Monte Carlo

4.1 L'approccio stocastico: il Metodo Monte Carlo

Uno degli approcci utilizzati nello svolgimento del presente lavoro è quello stocastico: in particolare nei capitoli successivi verrà descritta una euristica basata sul metodo Monte Carlo, del quale viene fatta una descrizione a grandi linee in questo paragrafo. Il Metodo Monte Carlo è un metodo stocastico non parametrico che viene utilizzato per trovare le soluzioni di problemi matematici resi complessi, ad esempio a causa della presenza di un numero elevato di variabili, e che non possono essere risolti facilmente con il calcolo integrale o, come nel nostro caso, con algoritmi esatti propri della ricerca operativa.
Esso consiste nel calcolare una serie di realizzazioni possibili del fenomeno in questione in modo casuale, generando quindi una serie di simulazioni che tengano conto delle caratteristiche probabilistiche del fenomeno, e cercando di esplorare l'intero spazio dei parametri del fenomeno. Una volta generato tale campione casuale di possibili soluzioni del problema da risolvere, si procede al calcolo delle misure di interesse, e, in seguito, al calcolo del valor medio di tali misure sulle realizzazioni. La simulazione offre una buona soluzione se tale valor medio converge al valore vero. L'efficienza del metodo aumenta al crescere delle dimensioni del problema.

4.1.1 Il Metodo Monte Carlo guidato

Come già detto il Metodo Monte Carlo ben si addice ad essere applicato per la risoluzione di fenomeni di cui si conosce la distribuzione di probabilità delle soluzioni. Tale distribuzione può essere sfruttata dall'algoritmo per far sì che le soluzioni scelte non siano del tutto casuali, ma seguano la stessa distribuzione di probabilità; si parla in questo caso di generazione di soluzioni pseudo-random, o di Metodo Monte Carlo guidato.
Nel caso specifico, poiché non è nota a priori la distribuzione di probabilità delle soluzioni, si è

pensato di guidare l'algoritmo imponendo che la scelta dei cammini non fosse del tutto casuale, ma fossero privilegiati gli archi con profitto più elevato. In effetti l'utilizzo di un Metodo Monte Carlo guidato non assicura un incremento della soluzione ottima rispetto all'utilizzo del Metodo semplice, infatti la soluzione data potrebbe essere già scelta dal metodo Monte Carlo semplice.

Schema del metodo

L'applicazione del metodo può essere genericamente schematizzata come segue:

- definire un dominio di possibili dati in input;
- generare input casuali, o pseudo casuali, dal dominio, secondo una certa distribuzione di probabilità;
- eseguire un calcolo deterministico utilizzando i dati in ingresso
- aggregare i risultati dei singoli calcole, nel risultato finale.

Il requisito fondamentale affinchè un sistema possa essere modellizzato da un metodo Monte Carlo è che esso sia descrivibile tramite funzioni di densità di probabilità. Per avere un risultato significativo devono essere condotte più simulazioni, da cui viene ricavato, come già detto, un valore medio dell'osservabile che si vuole misurare e l'errore statistico associato. Le tecniche Monte Carlo mostrano la loro utilità al crescere della complessità della situazione con cui si ha a che fare.
Il cuore di una simulazione Monte Carlo è il generatore di numeri random, o più correttamente di numeri pseudorandom. Questo significa che, se l'algoritmo di generazione viene fatto iniziare sempre allo stesso punto di partenza o seme, la sequenza di numeri random ottenuta sarà identica. Questa caratteristica è fondamentale perché garantisce la riproducibilità dell'esperimento, e quindi del possibile calcolo degli errori connessi alla scrittura dell'algoritmo.

4.1.2 Applicazione del Metodo Monte Carlo

Il Metodo Monte Carlo ha svariati ambiti di applicazione, uno dei più noti è quello del calcolo integrale, per cui si utilizza tale metodo nei casi in cui la risoluzione analitica sia troppo complessa. Esso ha inoltre avuto un discreto successo nel mondo della finanza, in cui viene utilizzato per il calcolo dei portafogli ottimi, dunque per risolvere problemi di ottimizzazione. In seguito si sfrutterà questo tipo di applicazione del metodo a problemi di ottimizzazione, nel caso in questione di ottimizzazione su grafo.

ESEMPIO

Le simulazioni Monte Carlo possono essere viste come una tecnica numerica per calcolare gli integrali, vediamo come:

- si consideri una funzione a valori reali f da integrare tra i punti $a, b \in \Re^+$;
- si individui un rettangolo che contiene la funzione da integrare nell'intervallo scelto;
- si scelgano a caso un certo numero di punti all'interno del rettangolo;
- si contino quanti punti sono al di sopra e quanti al di sotto della funzione;
- l'integrale è dato dal prodotto tra area del rettangolo e porzione di punti al di sotto della funzione.

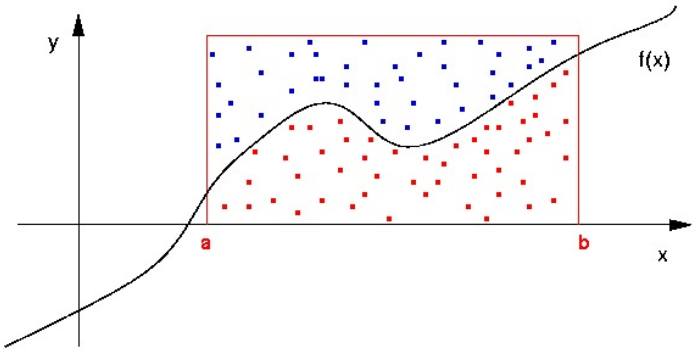

Figura 4.1: Esempio di funzione da integrare

Applicare il metodo Monte Carlo al problema di Arc Routing significa creare un certo numero di cicli sul grafo in modo, del tutto o parzialmente, casuale. Questo significa che è possibile ottenere qualsiasi soluzione ammissibile, con uguale probabilità nel caso random, con probabilità diverse nel caso pseudorandom. Diversamente dall'euristica presentata nell'ultimo paragrafo del capitolo precedente, tale euristica ha lo scopo di creare un certo numero di soluzioni ammissibili e realistiche, eliminando quindi quel vincolo che impedisce ad un controllore di ripassare per un arco già visitato da un collega.
Lo schema più generale dell'algoritmo che verrà descritto, dato il grafo con tutte le caratteristiche fornite nel modello MILP, è il seguente:

1. si generano n percorsi possibili che verificano i vincoli, in modo casuale;
2. si combinano $|W|$ percorsi in modo casuale;
3. si calcola il profitto di ciascuna soluzione tenendo conto del turnover.

4.2 La scelta delle soluzioni migliori

La **soluzione** del problema è costituita da una combinazione di k percorsi, dove si ricorda che $k = |W|$ è il numero di controllori. È stato già detto che il profitto in tale problema deriva dal

sanzionamento delle automobili parcheggiate in maniera irregolare. Esso è una variabile che varia nel tempo, ma non solo; essa dipende anche dal passaggio dei controllori stessi: nel momento in cui un controllore percorre una strada, che corrisponde ad un arco del nostro grafo, raccogliendo in tale strada le sanzioni delle automobili parcheggiate irregolarmente, il profitto caratteristico di tale arco del grafo si annulla, per poi ricominciare ad aumentare con il passare delle ore, fino ad un valore massimo, rappresentato dal valore pieno del profitto. Il tempo impiegato da un arco per tornare ad avere profitto pieno è detto turnover ed è caratteristico di ciascun arco.
Pertanto un passaggio fondamentale dell'algoritmo di risoluzione consiste nello scegliere i percorsi che vadano a formare le soluzioni vere e proprie, di cui poi si deve valutare il profitto effettivo. È evidente che, generato un alto numero di soluzioni ammissibili, non è ragionevole creare tutte le possibili combinazioni di soluzioni. Ciò per due motivi: in primo luogo il numero di soluzioni sarebbe eccessivamente elevato, visto che dati n percorsi e k controllori, il numero di soluzioni è dato dal coefficiente binomiale

$$n!/k!(n-k)!$$

il che significa che, ad esempio, per 100 percorsi generati avremmo 161700 soluzioni, il che si tradurrebbe a livello implementativo in un enorme impiego di tempo da parte del calcolatore. In secondo luogo, dato che lo scopo è quello di massimizzare il profitto e data la caratteristica fondamentale del problema, ossia la funzione del profitto nel tempo, il turnover, è importante che i controllori attraversino il minor numero possibile di archi uguali, o comunque che ripassino per archi già attraversati solamente dopo che è trascorso un tempo sufficiente a riformare su tale arco un profitto. Per tale motivo è stato scelto di non creare tutte le possibili combinazioni di k percorsi, ma di andare a creare delle soluzioni che fossero costituite da k percorsi il più possibile differenti. Per fare ciò, una volta generati i percorsi, si è proceduto a farne una clusterizzazione che producesse k insiemi distinti di percorsi, e sono state generate le soluzioni prendendo i k percorsi ognuno da un cluster differente.[1]

4.3 Risoluzione del problema

Nel primo paragrafo è già stata data una descrizione approssimativa del problema da affrontare: calcolare i cammini ottimi per una squadra di $|W|$ ausiliari della sosta, al fine di massimizzare i profitti derivanti dalle sanzioni fatte alle autovetture parcheggiate irregolarmente. Si sono già visti i **vincoli** che le soluzioni, ovvero i singoli cammini, trovati devono rispettare:

- il primo nodo del cammino deve essere il deposito;

[1] La clusterizzazione è stata fatta utilizzando il software R, i codici riguardanti questa parte dell'algoritmo sono riportati in Appendice. Il metodo utilizzato e la sua implementazione sono descritti nel paragrafo 4.7.

- l'ultimo nodo del cammino deve essere il deposito;
- ciascun controllore per percorrere tutto il cammino deve impiegare un tempo $\leq T$, dove T è il tempo di servizio.

Per quanto riguarda la descrizione del profitto relativo a ciascun percorso e i vincoli relativi ad esso, questi verranno chiariti in seguito.[2][3]

Dato un grafo rappresentante le rete stradale, viene calcolato un numero molto elevato di soluzioni ammissibili per il problema, le quali, stanti i vincoli sopracitati, hanno le seguenti caratteristiche: sono dei cicli che hanno come nodo di partenza e di arrivo il deposito, e il tempo totale di percorrenza di ciascun ciclo è $< T$.

Tutto ciò è stato fatto tramite un programma da noi implementato al calcolatore, in prima istanza in linguaggio C[4] e successivamente solo per rendere più agevoli alcuni passaggi dell'algoritmo, con il software R, cfr. [20][5]. [6][7]

Una volta ottenute tali soluzioni ammissibili è stato necessario calcolarne i profitti. Avendo descritto nel paragrafo precedente le caratteristiche del problema riguardanti i profitti, è chiaro che il calcolo del profitto di ciscun ciclo non è così semplice: è necessario non solo valutare il valore del profitto nel momento in cui un controllore passa su un determinato arco, ma anche calcolare il tempo trascorso dal passaggio del controllore precedente, se vi è stato. Pertanto per ciascun arco il profitto è pari al valore pieno nel caso in cui non vi sia passato nessun altro controllore prima o nel caso in cui il tempo trascorso dal precedente passaggio sia superiore al valore del turnover relativo all'arco in questione, o alla porzione di profitto relativa al momento in cui si attraversa l'arco rispetto al passaggio precedente. Dunque il profitto effettivo lungo un arco vale:

$$max(\Delta T/turnover * profitto_{max}, profitto_{max})$$

[2]Il problema originario, PWTP, Parking Warden Tour Problem, ha una formulazione molto più complessa caratterizzata da un elevato numero di vincoli. L'obiettivo di tale lavoro però è proprio quello di semplificare notevolmente il problema, dunque per la risoluzione tramite Metodo Monte Carlo si è scelto di considerare solo i vincoli indispensabili.

[3]Il modello di PLI del PWTP è riportato integralmente e in due versioni distinte, caso grafo orientato e caso grafo non orientato, in [1, pag.5 e pag.7]

[4]Per un approfondimento sull'argomento si veda[9]

[5]Il progetto è stato fatto implementando gli algoritmo in linguaggio C, ma nel momento in cui si è scelto di clusterizzare le soluzioni per aumentare l'efficienza, si è scelto di implementare nuovamente i vari algoritmo in linguaggio R in modo da poter utilizzare esclusivamente questo e da non dover passare da uno all'altro allungando inutilmente i tempi

[6]Per l'implementazione abbiamo sfruttato anche parti di codici C già esistenti.

[7]Il codice utilizzato è riportato in Appendice.

dove ΔT è la differenza di tempo tra due passaggi consecutivi per l'arco, *turnover* è il tempo di turnover per l'arco, e il $profitto_{max}$ è il profitto pieno per l'arco.

Vediamo ora l'algoritmo nello specifico.

ALGORITMO del METODO Monte Carlo

1. generazione di un percorso casuale che verifichi i vincoli;
2. iterazione del passaggio 1, fino ad ottenere n percorsi;
3. clusterizzazione degli n percorsi in k insiemi distinti;
4. generazione di soluzioni, ovvero di combinazioni di k percorsi ciascuno appartenente ad un diverso cluster;
5. calcolo del profitto effettivo per ciascuna soluzione, tenendo conto dei turnover;
6. scelta della soluzione con maggior profitto.

4.4 Implementazione

Come già detto, per l'applicazione del Metodo Monte Carlo al nostro problema, sono stati utilizzati i linguaggi di programmazione C ed R. I programmi che sono stati implementati in C e in R non sono stati modificati dal punto di vista della sintassi, ma ovviamente sono state apportate le dovute modifiche dal punto di vista logico; per tale motivo la descrizione dell'implementazione dei programmi può esser considerata generale, si riferisce quindi sia ai programmi in C sia a quelli in R.

4.5 Implementazione del grafo

Per prima cosa è stato necessario procedere con l'implementaizione di un programma che descrivesse il grafo, ovvero il modello della rete stradale. È noto che un grafo possa essere implementato in svariati modi, ciascuno adatto ad un grafo con differenti caratteristiche.
I principali metodi di implementazione sono i seguenti:

- matrice di adiacenza;
- lista di adiacenza;

- struttura forward star.

Il grafo in questione ha caratteristiche particolari poiché rappresenta una rete stradale:

- è planare: non esistono archi che si incrociano;
- è sparso.

Per tale motivo il metodo migliore per descriverlo, anche al fine di non occupare inutilmente un numero elevato di celle di memoria, è quello di utilizzare una struttura forward star.

Caratteristiche di un grafo planare

Un grafo planare è un grafo la cui rappresentazione grafica, a meno di isomorfismi[8], può essere fatta su un piano senza che gli archi si intersechino. Vi sono alcuni criteri e condizioni di planarità di un grafo, in seguito viene riportato un importante risultato di Kazimierz Kuratowski, che prende il nome di Teorema di Kuratowski:
Un grafo è planare se e solo se non contiene alcun sottografo che sia un'espansione di $K_{3,3}$ o un'espansione di K_5 .
I grafi $K_{3,3}$ e K_5 sono stati indicati dallo stesso come i due grafi non planari più ridotti, e sono riportati in figura 4.2, 4.3

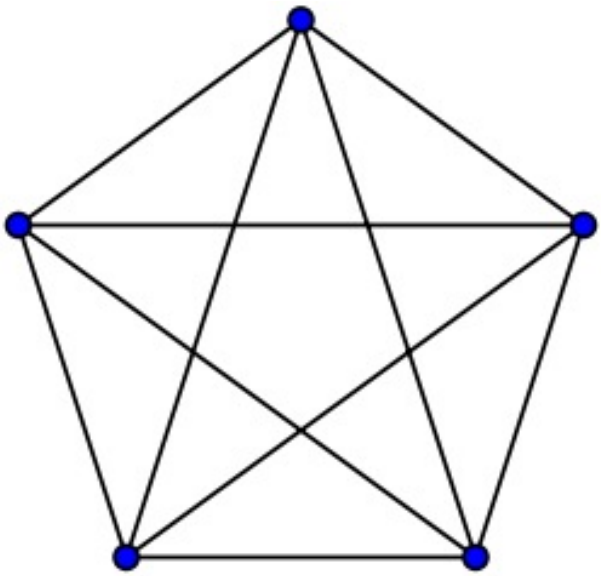

Figura 4.2: Grafo K_5

Inoltre si possono descrivere i seguenti criteri, per grafi semplici, planari, con n nodi ed e archi:

[8]Possono esistere ovviamente rappresentazioni di uno stesso grafo in cui gli archi si intersechino, ma affinchè un grafo sia planare è necessario che esista almeno una rappresentazione grafica in cui gli archi non si intersecano.

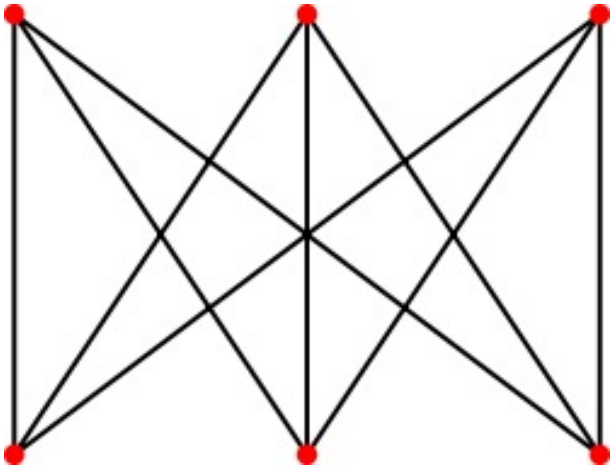

Figura 4.3: Grafo $K_{3,3}$

Teorema 1. Se $n^3 3$, allora $e^2 3n - 6$ Teorema 2. Se $n > 3$ e non vi sono cicli di lunghezza 3, allora $e^2 2n - 4$

Struttura forward star

Questo metodo di implementazione consiste nel memorizzare il grafo come elenco di coppie di nodi, che descrivono gli archi esistenti. È dunque il metodo migliore nel caso di un grafo molto sparso: infatti la matrice e le liste di adiacenza occupano uno spazio di memoria proporzionale al numero di nodi di cui è composto il grafo, a prescindere dal numero di archi presenti. Quindi nel caso di grafo sparso la maggior parte della memoria è occupata da zeri, il che inoltre peggiora l'efficienza del programma inutilmente.
Al contrario memorizzando esclusivamente gli archi, nel caso di grafo sparso, la memoria occupata è notevolmente minore.

In conclusione quindi, la nostra rete stradale è stata memorizzata come sequenza di archi, e per ciascun arco è stato memorizzato il tempo di percorrenza dell'arco e il profitto massimo ad esso relativo. Vediamo come viene memorizzato il seguente grafo:[9] [10]

```
0 1 3 4
1 0 3 4
0 2 3 4
2 0 3 4
1 3 4 5
3 1 4 5
```

[9] I grafi sono stati visualizzati tramite l'applicazione per linguaggio C Graphviz
[10] Esempio di struttura fwd star relativa al grafo in Figura 1

1 2 2 3

2 1 2 3

3 4 2 3

4 3 2 3

4 5 5 0

5 4 5 0

5 7 6 2

7 5 6 2

4 6 7 8

6 4 7 8

6 8 2 3

8 6 2 3

7 8 5 3

8 7 5 3

2 8 8 9

8 2 8 9

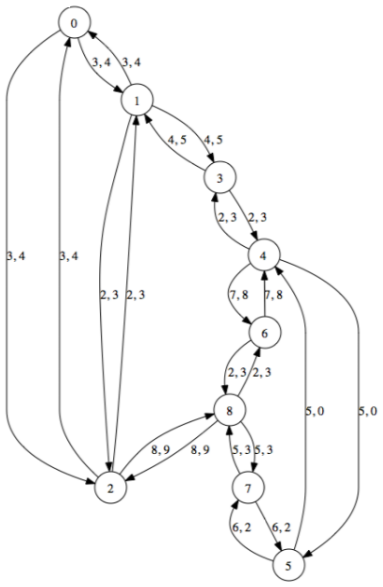

Figura 4.4: Esempio di grafo memorizzato come struttura fwd star

4.5.1 Visualizzazione dei grafi

Soprattutto in un primo momento, in cui è stato necessario testare gli algoritmi e i codici su grafi giocattolo, ovvero su grafi abbastanza piccoli per cui fosse possibile verificare manualmente le so-

luzioni fornite dal calcolatore, è risultato di grande importanza visualizzare i grafi costruiti.
In effetti anche nel confrontarsi con istanze più grandi e con l'istanza reale, la possibilità di visualizzare i grafi fornisce informazioni che sarebbe molto più difficile ottenere dalla sola lettura della matrice che li memorizza.[11]
A tale scopo è stato utilizzato il software Graphviz [12], software che sfrutta il linguaggio C e che prendendo in input i dati relativi al grafo, ne fornisce la rappresentazione grafica; qui si seguito se ne illustreranno brevemente le potenzialità.
Il programma chiede in ingresso un file di testo in cui siano riportate le caratteristiche del grafo: innanzitutto è necessario specificare se si tratta di grafo orientato o non orientato, il che si indica rispettivamente con i termini graph o digraph; si indica poi la dimensione della finestra in cui vogliamo visualizzare il grafo, la forma e il colore dei nodi e in seguito la sequenza di archi di cui è costituito il grafo. Accanto a ciascun arco è possibile inserire un'etichetta. Vediamo qualche esempio:

- esempio di grafo non orientato

```
graph G {
size="8,5"
node [ shape=circle, style=solid, color=black ];
0 -- 1 ;
0 -- 2 ;
1 -- 2 ;
1 -- 3 ;
3 -- 4 ;
```

[11] L'istanza reale è ovviamente visualizzabile come rete stradale della città di Milano, al contrario le istanze casuali sulle quali è stata fatta la campagna sperimentale vengono fornite direttamente come matrici, dunque è necessario poterle visualizzare per capirne la morfologia.
[12] [15]

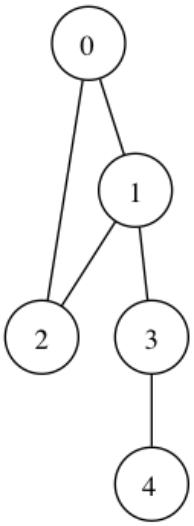

Figura 4.5: Esempio di grafo non orientato

- esempio di grafo orientato

```
digraph G {
size="8,5"
node [ shape=circle, style=solid, color=black ];
0 -> 1 ;
0 -> 2 ;
1 -> 2 ;
1 -> 3 ;
3 -> 4 ;
```

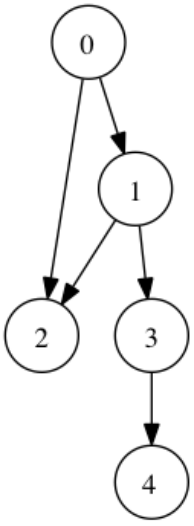

Figura 4.6: Esempio di grafo orientato

- esempio di grafo orientato con etichette

```
digraph G {
size="8,5"
node [ shape=circle, style=solid, color=black ];
0 -> 1 [ label="1" ];
0 -> 2 [ label="5" ];
1 -> 2 [ label="4" ];
1 -> 3 [ label="3" ];
3 -> 4 [ label="5" ];
```

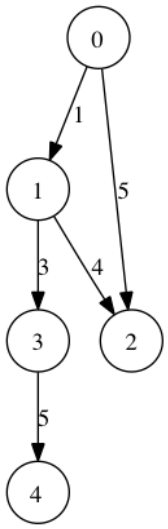

Figura 4.7: Esempio di grafo orientato con etichette

- esempio di grafo orientato con più etichette

```
digraph G {
size="8,5"
node [ shape=circle, style=solid, color=black ];
0 -> 1 [ label="1, 3" ];
0 -> 2 [ label="5, 3" ];
1 -> 2 [ label="4, 2" ];
1 -> 3 [ label="3, 7" ];
3 -> 4 [ label="5, 3" ];
```

Prendendo una delle istanze casuali utilizzate nella campagna sperimentale, è possibile vedere il risultato su un grafo più grande:

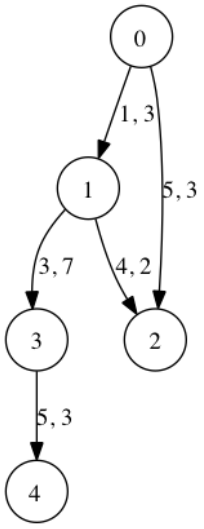

Figura 4.8: Esempio di grafo orientato con più etichette

4.6 Implementazione dell'algoritmo

Una volta implementato il grafo che rappresenta la rete stradale di interesse, è stato necessario implementare un programma che generasse dei cicli ammissibili sul grafo. L'idea seguita per questa parte del lavoro è stata la seguente: generare un alto numero di cicli ammissibili in modo casuale, e solo in un secondo momento calcolarne le combinazioni di $|W|$ percorsi, e quindi i profitti. In quest'ottica, l'unico vincolo da verificare per ciascun ciclo, oltre ovviamente l'obbligo di far partire il ciclo dal nodo corrispondente al deposito, era che il tempo totale del percorso fosse inferiore al tempo di servizio. In effetti, per quanto riguarda il vincolo relativo alla durata del percorso di un determinato controllore rispetto al tempo di servizio, non solo è stato imposto che il tempo totale di percorrenza del ciclo fosse minore, ma è stato imposto anche un limite inferiore. Infatti poiché i percorsi vengono generati in maniera casuale, c'è il rischio che vengano generati anche cicli molto brevi, i quali sicuramente non sarebbero efficienti riguardo alla funzione obiettivo, ma non avrebbero nemmeno molto senso dal punto di vista pratico. Per questo motivo i cicli, per essere considerati ammissibili dall'algoritmo, sono stati presi con un tempo di percorrenza t tale che

$$0.75 * T \leq t \leq T$$

detto T il tempo di servizio.

Implementazione del Metodo Monte Carlo

Per il Metodo Monte Carlo è stato sufficiente imporre di creare una sequenza di archi consecutivi, che partisse e terminasse nel nodo 1, il deposito, e che verificasse il vincolo di tempo. L'unico input da inserire era il grafo sotto forma di matrice, come descritta nel paragrafo precedente, e l'unico

output una matrice che riportava la sequenza degli archi, con relativi profitti e tempi, e una quinta colonna riportante i tempi cumulati, necessaria per il calcolo dei profitti.
Si veda l'implementazione schematicamente:

1. inizializzazione della matrice che conterrà gli archi del ciclo, i relativi tempi e i profitti;
2. generazione random di archi che vanno a generare la matrice relativa al ciclo;
3. chiusura del cammino con imposizione di ritornare al deposito;
4. calcolo dei tempi e dei profitti totali.

Differenze con l'euristca descritta nel terzo capitolo

Rispetto all'euristica descritta nei capitoli iniziali vi è una differenza: non è possibile attendere in un nodo.
Per quanto riguarda l'eliminazione di un possibile tempo di attesa da parte del controllore in un arco, essa è sostenuta dall'idea che il mantenere fermo un controllore in un nodo fino alla scadenza del tempo di turnover di un lato incidente al nodo non apporti effettivamente un aumento del profitto. In effetti se si immagina di avere un solo arco, di profitto p, tempo di percorrenza t e tempo di turnover T, dando la possibilità di restare fermo ad attendere il tempo di turnover il controllore raccoglierà profitto p al tempo 0, al tempo $t+T$, $2t+2T$, $3t+3T$, $\cdots$.
Si otterranno dunque i seguenti risultati:
$P^* = np$, il profitto totale;
$(n-1)*(t+T)$, il tempo totale;
con l'unico vincolo che $(n-1)*(t+T) \leq Turnover$, dove n è il numero di iterazioni del passaggio per l'arco.
Nel caso in cui non si diano la possibilità di attesa, si avrà la raccolta del profitto p al tempo 0, del profitto $p*t/T$ al tempo t, $2t$, $3t$, $\cdots$.
Dunque otterremo i seguenti risultati:
$P = p + p*(t/T)*(m-1)$, il profitto totale;
$(m-1)*t$, il tempo totale;
con l'unico vincolo che $(m-1)*t \leq Turnover$, dove m è il numero di iterazioni del passaggio per l'arco.
Se si immagina di fissare il tempo totale esattamente pari al tempo di turnover e non minore, si può andare ad eguagliare i due tempi in modo da ricavare n in funzione di m, e si trova:

$$m = 1 + (n-1)*t/(t+T)$$

A questo punto si va a verificare in quali casi

$$P^* > P$$

e si trova

$$T/(t+T) > 1$$

che non è mai verificato dal momento che le grandezze sono tutte positive.
È evidente che tale dimostrazione vale nel caso in cui si mantengono esattamente le stesse ipotesi fatte nell'euristica; nel caso si introduca la possibilità per il controllore di avere un tempo di attesa, non necessariamente pari al tempo di turnover dell'arco incidente al suo nodo, le cose potrebbero variare.

4.7 Implementazione della classificazione e del calcolo dei profitti effettivi

Una volta generati n cammini sul grafo, detto $|W| = k$ il numero di controllori avremmo $n!/k!(n-k)!$ soluzioni ammissibili. Come già detto, per evitare di calcolare un numero troppo elevato di combinazioni di cammini si è utilizzato un metodo di classificazione non supervisionata, il metodo delle k-medie.
Questo metodo crea le partizioni dello spazio usando il concetto di vicino più prossimo. Si prendono k punti $y_1, ..., y_k$ posizionati nello spazio dei cammini chiamati prototipi perché ciascuno viene usato come rappresentativo di un cluster.
La partizione j-esima è formata da tutti i punti che sono piu vicini al prototipo y_j che agli altri prototipi: l'obiettivo che l'algoritmo si prepone è quello di minimizzare la varianza totale tra cluster.
Quindi il problema del clustering, stabilito il numero k di prototipi, consiste nel trovare la posizione piu adatta per i k prototipi.
Il concetto di posizione più adatta è relativo ad un criterio di distanza. Nell'algoritmo delle k-medie si usa un criterio che tende a mantenere minima la distanza media tra ogni cammino dato e il prototipo a cui viene associato. L'algoritmo, che adotta un approccio greedy,è piuttosto semplice:

1. si inizializzano i k prototipi in qualche modo, per esempio assegno loro posizioni casuali;

2. si calcola la distanza di ciascun punto del training set da ciascun prototipo;

3. si assegna ciascun punto al prototipo più prossimo;

4. si ricalcolano i prototipi come medie dei rispettivi cluster:

$$y_j = \sum_{l \in \ cluster \ j} x_l$$

5. si itera da 1 finchè i prototipi non smettono di spostarsi.

Il concetto di base del clustering, che accomuna le varie tecniche, è la distanza tra due elementi, infatti la bontà delle analisi ottenute dagli algoritmi di clustering dipende essenzialmente da come essa è stata definita. Dunque nel caso in questione, dati i percorsi ottenuti, sono stati etichettati i percorsi e gli archi di ciascun percorso. L'intento della classificazione è quello di fornire k gruppi di percorsi il più possibile differenti; quindi sono stati inizializzati i k gruppi inserendo un numero casuale di percorsi e iterato l'algoritmo fino ad ottenere una soluzione accettabile, come segue:

1. per ciascun gruppo è stata calcolata la distanza tra i vari percorsi e sono stati iterativamente spostati i percorsi nel gruppo in modo da ottenere distanze inferiori;

2. la **distanza** tra i cammini è data dal numero di archi in comune tra i percorsi: maggiore è il numero di archi in comune, minore è la distanza.

Una volta ottenuti k gruppi di cicli, sono state generate le combinazioni abbinando a ciascuna soluzione un solo percorso per ogni gruppo: sono state così ottenute combinazioni di k percorsi più differenti possibili tra di loro.
Il calcolo del profitto effettivo è stato poi fatto nel modo seguente:

1. si è confrontato un arco di un percorso con tutti gli altri;

2. nel caso in cui un arco fosse ripetuto, si è tenuto in memoria l'istante in cui esso veniva percorso, per ogni volta che veniva percorso, sia in un verso sia in quello opposto;

3. sono stati ordinati i tempi in cui veniva attraversato l'arco ripetuto;

4. sono stati calcolati i tempi intercorrenti tra i passaggi consecutivi per l'arco;

5. se il tempo intercorso era superiore al tempo di turnover si utilizzava come profitto il valore massimo, altrimenti la porzione di profitto ottenuta fino a quel momento;

6. si sono sommati tutti i profitti degli archi ripetuti;

7. si sono iterati per tutti gli archi;

8. si è sommato al profitto ottenuto il profitto di tutti gli archi non ripetuti.

4.8 Il metodo Monte Carlo guidato

Nel metodo Monte Carlo semplice le soluzioni vengono scelte in maniera del tutto casuale, ovvero utilizzando una distribuzione di probabilità uniforme all'interno dell'insieme delle soluzioni ammissibili del problema. Questo significa che, dato tale insieme, la distribuzione di probabilità delle soluzioni è una variabile aleatoria uniforme: ogni soluzione ha la stessa probabilità di essere estratta dall'insieme. È possibile invece associare agli elementi di tale insieme, le soluzioni ammissibili, una distribuzione di probabilità che non sia una uniforme, ma rappresenti meglio il fenomeno in questione.

Nel caso specifico di questo lavoro, si è scelto di affiancare all'euristica basata sul metodo Monte Carlo semplice, un'altra basata sul metodo guidato.

Le assunzioni fatte fin'ora rispetto all'algoritmo sono comunque tutte valide, l'unica cosa da tenere in considerazione è il modo in cui vengono generate le soluzioni casuali.

Il Monte Carlo semplice si concretizza nel modo seguente: dato un nodo, il nodo successivo del percorso viene scelto tra i nodi adiacenti ad esso in modo casuale, attribuendo a ciascuno la stessa probabilità. Nel guidato invece si è scelto di associare a ciascuno dei nodi adiacenti un valore di probabilità dipendente dal profitto relativo all'arco incidente tra il nodo di partenza e il nodo di arrivo. Se prendiamo il grafo in figura 4.10, nel caso del Monte Carlo semplice partendo dal nodo 1 la probabilità di andare in ciascun nodo adiacente sarebbe:

$$P(2) = 1/3$$

$$P(3) = 1/3$$

$$P(4) = 1/3$$

Nel guidato invece:

$$P(2) = 8/35$$

$$P(3) = 15/35$$

$$P(4) = 12/35$$

Fatta questa precisazione, si può dire che l'algoritmo agisce allo stesso modo del metodo semplice; nei capitoli successivi sarà riportata la campagna sperimentale con i relativi risultati di entrambi i metodi.

4.9 I parametri dell'implementazione

Una volta implementato il metodo, è stato necessario imporre alcuni parametri per applicarlo, in particolare il numero di simulazioni sufficienti affinchè il metodo fornisse un valore di profitto sufficientemente vicino all'ottimo. Per fare questa analisi è stata presa un'istanza casuale di media grandezza, 50 nodi, e abbastanza densa come numero di archi, 90%. Sono stati poi confrontati i profitti ottenuti all'aumentare del numero delle simulazioni, ripetendo l'esperimento man mano con un numero più alto di simulazioni, ma utilizzando la stessa generazione casuale. In generale si è osservato che entro la centesima simulazione il risultato si appiattiva e non migliorava più.
Le istanze utilizzate per la campagna sperimentale però arrivavano ad avere anche 300 nodi ed è stata testata anche un'istanza di tali dimensioni. Si è potuto notare che la convergenza della curva del profitto in questo caso avveniva molto più lentamente, in generale entro la duecentesima simulazione, in pochi casi dopo. Poiché però lo scopo di un'euristica è ottenere un buon risultato in tempi relativamente brevi, si è scelto di generare per tutte le istanze 200 simulazioni, in quanto il miglioramento apportato dall'ulteriore aumento il numero di simulazioni non era abbastanza elevato da giustificare un notevole aumento dei tempi di calcolo. Per quanto riguarda le euristiche tale valore è l'unico da dover stimare, mentre per quanto riguarda gli Algoritmi Genetici si vedrà in seguito che vi è un numero molto più elevato di parametri da stimare.

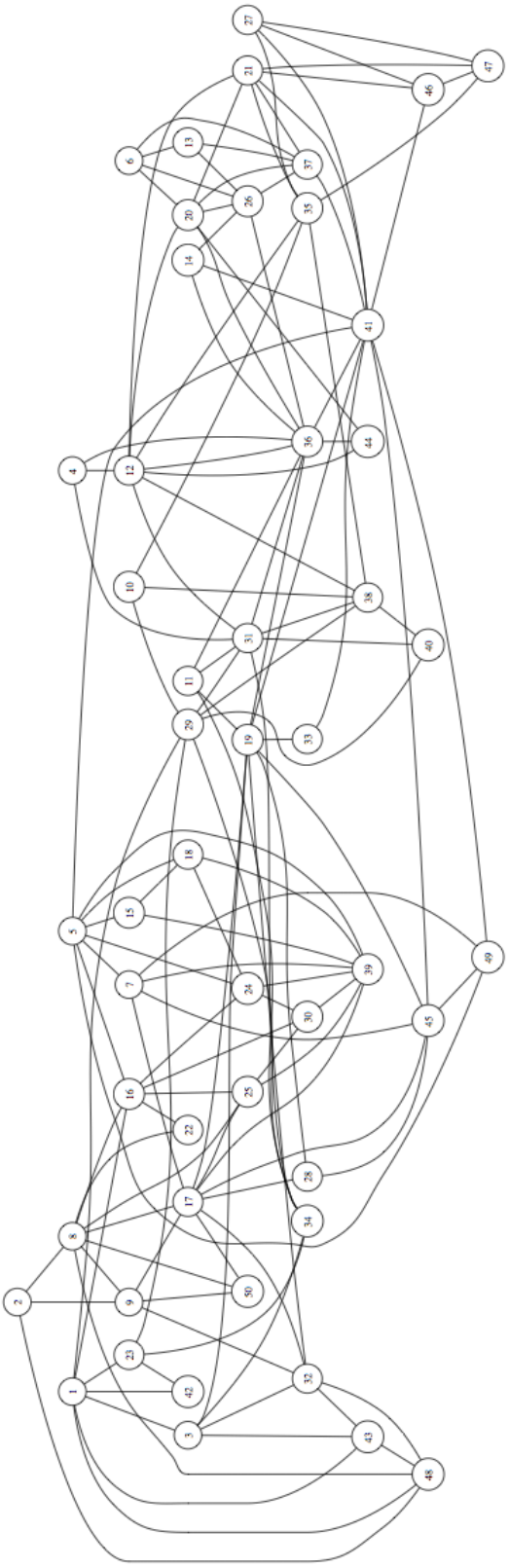

Figura 4.9: Rappresentazione di un'istanza casuale come grafo non orientato

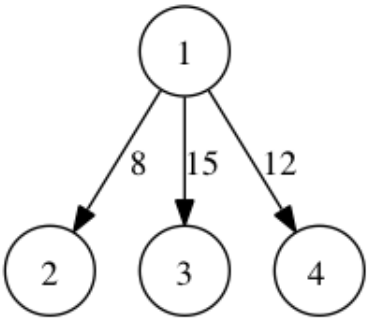

Figura 4.10: Esempio di grafo con archi pesati rappresentati il profitto relativo a ciascun arco

N nodi	Densità percentuale	Simulazioni (n)
50	30	200
50	60	200
50	90	200
100	30	200
100	60	200
100	90	200
200	30	200
200	60	200
200	90	200
300	30	200
300	60	200
300	90	200

Figura 4.11: Parametri delle istanze casuali.

Capitolo 5

Gli algoritmi genetici

5.1 Descrizione dei principi fondamentali

Tra la fine degli anni '50 e l'inizio degli anni '60 del '900 si iniziò a valutare la possibilità che i meccanismi evolutivi potessero portare a nuovi modelli nel campo degli algoritmi di ottimizzazione, in particolare al fine di trovare buone soluzioni a problemi in cui vi fosse un enorme numero di alternative. L'idea era quella di immaginare gli organismi viventi come risolutori di problemi, in grado di sopravvivere nel proprio ambiente sviluppando comportamenti e abilità che tramandano alle generazioni successive tramite il codice genetico.

Gli algoritmi genetici sono stati sviluppati basandosi sulle teorie evoluzionistiche di Charles Darwin, presentate nel suo libro *On the Origin of Species by Means of Natural Selection* del 1859, e sono stati trattati per la prima volta da John Holland nel 1975[1].

La teoria dell'evoluzione di Darwin spiega come gli individui possano sopravvivere e svilupparsi adattandosi all'ambiente. Lo scienziato, nei suoi studi, focalizza l'attenzione sul fatto che solo gli esseri viventi particolarmente idonei hanno possibilità di riprodursi, facendo così passare parte del proprio patrimonio genetico ai figli. Dunque gli elementi più adatti all'ambiente hanno non solo maggior probabilità di sopravvivere, ma anche maggior probabilità di trasmettere il proprio patrimonio genetico. Tra una generazione e l'altra esistono anche delle differenze che personalizzano ogni individuo facendo in modo che i figli non siano copie dei padri, e questi cambiamenti possono portare a una maggiore o minore idoneità dei nuovi individui rispetto ai vecchi. Il ciclo naturalmente si ripete creando generazioni sempre più idonee. Diversamente, si andrebbe dritti verso l'estinzione di una generazione non in grado di adeguarsi. Questo meccanismo prende il nome di selezione naturale e può funzionare solo in popolazioni numerose e prolifiche.

[1] John Holland è nato nel 1929 in Indiana, ha studiato Fisica alla Massachusetts Institute of Technology ed è ora Professore di Psicologia, Ingegneria elettrica e Computer Science all'Università del Michigan.

Gli Algoritmi Genetici (AG), proposti da John Holland partendo da questa teoria, simulano, attraverso processi computazionali, la selezione naturale: si definisce di fatto una popolazione di individui che evolvono di generazione in generazione attraverso meccanismi simili alla riproduzione sessuale e alla mutazione dei geni. In tal modo, si avrà una ricerca euristica che privilegia le zone dello spazio di ricerca dove è maggiormente possibile trovare soluzioni migliori, non trascurando altre zone a più bassa probabilità di successo in cui saranno impiegate comunque un numero minore di risorse.

L'idea di base è quella di codificare la soluzione di un problema sotto forma di una stringa di simboli detta genoma, usando una popolazione di dimensione costante in cui ogni individuo che rappresenta una possibile soluzione è codificato da una stringa differente.

5.1.1 Evoluzione naturale

L'evoluzione naturale, studiata e descritta da Darwin, agisce sul materiale genetico, ossia il genotipo di un individuo, e non sul fenotipo, cioè le caratteristiche fisiche: le caratteristiche che permettono alla generazione successiva di meglio adattarsi all'ambiente di riferimento saranno ereditate tramite il patrimonio genetico dei genitori, e non attraverso ciò che essi possono aver appreso nella propria vita.

- La selezione naturale è il meccanismo che favorisce la riproduzione degli individui che migliorano l'adattabilità all'ambiente mutevole ed elimina gli individui dalla minore potenzialità riprodutiva. Di fatto la selezione promuove le particolari combinazioni genetiche che danno vita ad un organismo più efficiente, selezionando il genotipo.

- La riproduzione è il fulcro del processo evolutivo, infatti la variabilità generazionale è prodotta dalla ricombinazione genica e dalle piccole e molto rare mutazioni casuali del codice genetico. Tale meccanismo rende l'evoluzione molto rapida in quanto accresce enormemente la variabilità. Inoltre esso opera contemporaneamente sull'intera popolazione.

Le teorie di Darwin non solo spiegano l'evoluzione delle singole specie nel proprio ambiente, ma sono alla base delle teorie di evoluzione delle intere popolazioni.[2]

Definizioni di base

- I vettori che portano tutta l'informazione relativa ad un individuo sono i cromosomi: filamenti di DNA composti da geni che, attraverso la codifica di una particolare proteina, determinano le caratteristiche specifiche dell'organismo;

[2]Con popolazione intendiamo un gruppo di individui della stessa specie che operano e si incrociano in uno stesso luogo.

- le posizioni dei geni all'interno del cromosoma sono dette locus;
- le diverse configurazioni delle proteine sono dette alleli;
- l'insieme dei cromosomi da cui è formato un organismo è detto genoma;
- il genotipo è l'insieme dei geni del genoma;
- il fenotipo è il risultato finale, ovvero l'individuo;
- la riproduzione sessuale è il meccanismo di ricombinazione del patrimonio genetico dei genitori, al fine di costituirne uno nuovo nella generazione successiva;
- l'idoneità di un individuo, che in natura si traduce con la maggior probabilità che lo stesso si riproduca, è detta fitness;
- la selezione naturale è il meccanismo attraverso cui vengono promossi come genitori gli individui con fenotipi più adatti, codificati attraverso gli specifici genotipi.

5.1.2 Evoluzione artificiale

Tutto ciò che è stato descritto sull'evoluzione, relativamente al mondo naturale, ha un corrispettivo nel mondo degli algoritmi genetici. Innanzitutto la popolazione è costituita da un certo numero di possibili soluzioni del problema a cui si sta applicando l'algoritmo; ciascuna soluzione viene codificata attraverso un cromosoma: una stringa di valori, 0 e 1 se stiamo utilizzando una codifica binaria, che rappresenta i geni. Il meccanismo che in natura è stato definito come riproduzione sessuale, ossia quello atto al rimescolamento dei geni, negli algoritmi è attuato attraverso un meccanismo detto anch'esso di riproduzione, a cui si associa l'incrocio di soluzioni candidate a questo, ed è detto crossover. Il fenotipo è il significato del cromosoma, cioè la decodifica della soluzione candidata del problema. A differenza che nella maggior parte degli organismi viventi, negli algoritmi si considerano individui con un singolo cromosoma, per cui individuo, cromosoma e genotipo si equivalgono.

Come è stato già spiegato per il mondo naturale, gli individui vengono selezionati per la riproduzione in base al valore della loro funzione di fitness. Negli algoritmi tale funzione, che valuta l'idoneità dell'individuo, è calcolata a partire dalla bontà della soluzione che si sta considerando. Ad esempio, se si si sta applicando l'algoritmo ad un problema di ottimizzazione, la funzione dipenderà dal valore della soluzione in termini di funzione obiettivo.

5.1.3 Il teorema degli schemi

Riguardo al funzionamento degli algoritmi genetici è stata scritta ad oggi pochissima teoria. In particolare si può citare il Teorema Fondamentale degli Algoritmi Genetici o Teorema degli schemi

di Holland, in cui viene assicurato che, sotto determinate ipotesi, gli individui con alti valori di fitness tendono a crescere esponenzialmente nella popolazione attraverso il meccanismo dell'incrocio, assicurando così la convergenza dell'algoritmo genetico verso una soluzione ottimale; esso dimostra che uno schema[3] prolifera più rapidamente se, oltre ad avere un alto valore di fitness, contiene un piccolo numero di geni specifici non lontani l'uno dall'altro. Questo teorema, dovuto ad Holland, spiega la potenza di un algoritmo genetico in termini di quantità di schemi processati.

5.1.4 L'ipotesi dei Building Block

La particolarità degli algoritmi genetici sta nel non esser descritti tramite uno schema fisso, ma dover essere definiti in modo specifico per ogni problema. Per tale motivo è fondamentale la corretta codifica finalizzata a favorire la convergenza dell'algoritmo. La potenza degli algoritmi genetici sta nella capacità di trovare buoni blocchi di costruzione: schemi di lunghezza definita corta, formata da bit che lavorano bene insieme e tendono a portare miglioramenti quando sono incorporati nello stesso individuo. Uno schema di codifica ben riuscita è uno schema che incoraggia la formazione di building block, assicurandosi che:

- i geni correlati siano vicini all'interno del cromosoma;
- ci sia poca interazione tra i geni.

In generale in natura queste due proprietà non sono verificate, in particolare la seconda.

5.1.5 Epistasis

Con il termine epistasi si indica il fatto che l'influenza di un gene sulla fitness di un individuo dipende da quali valori dei geni sono presenti altrove. Un gene è detto epistatico se dalla sua presenza dipende l'effetto di un altro gene in un altro locus. Quindi in generale con il termine epistasi ci si riferisce ad una forte interazione tra geni, interazione che può inibire l'effetto di alcuni di essi.

L'epistasi può essere affrontato da due differenti punti di vista: un problema di codifica o un problema teorico. Nel primo caso basterà utilizzare un differente tipo di codifica e un metodo di decodifica che non porti ad avere epistasi. Se si è invece di fronte al secondo caso, con l'impossibilità di ricodificare il problema per eliminare la dipendenza tra i geni, si dovrà utilizzare un differente approccio. La teoria però fornisce un risultato in tal senso: Vose e Liepins hanno mostrato che

[3]Definiamo schema un cromosoma generalizzato, di lunghezza predefinita, in cui siano specificati i valori assunti da alcune particolari posizioni di bit nella stringa, e siano invece lasciati indefiniti i valori associati alle altre posizioni. Definiamo inoltre istanza di uno schema un cromosoma contenente un dato schema; all'interno di esso, le posizioni fissate sono le posizioni del cromosoma con valore binario specificato nello schema.

in principio alcuni problemi possono essere codificati in maniera che si possano trattare come il semplice problema di contare gli 1. Analogamente alcune codifiche possono essere fatte semplicemente usando crossover e mutazione appropriatamente progettati. In questo modo è sempre possibile rappresentare alcuni problemi con epistasi assente o piccola. In ogni caso va detto che per problemi difficili lo sforzo coinvolto nell'inventare questa codifica è considerevole e costituisce effettivamente la soluzione del problema iniziale. La teoria tradizionale degli algoritmi genetici basata sul teorema degli schemi si basa sull'ipotesi che l'epistasi sia bassa, per cui quando ci si trova davanti a geni con alta epistasi, deve essere sviluppata una nuova teoria e inventati nuovi algoritmi, per agire in queste condizioni.

5.1.6 Caratteristiche generali degli algoritmi genetici

Esistono, oltre al già citato Teorema degli Schemi, due teoremi che danno indicazioni sull'efficienza con cui gli AG elaborano l'informazione contenuta nei cromosomi:

- il Teorema Fondamentale degli Algoritmi Genetici, il quale enuncia che gli individui che contengono un'informazione utile al sistema hanno un numero di discendenti che, se rapportato a quello di un individuo medio, è esponenzialmente crescente con il numero delle generazioni;
- il Teorema del Parallelismo Implicito, il quale afferma che il numero di building blocks elaborati dall'algoritmo genetico ad ogni iterazione è maggiore del numero di individui della popolazione.

È possibile riassumere i punti più interessanti degli AG: innanzitutto l'ultimo teorema mostra di fatto che, data una popolazione, la quantità di informazioni che essa fornisce è maggiore degli individui presenti nella popolazione stessa; inoltre un aspetto interessante è che data la generalità con cui sono definiti, gli AG possono essere applicati ad una vasta tipologia di problemi.

5.1.7 Gli algoritmi evolutivi

Gli algoritmi genetici costituiscono un sottoinsieme degli algoritmi evolutivi, termine generico che indica una gamma di sistemi di risoluzione dei problemi affini ai processi evolutivi. Essi consistono in strategie euristiche che imitano i processi di evoluzione naturale per risolvere i problemi di ricerca globale e si caratterizzano per la capacità di risolvere problemi anche con scarsa conoscenza del dominio. Tuttavia, la loro diffusione è limitata dalle difficoltà di progettazione dovute all'elevato numero di parametri da impostare e dalla necessità di utilizzare una dimensione di popolazione elevata per ottenere una buona convergenza dell'euristica in problemi di una certa difficoltà. Ne conseguono tempi elevati sia in fase di tuning dei parametri che durante l'esecuzione dell'algoritmo. In genere gli algoritmi utilizzati nelle discipline di Intelligenza Artificiale operano la ricerca di

un massimo o di un minimo globale in uno spazio finito sulla base di vincoli sullo spazio delle soluzioni. Fattori come la presenza di più punti di massimo locale, vincoli sul dominio o la non linearità, possono rendere la ricerca molto difficoltosa, per cui il problema non è risolubile in tempi accettabili. Allora si fa uso di algoritmi di tipo euristico che, pur risolvendo il problema con gradi di incertezza e non assicurando la convergenza della ricerca alla soluzione solo in casi particolari, richiedono tempi di convergenza molto minori.
Questa caratteristica porta ad una distinzione tra i metodi:

- metodi forti: sono orientati alla soluzione di un problema specifico, sulla base della conoscenza del dominio particolare e della rappresentazione interna del sistema in esame. Le buone soluzioni ottenute sono difficilmente adattabili ad altri compiti e con risultati non soddisfacenti.

- metodi deboli: utilizzano poca conoscenza del dominio, non sono orientati a un target specifico e risolvono una vasta gamma di problemi.

Gli algoritmi evolutivi sono algoritmi di ricerca euristici, considerati metodi deboli. Tuttavia è stata recentemente introdotta la nuova tipologia dei metodi deboli evolutivi, i quali hanno inizialmente poca conoscenza del dominio, ma che durante la loro evoluzione acquistano maggiore consapevolezza del problema, implementando alcune caratteristiche dei metodi forti.

5.1.8 Rappresentazione del problema

Prima che un algoritmo genetico possa essere implementato, risulta necessario effettuare un'adeguata codifica del problema, anche perché, come già suggerito nei paragrafi precedenti, la teoria degli AG non fornisce precise indicazioni sull'applicazione e sull'implementazione in modo univoco: in base al problema che si ha di fronte è necessario fare delle opportune scelte implementative.
Le operazioni eseguite sui cromosomi usate per creare nuove generazioni sono le seguenti:

1. codifica degli individui;

2. funzione di fitness, o funzione di valutazione;

3. riproduzione;

4. crossover, incrocio o ricombinazione;

5. mutazione;

Si illustra ora come si susseguono gli operatori all'interno dell'algoritmo: $t := 0$
Sia $P(0)$ la popolazione iniziale di m individui:

1. si genera $P'(t)$ da $P(t)$ applicando l'operatore di riproduzione a $P(t)$;
2. si genera $P''(t)$ da $P'(t)$ applicando l'operatore di crossover a $P'(t)$;
3. si genera $P'''(t)$ da $P''(t)$ applicando l'operatore mutazione a $P''(t)$;
4. $P(t+1)$=$P'''(t)$;
5. $t := t+1$

Si vede ora nello specifico quale significato hanno gli operatori appena elencati.

1. Come già detto, l'algoritmo genetico si applica ad una popolazione di soluzioni preesistente, ma per poter generare le popolazioni figlie è necessario codificare ogni soluzione della popolazione di partenza in modo da essere univocamente riconoscibile da parte dell'algoritmo. In generale ci si riferisce al cromosoma artificiale, ovvero alla sequenza di simboli che rappresenta un cromosoma con il nome stringa genetica. Il metodo più comune e più efficace per codificare le soluzioni è utilizzare la codifica binaria: il cromosoma di ciascun individuo della popolazione è una stringa di lunghezza finita composta di simboli binari. Ma vi sono molte altre possibilità di codifica; è di fondamentale importanza scegliere un tipo di codifica che sia adeguato al problema che si sta risolvendo.
2. La funzione di fitness ha lo scopo di giudicare la bontà di ciascun individuo in base al valore della soluzione che quell'individuo è in grado di fornire. Tale valore serve di fatto ad assegnare a ciascun individuo una certa probabilità di riproduzione: maggiore è il valore della fitness più è probabile che l'individuo venga scelto per la riproduzione.
3. La riproduzione crea una nuova $P'(t)$ con un numero di copie degli individui della vecchia $P(t)$ proporzionalmente crescente nella fitness.
4. Il crossover, o crossing over, è in natura il meccanismo di ricombinazione del materiale genetico proveniente dai due genitori. Dal punto di vista dell'algoritmo, una volta scelti gli individui atti alla riproduzione della popolazione si attua il crossover, ovvero l'incrocio tra coppie di individui scelti, come appena descritto, in base al valore della funzione di fitness. L'incrocio può avvenire in diversi modi:
    - il modo più semplice è quello chiamato single point crossing over, nel quale per ciascuna coppia viene scelto un punto intorno al quale fa avvenire lo scambio, come illustrato in figura5.1.
    - allo stesso modo è possibile definire il two point cross over, in cui vengono fatti due tagli nel cromosoma da incrociare, come illustrato in figura5.2.

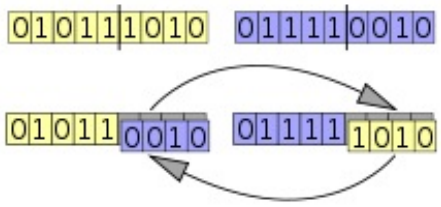

Figura 5.1: Esempio di single point crossover

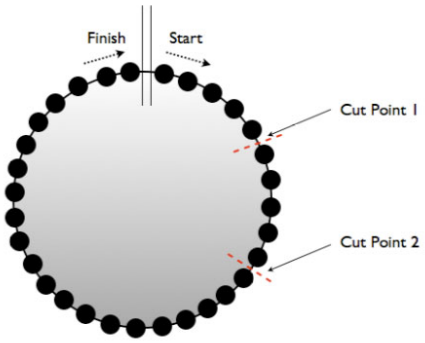

Figura 5.2: Esempio di two point crossover

- Esiste poi un metodo di crossover detto uniforme: esso consiste nella definizione di una maschera che viene applicata ad un genitore, maschera creata in modo casuale per ciascuna coppia di genitori. Come si può vedere in figura, vengono presi dal genitore a cui viene applicata la maschera tutti i geni.

Figura 5.3: Esempio di crossover uniforme

In generale esistono svariati metodi per fare il crossover, ciascuno più adatto al problema che dev'essere trattato. Per questo motivo non esiste un metodo più corretto per fare il crossover, ad esempio, nel caso in cui ci si trovi ad affrontare un problema codificato con stringhe molto piccole, indubbiamente è sconsigliato il two-point, in quanto potrebbe andare a rompere dei building block; in generale l'uniforme è considerato il metodo più robusto, ma è possibile ideare uno specifico metodo che risulti più adatto caso per caso.

5. Come in natura, può accadere che le stringhe della popolazione siano soggette a mutazione: gli elementi del cromosoma possono cambiare il proprio valore in base ad una certa probabilità, ad esempio se si sta utilizzando una codifica binaria, ciascun elemento con una certa probabilità potrà passare dal valore 1 allo 0 e viceversa. Se, al contrario, la codifica si basa sull'utilizzo di numeri reali, ciascun elemento può assumere un nuovo valore estratto da un insieme di riferimento.

5.1.9 Mutazione

Come in natura, anche per gli algoritmi genetici il crossover è il metodo più importante di rimescolamento tra gli individui, quindi avverrà con una frequenza molto maggiore rispetto alla mutazione. In particolare, mentre il crossover avviene ad ogni generazione, la probabilità che un individuo atto alla riproduzione subisca una mutazione viene dalla letteratura indicata pari circa a 0,001. La mutazione viene considerata un operatore minore, in quanto generalmente si ritiene che sia il crossover la principale forza che guida la ricerca dello spazio del problema.
Sia con riferimento al crossover che alle mutazioni, esse operano su blocchi di codice genetico detti Building Block. Alla base dei metodi genetici sta infatti quella che viene definita Building Block Hypothesis: l'idea è che un algoritmo genetico possa raggiungere buone performance attraverso la giustapposizione di schemi corti, di basso ordine e di elevata performance, detti appunto building block. Quindi oltre alla definizione degli operatori, è fondamentale andare a definire per ogni specifico problema i building block.
Un AG genetico può essere riassunto dunque attraverso i seguenti passi[4]:

1. codifica del problema in stringhe;
2. inizializzazione casuale di una popolazione di M cromosomi;
3. valutazione della fitness di ogni individuo della popolazione;
4. selezione di una coppia di cromosomi che avranno funzione di genitori;
5. incrocio della coppia in un punto scelto a caso per generare due discendenti, e se il crossover non avviene i due figli sono la copia identica dei genitori;
6. eventuale mutazione di geni dei figli discendenti;
7. ripetizione dei passi 4 5 e 6 fino a creare M discendenti;
8. la nuova popolazione sostituisce la vecchia;

[4]Rennard e Mitchell 1998

9. ripetizione dei passi a partire dal punto 3.

5.2 Implementazione dell'algoritmo

Per l'implementazione dell'AG è stato necessario valutare come costruire i vari operatori che lo caratterizzano, per poi valutare i parametri necessari; a tale scopo sono state fatte delle analisi preliminari su un'istanza giocattolo[5], in modo da poter in seguito applicare l'AG con i parametri risultati più appropriati, a tutte le istanze casuali e infine all'istanza reale.
L'algoritmo genetico viene applicato a seguito della generazione di un insieme di soluzioni ammissibili, fatta attraverso il codice utilizzato per l'implementazione del metodo Monte Carlo. Dunque l'insieme di partenza a cui viene applicato è costituito da un certo numero m di soluzioni casuali[6], ciascuna delle quali costituita da k percorsi, relative ai k controllori. È importante comprendere la struttura delle soluzioni, in relazione alla teoria degli algoritmi genetici: in particolare, si identifica un individuo come soluzione e quindi ciascun percorso come un cromosoma.

Codifica

Dato l'insieme di tutti i percorsi da cui sono state generate le varie soluzioni, esso è distinto in k parti relative a ciascun controllore, per cui ciascuna soluzione può essere vista come costituita da un elemento per ciascuno dei k sottoinsiemi. Dal momento che le soluzioni hanno questa forma, si potrebbe utilizzare la codifica binaria: ogni soluzione può essere definita con una stringa di valori della lunghezza pari al numero di percorsi totali generati, dove per ciascun sottoinsieme si segna il valore 1 in corrispondenza al percorso presente nella soluzione; ogni soluzione sarebbe dunque caratterizzata da una stringa di molte celle di valore 0 e k celle di valore 1 . Tale metodo è sicuramente corretto, ma per il tipo di problema risulta poco efficiente: in effetti, dato che il numero di percorsi complessivo è sicuramente molto più elevato di k, verrebbero occupate numerose celle di memoria inutilmente.[7] È stato dunque utilizzato un altro metodo: sappiamo che ogni soluzione finale è composta da un percorso per ciascuno dei k cluster, sottoinsiemi dell'insieme dei percorsi; per ogni sottoinsieme i percorsi presenti sono stati messi in colonna, ottenendo in questo modo k matrici di dimensioni $r * m$, dove m è il numero di percorsi presenti nel cluster, e r è la lunghezza standard che è stata data inizialmente alla matrice dei percorsi (ogni percorso viene memorizzato in una matrice di lunghezza data, ovviamente maggiore o uguale al numero massimo di archi che formano il percorso, e in cui per ogni arco vengono indicate anche le caratteristiche di tempo di percorrenza e profitto massimo). Questa struttura permette di identificare ciascun percorso in

[5] Analisi riportate nei capitoli successivi.
[6] Il valore m è sato stimato a seguito dell'analisi sul grafo giocattolo.
[7] È lo stesso motivo che ha spinto a scegliere la struttura forward star per la memorizzazione del grafo.

base alla posizione che ha all'interno della matrice.[8] Quindi la codifica è fatta con numeri reali: ogni soluzione è descritta da k numeri, ciascuno dei quali indica la posizione del relativo percorso all'interno della matrice dei percorsi (figura 5.4).

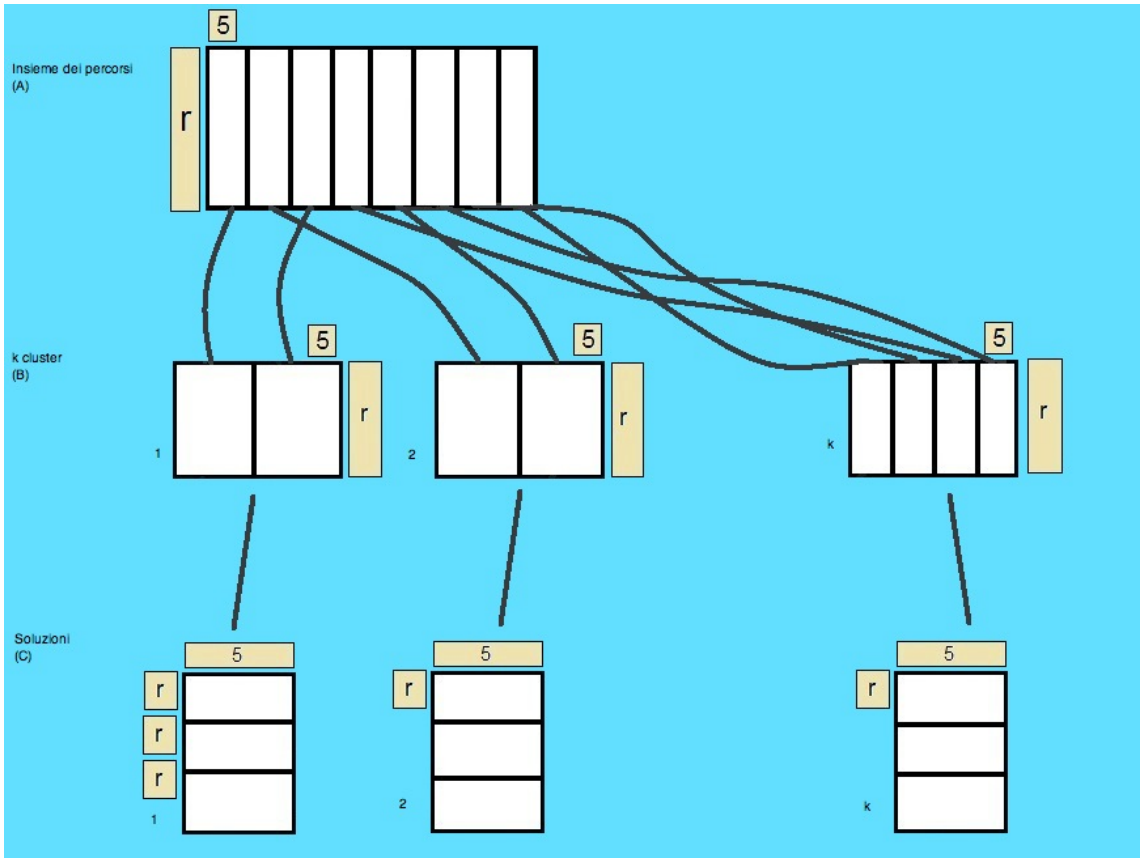

Figura 5.4: Schema della codifica delle soluzioni.

Fitness

La funzione di fitness viene memorizzata per ciascuna delle matrici in un vettore. Per ciascuna soluzione il valore della funzione è dato dal profitto normalizzato rispetto al valore massimo di profitto presente nella matrice, ciò permette di utilizzare il profitto come valore di probabilità che l'individuo sia scelto come genitore.

Riproduzione

La riproduzione è il primo degli operatori che viene applicato alla popolazione di partenza. Una volta generato l'insieme di soluzioni ammissibili costituito da m individui, si applica la riproduzione: la popolazione che si vuole ottenere a seguito di tale operazione ha una determinata dimensione,

[8] È possibile enumerare i percorsi in modo modulare, conoscendo la lunghezza r di ciascuno.

dunque ciò che viene fatto consiste nel prendere le soluzioni più profittevoli, quelle cioè con un valore di fitness maggiore ed inserirle nell'insieme di partenza in quantità proporzionale al valore di fitness: un individuo sarà ripetuto un numero di volte pari alla propria bontà all'interno dell'insieme di partenza.

Crossover

Si è scelto di utilizzare il già descritto single point cross over: nello specifico, ciò significa che, dato un individuo costituito da $1 \dots i \dots k$ percorsi, l'incrocio avviene sempre in un dato punto i. Le soluzioni da incrociare vengono prese in base alla probabilità data dalla funzione di fitness.

Mutazione

Una volta selezionati i genitori, uno dei k percorsi viene tagliato in un punto a caso, si elimina l'arco corrispondente e si costruisce un nuovo percorso tra i due nodi dell'arco che tagliato, che deve verificare solo il tempo di servizio complessivo.

5.2.1 La mutazione e il pacchetto igraph

Per implementare le funzioni relative alle mutazioni è stato utilizzato il pacchetto igraph per il software R[9]. Igraph è un pacchetto gratuito che serve per creare e manipolare grafi, orientati o non orientati, e include implementazioni degli algoritmi più importanti della teoria dei grafi. L'efficienza dell'implementazione di igraph permette di manipolare agevolmente grafi di grandi dimensioni. Tale pacchetto può essere utilizzato con differenti linguaggi di programmazione, come C, Python e R.
Una volta creata l'ultima generazione, per il calcolo del profitto complessivo è stato riutilazzato il codice già presentato nei capitoli precedenti. Nel prossimo capitolo verranno illustrati i risultati e soprattutto i miglioramenti a cui ha portato tale algoritmo.

[9] [25]

Capitolo 6

Analisi degli algoritmi tramite un'istanza giocattolo

Come detto nel capitolo precedente, per tarare l'algoritmo genetico implementato è stato utilizzato un grafo giocattolo costruito *ad hoc*. In particolare è stato costruito a tavolino un grafo abbastanza grande ma di cui fosse semplice calcolare il massimo valore di profitto nel caso in cui si abbiano 3 controllori ($k = 3$).

6.1 Costruzione e descrizione del grafo

Il grafo giocattolo è stato costruito come grafo griglia, ovvero una vera e propria griglia $6 * 6$ in cui ogni punto di incrocio rappresenta un nodo. Dunque esso è costituito da un totale di n=36 nodi e e=60 archi, risultando essere un grafo di fatto molto connesso.[1] Il tempo di servizio è stato fissato a 240 minuti e il tempo di turnover a 90 minuti. Sugli archi sono stati posti i valori di tempo e di profitto: il tempo impiegato per attraversare un arco è stato fissato per tutti pari a 20 minuti, mentre per quanto riguarda il profitto è stato fissato pari a 0 per tutti gli archi eccetto 9, il grafo griglia è rappresentato in figura 6.1. Il profitto di tale soluzione è pari a 900, ed essendo noto è facile fare analisi e confronti tra i metodi e verificare i vari parametri. Così costruito, è molto semplice andare a visualizzare i tre percorsi che forniscono il valore massimo di profitto, riportato in figura 6.3.

[1] Per la precisione, si tratta di un grafo 2-connesso, questo significa che vi sono almeno due cammini possibili tra ciascuna coppia di archi, e quindi che se viene eliminato un nodo in un percorso tra una coppia di archi, questi possono comunque essere ricongiunti.

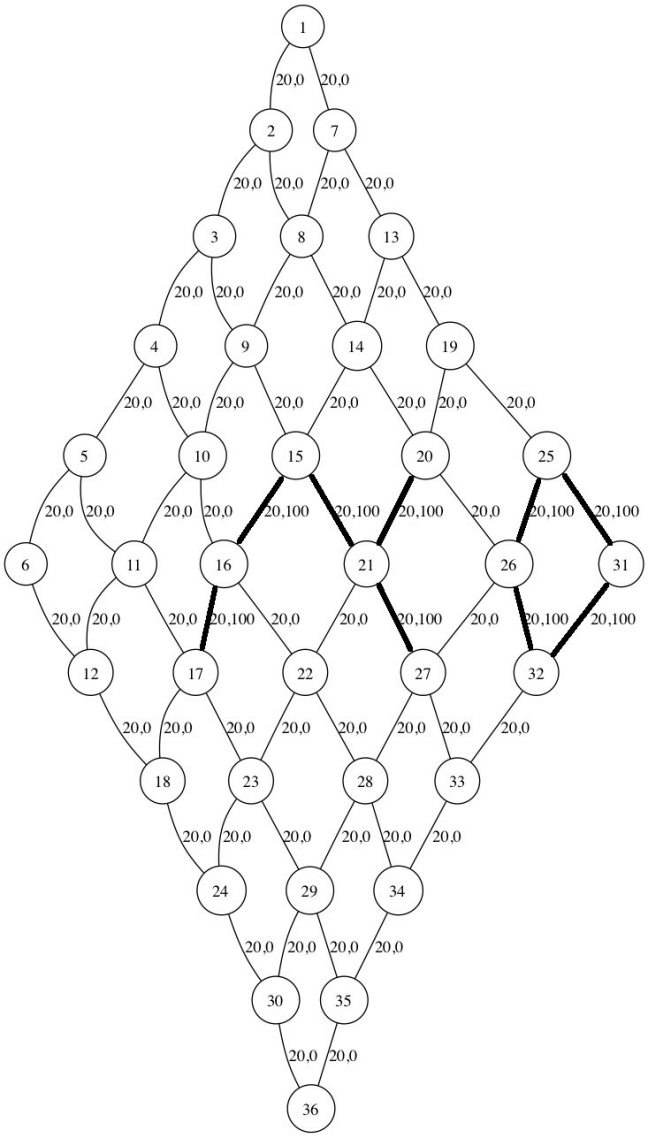

Figura 6.1: Rappresentazione del grafo griglia

6.2 Monte Carlo semplice e guidato sul grafo griglia

A solo scopo informativo e al fine di permettere un confronto tra i vari metodi, su un'istanza nota, è stato calcolato il profitto dei metodi discussi nei capitoli precedenti. Ciò che risulta interessante è vedere che, per come è fatto tale grafo, i due metodi non funzionano particolarmente bene: infatti è necessario generare un numero molto elevato di simulazioni per permettere la convergenza dal momento che i percorsi possibili sono molti.

Inoltre è evidente che, non essendoci molta varietà nei valori dei profitti, il semplice e il guidato si comporteranno quasi allo stesso modo, dato che ad ogni nodo la scelta per il nodo successivo solo in pochissimi casi sarà tra due archi con diversa probabilità. Tale analisi, benchè poco rincuorante, permette di mostrare che i metodi e i parametri utilizzati, non solo devono essere ben valutati in

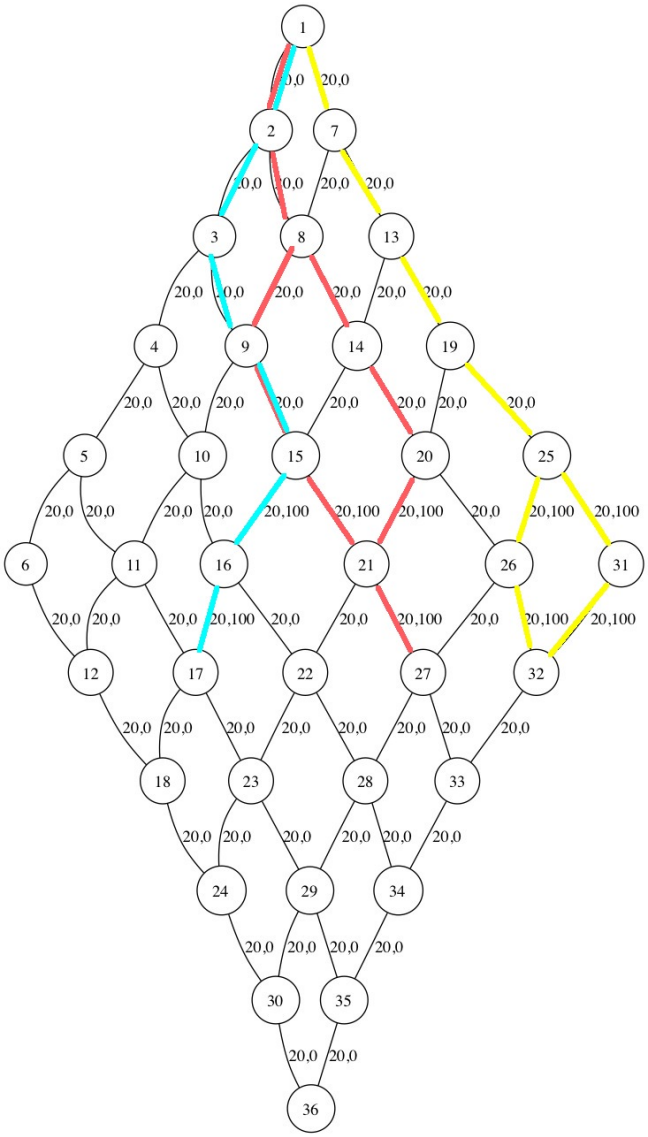

Figura 6.2: Rappresentazione dei percorsi ottimi sul grafo griglia

base al problema che si affronta, ma anche in relazione all'istanza specifica.

6.3 Analisi dei parametri dell'Algoritmo Genetico

Inizialmente, per verificare l'esattezza del metodo è stata costruita una prima popolazione *ad hoc*, che contenesse certamente i percorsi ottimi. Tale analisi non solo è servita a mostrare l'effettivo funzionamento del metodo, ma anche a capire come agire sui parametri per migliorarne l'efficienza. In particolare, come già detto, si è notato che per ottenere dei buoni risultati in tempi adeguati è necessario avere una popolazione di partenza sufficientemente piccola ma già buona in termini di profitti degli individui presenti. Ed infatti una popolazione troppo grande rende molto elevato il numero di possibili incroci, nonostante la presenza del fattore di fitness che guida la scelta degli

individui da far incrociare.

È opportuno ribadire che i parametri da stimare per l'algoritmo sono i seguenti:

- numerosità dell'insieme delle soluzioni a cui applicare il metodo;
- numerosità della popolazione iniziale, ossia quella prodotta dalla riproduzione senza crossover;
- numero di generazioni da attuare;
- probabilità di mutazione.

In ogni caso l'algoritmo con una popolazione iniziale di soli 5 individui, in cui siano presenti i percorsi ottimi, converge al valore massimo alla decima generazione e con una probabilità di mutazione pari a 1/50. Il problema principale dunque, una volta individuata l'effettiva convergenza del metodo, è la stima del numero di soluzioni iniziali su cui applicare la riproduzione. Infatti, se da un lato è necessario che la popolazione a cui vengono applicati gli operatori di crossover e mutazione non sia troppo grande poiché altrimenti le possibili combinazioni sarebbero un numero molto elevato, è pur vero che, dal momento che il numero complessivo di cicli partenti da 1 su questo grafo è molto alto, se non viene generato un numero sufficiente di soluzioni potremmo non avere all'interno della popolazione iniziale non solo i percorsi che devono fornire l'ottimo, ma nemmeno percorsi abbastanza simili da poter garantire che l'avvento della mutazione possa crearli. Sono state fatte prove con i seguenti parametri:

Legenda:
Numerosità dell' insieme di soluzioni = N
Numerosità della popolazione iniziale = No
Numero di generazioni = Ng
Probabilità di mutazione = m

N	1000	2000	10000	
No	10	20	50	100
Ng	100			
m	0.18	0.08		

Figura 6.3: Parametri utilizzati per il grafo giocattolo.

tramite le quali si è deciso di fissare i valori seguenti:

- numerosità della popolazione iniziale=100
- probabilità di mutazione=1/50

Resta da valutare quale debba essere il valore dell'insieme delle soluzioni di partenza. Bisogna premettere che fino a questo punto, avendo garantito a mano la presenza dei percorsi ottimi

all'interno delle soluzioni; ciò ha fatto sì che l'algoritmo non sono convergesse, ma convergesse alla soluzione esatta. In generale invece, ad un metodo euristico o metaeuristico non si richiede che la soluzione fornita sia esatta, bensì che sia buona e che venga raggiunta in tempi brevi. L'analisi necessaria a stimare quest'ultimo parametro non può prescindere da tali considerazioni, pertanto non avrebbe senso creare un insieme di partenza quanto più grande possibile, in quanto questo necessiterebbe di lunghissimi tempi di calcolo; servirebbe invece creare un insieme sufficientemente grande e allo stesso tempo sufficientemente rapido da generare.
I risultati ottenuti dai Metodi Monte Carlo sono stati i seguenti:

- 317 con il metodo Monte Carlo;
- 319 con il metodo Monte Carlo guidato;
- 600 con l'algoritmo Genetico.

Il risultato ottenuto tramite l'algoritmo Genetico è notevolmente più elevato di quelli ottenuti con gli altri metodi per quanto sia una soluzione non molto alta ma, come detto, il problema principale che rende l'algoritmo implementato come sopra descritto è la definizione della generazione di partenza.

6.3.1 Metodo alternativo di generazione della popolazione iniziale

La generazione iniziale del metodo genetico fin qui descritto è stata prodotta generando cammini casuali con lo stesso criterio e gli stessi codici del metodo Monte Carlo. Dati i problemi legati a tale punto dell'algoritmo, si è scelto di abbandonare la strada della generazione casuale di percorsi e di generarli in modo più guidato.
La generazione può essere schematizzata nel seguente modo:

1. per ogni nodo del grafo viene calcolata una quantità U pari alla somma dei profitti degli archi incidenti nel nodo;
2. ogni nodo j viene dunque etichettato con il rispettivo valore U_j;
3. i nodi vengono ordinati in ordine descrescente di U;
4. un certo numero di nodi con il valore più alto di U viene inserito in un insieme E;
5. per ciascun nodo $j \in E$ vengono etichettati i nodi adiacenti con una quantità $U = U_i - \alpha$ e in seguito inseriti in E;
6. il procedimento viene iterato fino ad avere in E tutti i nodi del grafo.

Questo procedimento dovrebbe portare i percorsi sempre in zone con maggior profitto, non in modo locale come per il metodo Monte Carlo guidato, ma in maniera globale. In questo caso devono essere anche calcolati alpha e il numero di nodi appartenenti a E al primo passo.

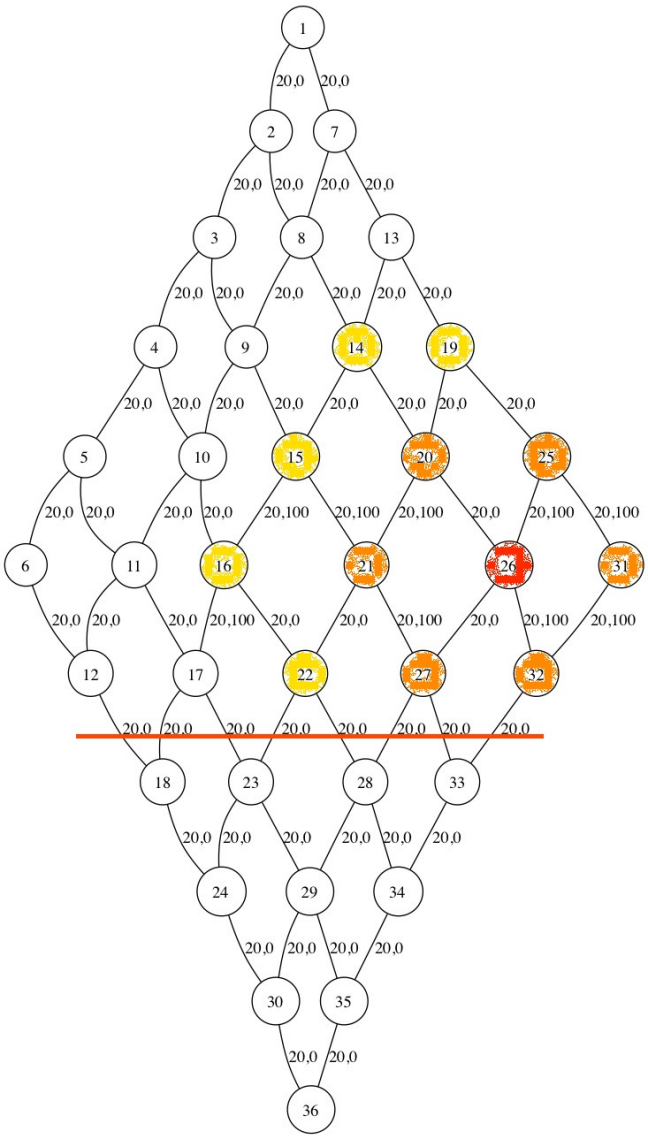

Figura 6.4: Rappresentazione della decrescita dell'intensità di un nodo significativo, il nodo 26, proporzionale all'allontanamento dal nodo stesso. Inoltre la riga nella parte bassa del grafo rappresenta il limite oltre al quale i controllori, a causa del ridotto tempo di servizio, non possono arrivare.

In effetti avendo fatto con tale metodo le stesse prove fatte con quello precedente sul grafo giocattolo, non è stato riscontrato un netto miglioramento, ed è stato quindi utilizzato per le campagne sperimentali e per l'istanza reale il metodo precedente.

6.4 Conclusioni ottenute dall'analisi del grafo giocattolo

Il grafo giocattolo, come già detto in precedenza, non ha avuto esclusivamente il compito di tarare i parametri dell'algoritmo genetico, bensì ha permesso di comprendere le potenzialità e soprattutto le criticità dell'algoritmo. Come accennato in fase di descrizione del problema, la sua caratteristica principale è data dal fatto che l'esistenza del tempo di turnover fa sì che il profitto di ciascun percorso dipenda dagli altri percorsi che compongono la soluzione. Tale caratteristica, nell'ambito degli algoritmi genetici, si traduce in quella che è stata definita epistasi, ovvero la dipendenza tra geni che ne inibisce l'efficienza. Come si è visto nella descrizione di tale caratteristica, essa può essere risolta ricodificando il problema o cambiando la strategia stessa del genetico. Ma dall'analisi appena fatta si evince che il problema principale del metodo è la scarsa efficienza del primo suo passo: la generazione dei percorsi; infatti, se i percorsi che devono fornire l'ottimo non sono già presenti nella generazione iniziale, attraverso il crossover o la mutazione non verranno mai riprodotti. Una possibile soluzione a tale problema potrebbe essere quella di applicare una fase genetica già in fase di generazione dei percorsi, in modo da assicurare la presenza di buoni percorsi già nella generazione di partenza, oppure andare a ridefinire il metodo di crossover che, per come è fatto, non permette la scomposizione del singolo cammino, ma solo la ricombinazione di cammini appartenenti a soluzioni differenti.
Queste potrebbero dunque essere le strade da intraprendere per rendere più efficiente il metodo.

Capitolo 7

Campagna sperimentale

7.1 Istanza reale

Per valutare l'effettivo comportamento dell'algoritmo genetico, esso è stato utilizzato per risolvere il problema sull'istanza reale già utilizzata in [3]. Come già detto in precedenza, non è stata utilizzata tutta l'area cittadina a questo scopo, ma in una zona che abbiamo chiamato *zona di rilievo*.

La zona di rilievo Approfondiamo ora le caratteristiche della zona di rilievo[1].

Il campione, affinchè fosse quanto più significativo per la stima dei parametri del modello, è stato scelto in base ai seguenti criteri:

- attrattività, cioè una zona che, per la densità di funzioni commerciali e sociali, fosse caratterizzata da alta domanda di sosta;
- significatività, valutata in funzione dell'estensione, in termini di lunghezza complessiva, della rete;
- omogeneità, valutata internamente al campione, tenendo conto dell'effetto della presenza di poli attrattori quali, ad esempio, un polo fieristico o una struttura ospedaliera;
- compresenza di sosta regolamentata per residenti e a rotazione.

A seguito di tali considerazioni, infine, la scelta è ricaduta su una parte di territorio delimitata a ovest da Corso Buenos Aires, a sud da piazzale Lavater, a nord da piazzale Bacone e a est da via Eustachi, come riportato in figura 7.2. Tale zona, che da qui in avanti verrà chiamata *zona di rilievo*, rappresenta a sua volta un sottoinsieme di una zona di maggiore estensione che verrà invece presa come scenario per la campagna sperimentale. Ci si riferirà a questa seconda zona come *zona*

[1] Per approfondimenti vd.[3]

di simulazione.

Figura 7.1: Inquadramento della zona di rilievo sul territorio milanese

La zona di rilievo è esterna ad Ecopass per avere a disposizione dati di traffico indipendenti, in prima battuta, dal tipo di autoveicolo in circolazione. Inoltre, dal punto di vista delle funzioni d'uso del territorio, la zona prescelta rappresenta un buon compromesso tra la funzione residenziale, concentrata per lo più tra via Morgagni e corso Buenos Aires e funzione commerciale. Ci si aspetta, infatti, che in prossimità di zone a carattere commerciale, il ricambio di veicoli in sosta sia più frequente, mentre nelle zone a carattere residenziale sia meno frequente.

La regolamentazione della sosta nella zona di rilievo comprende i residenti e i parcheggi a rotazione, con vie totalmente dedicate alla sosta residenziale, vie totalmente dedicate alla sosta a rotazione e vie miste.

Si presenta, infine, priva di singoli punti attrattori, quali, ad esempio, ospedali, centri commerciali, luoghi di ricreazione di massa, e, rispetto all'estensione complessiva della rete stradale milanese rappresenta lo $0,03$ %. Indicativamente, considerando invece solo la rete stradale regolamentata, la zona di rilievo rappresenta circa il 2 %.

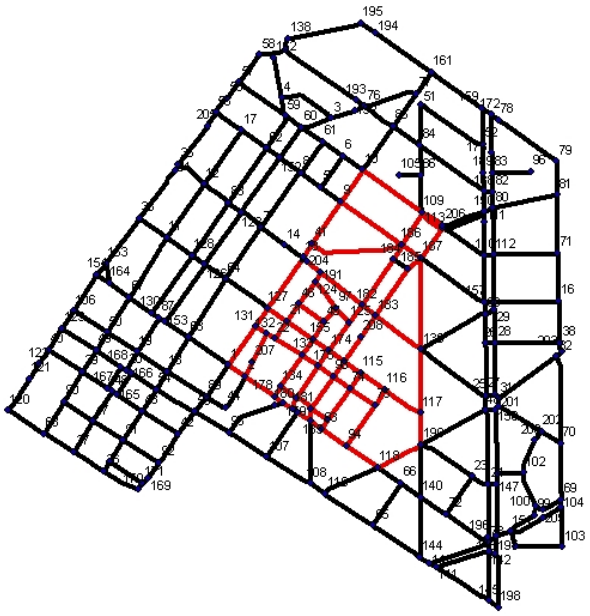

Figura 7.2: Inquadramento della zona simulazione sul territorio

I dati sono stati ricavati dal dott. Lia, le caratteristiche e i particolari al riguardo sono riportati nel paragrafo 4.3 della tesi dello stesso. In questo lavoro ci limitiamo a ricordare che i dati, gli stessi richiamati nell'articolo già citato, sono relativi a due momenti della giornata, mattina e pomeriggio, nella zona descritta.
In [1] sono riportati i seguenti valori di profitto:

- mattino: 4099, con upper bound 5099;
- pomergiggio: 8719, con upper bound 10766.

7.1.1 I metodi Monte Carlo

Utilizzando i metodi Monte Carlo per l'istanza reale, sono stati ottenuti i seguenti risultati:

- mattino: 3358 per semplice, 3542 per il guidato;
- pomeriggio 3458 per il semplice, 3539 per il guidato.

7.1.2 L'algoritmo Genetico

Con l'algoritmo Genetico sono stati raggiunti i seguenti valori di profitto:

- mattino: 4601;
- pomeriggio: 9614.

Risulta dunque evidente che anche per quanto riguarda l'istanza reale, il metodo metaeuristico fornisce risultati migliori rispetto all'euristica basata sul modello esatto di MILP.
Inoltre, come già detto, la caratteristica fondamentale dei metodi euristici e metaeuristici, che li contraddistingue da quelli esatti, è la rapidità nella ricerca della soluzione, riscontrata anche in questo caso.
Si riassumono i risultati ottenuti nella tabella in figura 7.3, in cui sono riportati anche i valori di gap percentuali dei vari metodi rispetto al bound.

	Upper bound	Euristica	Montecarlo	Montecarlo g	Genetico	Gap Euristica	Gap Montecarlo	Gap montecarlo g	Gap genetico
morning	5099	4099	3358	3542	4601	19.6 %	34%	30.52%	10%
afternoon	10766	8719	3458	3539	9614	19.02 %	67.87%	67.14%	11%

Figura 7.3: Risultati della campagna sperimentale sull'istanza reale.

Dunque si evince un netto miglioramento, per quanto riguarda il valore del profitto, dovuto all'utilizzo dell'algoritmo genetico.
Anche per quanto riguarda i tempi di calcolo, l'algoritmo genetico è decisamente più rapido: nell'articolo sono riportati i tempi di calcolo per le due istanze, mattino e pomeriggio, per due differenti metodi esatti, in un caso il tempo di calcolo è nell'ordine di 10^5 secondi, mentre per il secondo è nell'ordire dei 10^4. Con l'algoritmo genetico i tempi sono ridotti a circa un minuto.

7.2 Istanze casuali

I metodi descritti nelle sezioni precedenti, una volta implementati sono stati applicati ad un certo numero di istanze casuali, le stesse sulle quali era stata applicata l'euristica studiata dal dott. Lia nella sua Tesi e richiamate in [1], in modo da poterne confrontare i risultati e verificarne l'efficienza.
Le istanze giocattolo sono 60, generate con le seguenti caratteristiche:

- numero di nodi $i = 1, \cdots, 300$
- densità di archi in relazione al numero di nodi $d = 30\%, 60\%, 90\%$

Inoltre, per ciascuna combinazioni di tali caratteristiche sono state create cinque istanze. In realtà le istanze create sono sessanta, ma l'algoritmo è stato applicato ad un numero inferiore di istanze poiché alcune sono inutilizzabili per diversi motivi che saranno analizzati in seguito.[2]

7.2.1 I metodi Monte Carlo

Tramite l'applicazione del metodo Monte Carlo è stato possibile per ogni istanza calcolare un numero anche molto alto di soluzioni ammissibili, il che - come detto nel capitolo di descrizione del

[2]Tali problemi, relativi alla morfologia dei grafi delle istanze casuali, erano già stati sollevati in [1, pag. 13]

metodo - lo rende più efficiente. Le soluzioni ottenute sulle varie istanze sono buone sia in termini di profitti che in termini di tempo di calcolo della soluzione. Al riguardo è necessario fare alcune precisazioni: il calcolo della singola soluzione casuale e relativo profitto ha un tempo di calcolo decisamente ridotto, ma come già detto il Metodo Monte Carlo è tanto più efficiente quante più soluzioni vengono generate, per questo motivo per ottenere dei buoni valori di profitto per ciascuna istanza sono state create 100 soluzioni ed è stata poi considerata quella con il valore massimo.
La tabella qui di seguito riportata illustra i risultati ottenuti sulle istanze casuali. Per coerenza con i risultati ottenuti con l'euristica di [1] vengono riportate tutte le istanze casuali, nel caso di quelle che hanno arrecato i problemi descritti in precedenza, ciò è riportato esplicitamente.
Nella tabella sono riportate tutte le istanze: nella prima colonna si trova il nome dell'istanza, nella seconda il profitto massimo trovato per l'istanza e nella terza il tempo di calcolo della soluzione. Ciascuna istanza è indicata con una sigla della forma planarXXdYYpZZ, dove XX indica il numero di nodi, YY la densità e ZZ è un numero identificativo dell'istanza. Per comodità verranno analizzate separatamente le istanze con lo stesso numero di nodi.

Monte Carlo

Istanze con 50 nodi

Istanza	Profitto	Upper bound	Gap perc
planar50d30p01	-	300	-
planar50d30p02	2039	4519	54.89
planar50d30p03	1597	2210	27.75
planar50d30p04	2578	4736	45.57
planar50d30p05	-	0	-
planar50d60p01	1777	3998	55.56
planar50d60p02	2362	4950	52.28
planar50d60p03	1693	4350	61.08
planar50d60p04	-	0	-
planar50d60p05	2613	4650	43.81
planar50d90p01	2012	4853	58.55
planar50d90p02	2603	4866	46.50
planar50d90p03	1745	3686	52.66
planar50d90p04	2951	5100	42.14
planar50d90p05	1921	4500	57.32

Figura 7.4: Risultati della campagna sperimentale del metodo Monte Carlo sulle istanze casuali con 50 nodi.

Si è rilevato che alcune delle istanze casuali generate non hanno prodotto alcun risultato, analizzando i grafi ad esse relative è stato notato che:

1. planar50d30p01: il grafo di questa istanza non è connesso. In particolare il nodo 1, rappresentante il deposito, è in comunicazione solo con l'arco numero 4, per cui pur utilizzando il codice che ammette la possibilità di ripassare su un arco, il profitto massimo che si riesce a ottenere è di 100, lo stesso ottenuto con l'euristica;
2. planar50d30p05: il grafo di tale istanza non ha il nodo 1, quindi è impossibile utilizzarla;
3. planar50d60p04: questa istanza ha gli stessi problemi di quella riportata al punto 1;

Istanze con 100 nodi

Istanza	Profitto	Upper bound	Gap perc
planar100d30p01	1981	3150	37,12
planar100d30p02	2466	4275	42,32
planar100d30p03	1890	3830	50,66
planar100d30p04	1443	1650	12,58
planar100d30p05	1916	3817	49,80
planar100d60p01	1630	4200	61,20
planar100d60p02	2482	4468	44,44
planar100d60p03	2894	4050	28,55
planar100d60p04	2560	4650	44,95
planar100d60p05	1504	4075	63,09
planar100d90p01	1920	4757	59,65
planar100d90p02	2331	4080	42,86
planar100d90p03	1940	4500	56,90
planar100d90p04	1308	4318	69,71
planar100d90p05	2042	4134	50,60

Figura 7.5: Risultati della campagna sperimentale del metodo Monte Carlo sulle istanze casuali con 100 nodi.

Istanze con 200 nodi

Istanza	Profitto	Upper bound	Gap perc
planar200d30p01	1988	3711	46.43
planar200d30p02	1991	4091	51.33
planar200d30p03	1936	4049	52.19
planar200d30p04	2016	4474	54.94
planar200d30p05	1934	4500	57.02
planar200d60p01	2142	5327	59.79
planar200d60p02	2119	4950	57.19
planar200d60p03	1819	4950	63.25
planar200d60p04	2037	4500	54.73
planar200d60p05	2135	5158	58.61
planar200d90p01	1900	5042	62.32
planar200d90p02	2281	4950	53.92
planar200d90p03	2108	4870	56.71
planar200d90p04	2031	4609	55.93
planar200d90p05	2408	5157	53.31

Figura 7.6: Risultati della campagna sperimentale del metodo Monte Carlo sulle istanze casuali con 200 nodi.

Istanze con 300 nodi

Istanza	Profitto
planar300d30p01	1988
planar300d30p02	1155
planar300d30p03	2073
planar300d30p04	1951
planar300d30p05	1917
planar300d60p01	2600
planar300d60p02	2827
planar300d60p03	2566
planar300d60p04	1092
planar300d60p05	2344
planar300d90p01	1501
planar300d90p02	1375
planar300d90p03	1444
planar300d90p04	2221
planar300d90p05	1817

Figura 7.7: Risultati della campagna sperimentale del metodo Monte Carlo sulle istanze casuali con 300 nodi.

Per quanto riguarda le istanze casuali con 300 nodi, non sono disponibili i valori degli upper bound, in quanto tali istanze non erano state utilizzate in precedenza.

Monte Carlo guidato

Istanze con 50 nodi

Istanza	Profitto	Upper bound	Gap perc
planar50d30p01	-	300	-
planar50d30p02	2232	4519	50.61
planar50d30p03	1413	2210	36.06
planar50d30p04	2957	4736	37.56
planar50d30p05	-	0	-
planar50d60p01	2200	3998	44.97
planar50d60p02	2636	4950	46.75
planar50d60p03	2099	4350	51.75
planar50d60p04	-	0	-
planar50d60p05	2827	4650	39.20
planar50d90p01	1692	4853	65.13
planar50d90p02	2542	4866	47.76
planar50d90p03	2005	3686	45.60
planar50d90p04	3602	5100	29.37
planar50d90p05	2523	4500	43.93

Figura 7.8: Risultati della campagna sperimentale del metodo Monte Carlo guidato sulle istanze casuali con 50 nodi.

Istanze con 100 nodi

Istanza	Profitto	Upper bound	Gap perc
planar100d30p01	1917	3150	39.14
planar100d30p02	2568	4275	39.93
planar100d30p03	1984	3830	48.20
planar100d30p04	1598	1650	3.15
planar100d30p05	2158	3817	43.46
planar100d60p01	2224	4200	47.05
planar100d60p02	2143	4468	52.04
planar100d60p03	3186	4050	21.33
planar100d60p04	2523	4650	45.74
planar100d60p05	1573	4075	61.40
planar100d90p01	2351	4757	50.58
planar100d90p02	2745	4080	32.72
planar100d90p03	2136	4500	52.53
planar100d90p04	2134	4318	50.58
planar100d90p05	2475	4134	40.13

Figura 7.9: Risultati della campagna sperimentale del metodo Monte Carlo guidato sulle istanze casuali con 100 nodi.

Istanze con 200 nodi

Istanza	Profitto	Upper bound	Gap perc
planar200d30p01	2342	3711	36.89
planar200d30p02	2513	4091	38.57
planar200d30p03	2302	4049	43.15
planar200d30p04	1970	4474	55.97
planar200d30p05	2407	4500	46.51
planar200d60p01	2300	5327	56.82
planar200d60p02	2229	4950	54.97
planar200d60p03	2386	4950	51.80
planar200d60p04	2385	4500	47.00
planar200d60p05	2195	5158	57.44
planar200d90p01	2490	5042	50.61
planar200d90p02	2347	4950	52.59
planar200d90p03	2721	4870	44.13
planar200d90p04	2385	4609	48.25
planar200d90p05	2310	5157	55.21

Figura 7.10: Risultati della campagna sperimentale del metodo Monte Carlo guidato sulle istanze casuali con 200 nodi.

Istanze con 300 nodi

Istanza	Profitto
planar300d30p01	1333
planar300d30p02	1036
planar300d30p03	2134
planar300d30p04	2003
planar300d30p05	2546
planar300d60p01	2660
planar300d60p02	2642
planar300d60p03	2494
planar300d60p04	1347
planar300d60p05	2336
planar300d90p01	1396
planar300d90p02	1448
planar300d90p03	1362
planar300d90p04	2890
planar300d90p05	2081

Figura 7.11: Risultati della campagna sperimentale del metodo Monte Carlo guidato sulle istanze casuali con 300 nodi.

7.2.2 Algoritmo Genetico

Istanze con 50 nodi

Istanza	Profitto	Upper bound	Gap perc	Miglioramento montecarlo
planar50d30p01	-	300	-	47.21
planar50d30p02	3001	4519	33.59	17.66
planar50d30p03	1662	2210	24.80	32.59
planar50d30p04	3418	4736	27.83	-
planar50d30p05	-	0	-	52.14
planar50d60p01	2703	3998	32.39	13.67
planar50d60p02	2685	4950	45.76	10.16
planar50d60p03	1865	4350	57.13	
planar50d60p04	-	0	-	-
planar50d60p05	3248	4650	30.15	24.32
planar50d90p01	2365	4853	51.27	39.80
planar50d90p02	2896	4866	40.48	13.92
planar50d90p03	2177	3686	40.94	24.76
planar50d90p04	2765	5100	45.78	-6.29
planar50d90p05	2666	4500	40.76	38.80

Figura 7.12: Risultati della campagna sperimentale dell'algoritmo Genetico sulle istanze casuali con 50 nodi.

Istanze con 100 nodi

Istanza	Profitto	Upper bound	Gap perc	Miglioramento montecarlo
planar100d30p01	2325	3150	26.19	21.26
planar100d30p02	3080	4275	27.95	24.91
planar100d30p03	2493	3830	34.91	31.92
planar100d30p04	1610	1650	2.42	18.75
planar100d30p05	2693	3817	29.45	40.53
planar100d60p01	2615	4200	37.74	60.45
planar100d60p02	1671	4468	62.60	-22.04
planar100d60p03	1907	4050	52.91	-34.10
planar100d60p04	2949	4650	36.58	16.87
planar100d60p05	1629	4075	60.02	8.30
planar100d90p01	2782	4757	41.52	44.93
planar100d90p02	2919	4080	28.46	25.20
planar100d90p03	2455	4500	45.44	26.57
planar100d90p04	1902	4318	55.95	45.43
planar100d90p05	2508	4134	39.33	22.81

Figura 7.13: Risultati della campagna sperimentale dell'algoritmo Genetico sulle istanze casuali con 100 nodi.

Istanze con 200 nodi

Istanza	Profitto	Upper bound	Gap perc	Miglioramento montecarlo
planar200d30p01	1962	3711	47.13	-1.32
planar200d30p02	3021	4091	26.15	51.74
planar200d30p03	2356	4049	41.81	21.49
planar200d30p04	2235	4474	50.04	13.43
planar200d30p05	2566	4500	42.98	32.67
planar200d60p01	2315	5327	56.54	8.08
planar200d60p02	2494	4950	49.62	17.70
planar200d60p03	3055	4950	38.28	67.91
planar200d60p04	2967	4500	34.07	45.67
planar200d60p05	2415	5158	53.18	13.11
planar200d90p01	2594	5042	48.55	36.51
planar200d90p02	2983	4950	39.74	30.76
planar200d90p03	2215	4870	54.52	5.08
planar200d90p04	2313	4609	49.82	13.85
planar200d90p05	2474	5157	52.03	7.09

Figura 7.14: Risultati della campagna sperimentale dell'algoritmo Genetico sulle istanze casuali con 200 nodi.

Istanze con 300 nodi

Istanza	Profitto	Miglioramento montecarlo
planar300d30p01	2224	36.15
planar300d30p02	2977	187.47
planar300d30p03	2353	13.53
planar300d30p04	2231	14.33
planar300d30p05	2971	54.95
planar300d60p01	2030	-21.92
planar300d60p02	3365	27.36
planar300d60p03	3138	25.81
planar300d60p04	2953	170.19
planar300d60p05	2811	20.36
planar300d90p01	2654	90.12
planar300d90p02	3012	119.05
planar300d90p03	2659	95.24
planar300d90p04	3214	44.71
planar300d90p05	2815	54.96

Figura 7.15: Risultati della campagna sperimentale dell'algoritmo Genetico sulle istanze casuali con 300 nodi.

7.3 Conclusioni sulle campagne sperimentali

Ciò che si evince dalle campagne sperimentali fatte sulle istanze casuali è che **l'algoritmo Genetico** porta, nella maggior parte dei casi, a **risultati migliori** rispetto a quelli ricavati tramite i metodi Monte Carlo.

Inoltre dall'euristica di [1] erano stati ricavati gli upper bound per il problema, nelle tabelle dei paragrafi precedenti sono riportati i gap percentuali rispetto a tali bound.

Apparentemente i risultati non appaiono così elevati rispetto a tali valori, ma si ricordi che i bound sono stati ricavati nel seguente modo: si è calcolato il percorso di profitto massimo sulla rete, ed è stato moltiplicato per il numero di controllori [3].

É abbastanza evidente che tali bound siano piuttosto grossolani, ed è dunque del tutto normale ottenere valori anche notevolmente più bassi.

Il risultato più interessante ottenuto con l'algoritmo Genetico però, non è il valore di profitto, comunque piuttosto alto, bensì il **netto miglioramento in termini di tempi di calcolo**.

Come detto inizialmente il problema maggiore del metodo esatto e dell'euristica basata sullo stesso, è la crescita dei tempi di calcolo rispetto alle dimensioni delle istanze. In particolare i risultati ottenuti con l'euristica sulle istanze casuali hanno la caratteristica di variare notevolmente, crescendo con le dimensioni del problema, ma anche con la complessità del grafo rappresentante l'istanza.

Tempi di calcolo dell'euristica basata sul metodo esatto

Istanze n nodi	μ euristica	σ euristica	μ montecarlo	σ montecarlo	μ montecarlo guid	σ montecarlo guid	μ genetico	σ genetico
50	1243.5	1196.5	57	0.01	58	0.02	847	0.02
100	4722	4707	59	0.02	58	0.02	916	0.02
200	8485.5	8383.5	59	0.01	59	0.01	918	0.01

Figura 7.16: Un'analisi dei tempi di calcolo dei vari metodi, riportati in secondi. μ =media, σ =deviazione standard

Tale caratteristica non viene ritrovata nei tempi di calcolo nè dei metodi Monte Carlo, nè dell'algoritmo genetico. Per i primi i tempi di calcolo sono estremamente ridotti, nell'ordine circa

[3] Ricordiamo che tutte le prove, di [3], [1] e di questo lavoro, sono state fatte con $k = 3$

del minuto, per ogni istanza; per quanto riguarda l'algoritmo genetico, i tempi sono strettamente dipendenti dai parametri utilizzati (dimensione della popolazione iniziale, numero di generazioni e percentuale di mutazione), con i parametri descritti in precedenza i tempi di calcolo sono di circa 900 secondi per ciascuna istanza.

Capitolo 8

Conclusioni, sviluppi e altri campi di applicazione

Il lavoro svolto ha avuto lo scopo di risolvere un problema di ottimizzazione su grafo, in modo più efficiente di quanto fosse stato fatto in precedenti lavori.
Sono stati dunque implementati due metodi euristici basati sul metodo Monte Carlo e Monte Carlo guidato, e un metodo metaeuristico, l'algoritmo Genetico. Questi, a differenza del metodo di PLI utilizzato in precedenza, hanno tutti portato ad una notevole diminuzione dei tempi di calcolo, scopo principale dell'utilizzo di metodi non esatti.
Inoltre con l'algoritmo Genetico sono stati ottenuti valori di profitti piuttosto elevati .
Come è stato già verificato ed analizzato per il grafo griglia, il problema reale riscontrato sia nel caso dell'utilizzo di metodi basati sui Monte Carlo, sia dell'algoritmo genetico, è di trovare un metodo efficiente per generare i percorsi di partenza, ovvero per generare l'insieme di soluzioni iniziali su cui lavorano i metodi euristici e metaeuristici. Dunque benchè siano stati trovati svariati metodi non esatti, efficienti per la risoluzione del problema dato, l'applicazione ad un insieme di soluzioni iniziali non adeguato ne ha limitato notevolmente le potenzialità. Il più importante ambito di miglioramento dunque, se si vuole proseguire sulla strada della risoluzione tramite metodi euristici e metaeuristici, è quello appena descritto.
In conclusione si noti che il problema descritto, benchè prenda spunto dal reale problema di sanzionamento delle autovetture in sosta irregolare, ha numerosi altri campi di applicazione:

- raccolta immondizia;
- rimozione della neve dalle strade;
- raccolta generi agricoli.

Appendice A

Codici utilizzati

A.1 Il codice della struttura forward star

```
/*
arcs.c
 */

#include <stdlib.h>
#include <string.h>
#include <stdio.h>

#include "debug.h"
#include "fwdstar.h"

int arc_data_less_cost(struct arc_data_t* a, struct arc_data_t* b)
{
return (a->cost < b->cost);
}

int arc_data_less_risk(struct arc_data_t* a, struct arc_data_t* b)
{
return (a->risk < b->risk);
}

int arc_data_add(struct arc_data_t* a, struct arc_data_t* b)
```

```
{
a->cost = infinity_sum(a->cost, b->cost);
a->risk = infinity_sum(a->risk, b->risk);
return 0;
}

int fwd_init(struct fwdstar_t* t, unsigned n, unsigned a)
{
printd("%p %d %d\n", t, n, a);

if (!(t->idx = malloc(sizeof(struct arc_t*)*n)))
{
perror("malloc");
return -1;
}
memset(t->idx, 0, sizeof(struct arc_t*)*n);

if (!(t->arc = malloc(sizeof(struct arc_t)*(a + 1))))
{
perror("malloc");
free(t->idx);
return -1;
}
memset(t->arc, 0, sizeof(struct arc_t)*(a + 1));
#if 0
if (!(t->data = malloc(sizeof(struct arc_data_t)*n)))
{
perror("malloc");
free(t->idx);
free(t->arc);
return -1;
}
#else
t->data = NULL;
#endif
```

```
t->N = n;
t->A = a;
t->dfree = t->arc;

return 0;
}

int fwd_read_from_file(struct fwdstar_t* star, FILE* infile, int retarcs)
{
char str[255];
unsigned nodes, arcs = 0;
enum {INIT, ARCS} state = INIT;

struct arc_t newarc;

if (!infile || !star)
{
error("%s: Bad (NULL) parameter.\n", __FUNCTION__);
return -1;
}

while (fgets(str, 255, infile))
if ('#' == str[0])
continue;
else if (state == INIT && 2 == sscanf(str, "%u %u", &nodes, &arcs))
{
if (retarcs)
arcs *= 2;
//nodes += 1;
fwd_init(star, nodes, arcs);
state = ARCS;
}
else if (state == ARCS && 4 == sscanf(str, "%u %u %f %f", &newarc.src, &newarc.dest,
&newarc.data.risk, &newarc.data.cost))
{
```

```
if (newarc.src >= star->N || newarc.dest >= star->N)
{
error("Something's bad with input data: %u->%u is not a valid arc (outside nodes space!)\n
(Maybe your index starts from 1 instead of 0...)\n", newarc.src, newarc.dest);
break;
}
fwd_pusharc(star, &newarc);
arcs--;

if (retarcs)
{
struct arc_t newarc2;

newarc2.src = newarc.dest;
newarc2.dest = newarc.src;
newarc2.data.cost = newarc.data.cost;
newarc2.data.risk = newarc.data.risk;
fwd_pusharc(star, &newarc2);
arcs--;
}
}
else
error("%s: Unrecognized line %s", __FUNCTION__, str);

if (arcs != 0)
error("%s: Arc number mismatch (%d to go?)\n", __FUNCTION__, arcs);

return 0;
}

void fwd_printarcs(struct fwdstar_t* t)
{
unsigned i;
struct arc_t* j;
```

```
printd("%p\n", t);

if (!t)
return;

for (i = 0; i < t->N; i++)
{
printf("%4u", i);
if ((j = t->idx[i]))
while (j->src == i)
{
printf(" ->%u:", j->dest);
arc_data_print(&(j->data));
j++;
}
printf("\n");
}
}

#define arc_swap(a, b) { struct arc_t tmp;\
memcpy(&tmp, a, sizeof(struct arc_t));\
memcpy(a, b, sizeof(struct arc_t));\
memcpy(b, &tmp, sizeof(struct arc_t));\
   }

/* just a stub, use qsort instead */
int fwd_reindex(struct fwdstar_t* t)
{
int modified = 1;

if (!t)
return -1;

while (modified)
{
```

```
unsigned int p = 0xffffffff;
unsigned int i;
modified = 0;
for (i = 0; i < t->A; i++)
{
if (t->arc[i].src >= t->N)
{
error("%s: WTF! arc %u starts from node %u of %u\n", __FUNCTION__, i, t->arc[i].src, t->N);
return 1;
}
if (p != t->arc[i].src)
{
p = t->arc[i].src;
t->idx[p] = &(t->arc[i]);
}
if ((i < t->A - 1) && t->arc[i].src > t->arc[i + 1].src)
{
arc_swap(&(t->arc[i]), &(t->arc[i + 1]));
modified = 1;
}
}
}

return 0;
}

int fwd_pusharc(struct fwdstar_t* net, struct arc_t* a)
{
memcpy(net->dfree, a, sizeof(struct arc_t));

net->dfree++;

return 0;
}
```

```
int fwd_revert(struct fwdstar_t* main, struct fwdstar_t* inverse)
{
unsigned int i;

if (!main || !inverse)
return -1;

fwd_init(inverse, main->N, main->A);

for (i = 0; i < main->A; i++)
{
inverse->arc[i].src = main->arc[i].dest;
inverse->arc[i].dest = main->arc[i].src;
inverse->arc[i].data = main->arc[i].data;
}

return fwd_reindex(inverse);
}
/* EOF */
```

A.2 Il codice dell'algoritmo di Dijkstra

```
/* dijkstra.c */
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

#include "debug.h"

#include "dijkstra.h"

#define VISITED(n) (1 == n)
#define VISIT(n) {n = 1;}
```

```
int fwd_dijk_init(struct fwdstar_t* g)
{
if (g)
{
if (!g->data)
{
if (!(g->data = malloc(sizeof(struct dijkstra_res_t)*g->N)))
{
perror("malloc");
return -1;
}
} else {
error("Forward star data field already initialized.\n");
return -1;
}
} else {
error("Forward star is null.\n");
return -1;
}

return 0;
}

void fwd_dijk_free(struct fwdstar_t* g)
{
if (g)
free(g->data);
}

int dijkstra(struct fwdstar_t* g, arc_data_f less, unsigned so, unsigned si,
struct arc_data_t* res)
{
int i, smallidx;
char *visited;
unsigned count;
```

```
struct arc_data_t smallest;

printd("%d->%d\n", so, si);

if (!g)
return -1;

count = g->N;

visited = malloc(g->N);
memset(visited, 0x0, g->N);

if (!g->data)
{
error("Forward star data field not initialized, maybe you need to call fwd_dijk_init.\n");
return -1;
}

for (i = 0; i < g->N; i++)
arc_data_set_infinity(&fwd_dijk_res(g, i).data);

memset(&fwd_dijk_res(g, so), 0x0, sizeof(struct dijkstra_res_t));

while (count)
{
arc_data_set_infinity(&smallest);
smallidx = -1;
for (i = 0; i < g->N; i++)
if (!visited[i])
if (less(&fwd_dijk_res(g, i).data, &smallest))
{
smallest = fwd_dijk_res(g, i).data;
smallidx = i;
}
```

```
if (-1 == smallidx || (smallidx == si && si != so))
break;

/*printf("smallest=%u idx=%u\n", smallest, smallidx);*/

VISIT(visited[smallidx]);
count--;

if (g->idx[smallidx])
{
struct arc_t *ap = g->idx[smallidx];

while (ap->src == smallidx)
{
printd("%d->%d\n", ap->src, ap->dest);
if (!VISITED(visited[ap->dest]))
{
struct arc_data_t alt;
memcpy(&alt, &(ap->data), sizeof(struct arc_data_t));
arc_data_add(&alt, &fwd_dijk_res(g, smallidx).data);
#if 0
arc_print_data(&(((struct arc_data_t*) g->data)[smallidx]));
printf(" + ");
arc_print_data(&(ap->data));
printf(" = ");
arc_print_data(&alt);
printf("\n");
#endif
if (less(&alt, &fwd_dijk_res(g, ap->dest).data))
{
memcpy(&fwd_dijk_res(g, ap->dest).data, &alt, sizeof(struct arc_data_t));
fwd_dijk_res(g, ap->dest).prev = smallidx;
}
}
ap++;
```

```
}
}
}

free(visited);

#if 0
printf("D%02d: ", so);
for (i = 0; i < g->N; i++)
arc_data_print(&(((struct arc_data_t*) g->data)[i]));
printf("\n");
#endif

if (res)
memcpy(res, &fwd_dijk_res(g, si), sizeof(struct arc_data_t));

return DIJKSTRA_OK;
}

/* EOF */
```

A.3 Il codice in R del Monte Carlo semplice

```
#clean up
 rm(list=ls(all=T))


#Lettura della matrice di dati relativa all'istanz

dati=read.table('dati.txt')
#dati[,4]=-dati[,4]
gra = dati
gra1= as.matrix(gra)
```

```
gra<-as.matrix(gra)
gra=data.frame(gra)
id=c(1:dim(gra)[[1]])
gra=cbind(gra,id)
attach(gra)
n<-dim(gra)



#Funzione che calcola un percorso random sulla rete

PAT<-function(gra)
{
tempimax<-0
profits=0

gra1=as.matrix(gra)
#prendo la lunghezza del cammino random
lunghezzacamm<-sample(6:15, 1, replace = TRUE)
#start<-sample(1:n[1],1, replace = TRUE)
cammrand<-matrix(0,30,5)
cammrand[1,]<-gra1[1,]
seg<-1
#creo i percorsi
for(l in 2:30) {
possibili1<-which(gra1[,1]==cammrand[l-1,2])
possibili2<-which(gra1[,2]==cammrand[l-1,2])
possibili<-c(possibili2,possibili1)
possibili<-possibili[which(possibili!=seg)]
seg<-sample(possibili,1, replace = TRUE)
tempimax<-tempimax+gra1[seg,3]
profits<-profits+gra1[seg,4]
if (l>6 & seg==1) {
if (cammrand[l-1,2]==gra1[seg,1]) {
cammrand[l,]<-gra1[seg,]
```

```
}
else {
cammrand[l,1]<-gra1[seg,2]
cammrand[l,2]<-gra1[seg,1]
cammrand[l,3]<-gra1[seg,3]
cammrand[l,4]<-gra1[seg,4]
cammrand[l,5]<-gra1[seg,5]
}
break
}
if (cammrand[l-1,2]==gra1[seg,1]) {
cammrand[l,]<-gra1[seg,]
}
else {
cammrand[l,1]<-gra1[seg,2]
cammrand[l,2]<-gra1[seg,1]
cammrand[l,3]<-gra1[seg,3]
cammrand[l,4]<-gra1[seg,4]
cammrand[l,5]<-gra1[seg,5]
}}
tempi=sum(cammrand[,3])
pro=sum(cammrand[,4])
arcx=cammrand[,5]
cammrand
tempimax
pro
arcx
if (tempi <180 | tempi > 240 ){
PAT(gra)
}
else{
return (cammrand)
}
}
```

```
##################################


#Creazione di 50 percorsi random
m=50
P=PAT(gra)
for(i in 1:(m-1)){
A=PAT(gra)
P=cbind(P,A)
}



#Funzione che toglie le righe di soli zeri dalla matrice

toz<-function(S){
i=0
while (i<dim(S)[[1]]){
i=i+1
if (S[i,1]==0 & S[i,2]==0){
S=S[-i,]
}
}
return(S)
}


# Funzione che calcola il profitto relativo a una combinazione di k percorsi,
 tenendo conto del turnover.


Profitto<-function(P){
Pze= c()
for (i in 1:(dim(P)[[2]]/5  ) ){
```

```
Pze=rbind(Pze,P[,(5*i-4):(i*5)])
}
Pze
n=dim(P)[[1]]
Pzer=cbind(Pze,c(rep(c(1:50),times=c(rep(30,50)))))
Pzerc= Pzer[order(Pzer[,5]),]
ZPZer=Pzerc[,5:6]
k=3
dati=ZPZer
K=kmeans(dati,k)$cluster
dati2=cbind(dati,K)
#arco cammin cluster
I1 =dati2[which(dati2[,3]=='1'),]
I2 =dati2[which(dati2[,3]=='2'),]
I3 =dati2[which(dati2[,3]=='3'),]
I10=rep(0,50)
for (i in 1:50 ){
for( j in 1: dim(I1)[[1]]) {
if( I1[i,2]== j){
I10[j] = I10 [j]+1
}}}
I20=rep(0,50)
for (i in 1:50 ){
for( j in 1: dim(I2)[[1]]) {
if( I2[i,2]== j){
I20[j] = I20 [j]+1
}}}
I30=rep(0,50)
for (i in 1:50 ){
for( j in 1: dim(I3)[[1]]) {
if( I3[i,2]== j){
I30[j] = I30 [j]+1
}}}
I10
I20
```

```
I30
CL1=c()
CL2=c()
CL3=c ()
for (i in 1:50 ) {
if(   max(I10[i], I20[ i ],I30 [i ])==I10[i]) {
CL1=c(CL1,i)
}
if(   max(I10[i], I20[ i ],I30 [i ])==I20[i] ){
CL2=c(CL2,i)
}
if(    max(I10[i], I20[ i ],I30 [i ])==I30[i]){
CL3=c(CL3,i)
}
}
CL1
CL2
CL3
for(q in 1:length(CL1)){
for(w in 1:length(CL2)){
for(t in 1:length( CL3)){
if(CL1[q]  != CL3[ t] & CL2[w] != CL1[q]  & CL3[t] != CL2[ w ]){
A=P[,(5*  CL1[q] -4): (5*CL1[q])]
B=P[,(5*  CL2[w] -4): (5*CL2[w])]
C=P[,(5*  CL3[t] -4): (5*CL3[t])]
}
}
}
}
A=cbind(A,c(0,cumsum(A[1:dim(A)[[1]]-1,3])))
B=cbind(B,c(0,cumsum(B[1:dim(B)[[1]]-1,3])))
C=cbind(C,c(0,cumsum(C[1:dim(C)[[1]]-1,3])))
n= dim(A) [[1]]
A=A[,-5]
B=B[,-5]
```

```
C=C[,-5]
A
B
C
ta=rep(0,5)
tb=rep(0,5)
tc=rep(0,5)
tab=rep(0,5)
tcb=rep(0,5)
tac=rep(0,5)
for (k in 1:n){
for ( j in 1:n){
if (A[k,1]==A[j,1] & A[k,2]==A[j,2]  & j!=k | A[k,1]==A[j,2] & A[k,2]==A[j,1 ] & j!=k){
ta= rbind(ta, A[k,],A[j,])}
if (B[k,1]==B[j,1] & B[k,2]==B[j,2] & j!=k  | B[k,1]==B[j,2] & B[k,2]==B[j,1 ] & j != k){
tb= rbind(tb, B[k,],B[j,])
}
if (C[k,1]==C[j,1] & C[k,2]==C[j,2] & j!=k | C[k,1]==C[j,2] & C[k,2]==C[j,1 ] & j!=k ){
tc= rbind(tc, C[k,],C[j,])
}
if (A[k,1]==B[j,1] & A[k,2]==B[j,2] | A[k,1]==B[j,2] & A[k,2]==B[j,1 ]){
tab= rbind(tab, A[k,],B[j,])
}
if (C[k,1]==B[j,1] & C[k,2]==B[j,2] | C[k,1]==B[j,2] & C[k,2]==B[j,1 ]){
tcb= rbind(tcb, C[k,],B[j,])
}
if (A[k,1]==C[j,1] & A[k,2]==C[j,2] | A[k,1]==C[j,2] & A[k,2]==C[j,1 ]){
tac= rbind(tac, A[k,],C[j,])
}
}}
#ta=ta[which(duplicated(ta[,5])==FALSE),]
#tb=tb[which(duplicated(tb[,5])==FALSE),]
#tc=tc[which(duplicated(tc[,5])==FALSE),]
#tab=tab[which(duplicated(tab[,5])==FALSE),]
#tcb=tcb[which(duplicated(tcb[,5])==FALSE),]
```

```
#tac=tac[which(duplicated(tac[,5])==FALSE),]
tempiarchiuguali=rbind(ta,tb,tc,tab,tcb,tac)
tempiarchiuguali
TEMPIARCHIUGUALI= toz(tempiarchiuguali)
T=toz(tempiarchiuguali)
T=T[which(duplicated(T[,5])==FALSE),]
PROF= sum(T[,4])
T= T[order(T[,5]),]
PROFITTOSINGL=sum(A[,4])+sum(B[,4])+sum(C[,4])-sum(T[,4])
for (k in 1:dim(T)[[1]]){
for ( j in k:dim(T)[[1]]){
if (T[k,1]==T[j,1] & T[k,2]==T[j,2]  & j!=k | T[k,1]==T[j,2] & T[k,2]==T[j,1 ] & j!=k){
if(T[k,5]<T[j,5]){
DeltaT=T[j,5]-T[k,5]
T[j,4]=min((DeltaT/90)*T[j,4],T[j,4])
}
else if (T[k,5]>T[j,5]){
DeltaT=-T[j,5]+T[k,5]
T[j,4]=min((DeltaT/90)*T[j,4],T[j,4])
}
}
}
}
PROFTURNOVE= sum(T[,4])
PROFITTO_TOTALE_DELUXE= PROFTURNOVE + PROFITTOSINGL
D=list( PROFITTO_TOTALE_DELUXE)
PROFITTO_TOTALE_DELUXE
if(PROFITTO_TOTALE_DELUXE < 10){
Profitto(P)
}
else{
return(D)
}
}
Y=0
```

```
for(i in 1:50 ){
Y=c(Y,Profitto(P))
}
Y
```

A.4 Il codice in R del Monte Carlo guidato

```
#Lettura della matrice di dati relativi all'istanz
dati=read.table('50904.txt')
#dati[,4]=-dati[,4]
n2<- dim(dati)[[1]]
E=c(1:n2)
dati= cbind(dati,E)
gra = dati
gra1= as.matrix(gra)
gra<-as.matrix(gra)
gra=data.frame(gra)
attach(gra)
n<-dim(gra)

#Calcolo del vettore di probabilità per ciascun arco . Tale probabilità si basa
 sul profitto degli archi. Più è alto il valore del profitto su un arco, e più
  è alta la probabilità che venga preso come arco nel percors.

Pro= 0
grar=gra
grar[,4]=grar[,4]+ rnorm( dim(gra)[[1]],0.5,0.01)
for(i in 0:49){
b=which(gra[,1]==i)
v=dim(b)
B=grar[b,]
j=sum(B[,4])
```

```
k= B[,4]/j
Pro=c(Pro,k)
}
Pro=Pro[2:length(Pro)]

# Funzione che calcola un percorso random guidato sulla rete stradale.

PAT<-function(gra,Pro)
{
tempimax<-0
profits=0

gra1=as.matrix(gra)
#prendo la lunghezza del cammino random
lunghezzacamm<-sample(6:15, 1, replace = TRUE)
#start<-sample(1:n[1],1, replace = TRUE)
cammrand<-matrix(0,30,5)
cammrand[1,]<-gra1[1,]
seg<-1
for(l in 2:30) {
possibili1<-which(gra1[,1]==cammrand[l-1,2])
possibili2<-which(gra1[,2]==cammrand[l-1,2])
possibili<-c(possibili2,possibili1)
possibili<-possibili[which(possibili!=seg)]
Probb=Pro[possibili]
#cat(length(Probb))
#cat("\n")
#cat(Probb)
#cat("\n")
#cat(length(possibili))
if (length(possibili)!=1) {
seg<-sample(possibili,1, replace = TRUE,prob=Probb)
}
else{ seg<-sample(possibili,1, replace = TRUE)
}
```

```
tempimax<-tempimax+gra1[seg,3]
profits<-profits+gra1[seg,4]
if (l>6 & seg==1) {
if (cammrand[l-1,2]==gra1[seg,1]) {
cammrand[l,]<-gra1[seg,]
}
else {
cammrand[l,1]<-gra1[seg,2]
cammrand[l,2]<-gra1[seg,1]
cammrand[l,3]<-gra1[seg,3]
cammrand[l,4]<-gra1[seg,4]
cammrand[l,5]<-gra1[seg,5]
}
break
}
if (cammrand[l-1,2]==gra1[seg,1]) {
cammrand[l,]<-gra1[seg,]
}
else {
cammrand[l,1]<-gra1[seg,2]
cammrand[l,2]<-gra1[seg,1]
cammrand[l,3]<-gra1[seg,3]
cammrand[l,4]<-gra1[seg,4]
cammrand[l,5]<-gra1[seg,5]
}
}
tempi=sum(cammrand[,3])
pro=sum(cammrand[,4])
arcx=cammrand[,5]
#stampo il tutto alla fine
cammrand
tempimax
pro
arcx
if (tempi <180 | tempi > 240 ){
```

```
PAT(gra,Pro)
}
else{
return (cammrand)
}
}

#Creazione di 50 percorsi random

#PAT(gra,Pro)
m=50
P=PAT(gra,Pro)
for(i in 1:(m-1)){
A=PAT(gra,Pro)
P=cbind(P,A)
}



#Funzione che toglie le righe di soli zeri dalla matrice

toz<-function(S){
i=0
while (i<dim(S)[[1]]){
i=i+1
if (S[i,1]==0 & S[i,2]==0){
S=S[-i,]
}
}
return(S)
}



#Funzione che calcola il profitto relativo a una combinazione di k percorsi,
tenendo conto del turnover
```

```
Profitto<-function(P){
Pze= c()
for (i in 1:(dim(P)[[2]]/5  ) ){
Pze=rbind(Pze,P[,(5*i-4):(i*5)])
}
Pze
n=dim(P)[[1]]
Pzer=cbind(Pze,c(rep(c(1:50),times=c(rep(30,50)))))
Pzerc= Pzer[order(Pzer[,5]),]
ZPZer=Pzerc[,5:6]
k=3
dati=ZPZer
K=kmeans(dati,k)$cluster
dati2=cbind(dati,K)
#arco cammin cluster
I1 =dati2[which(dati2[,3]=='1'),]
I2 =dati2[which(dati2[,3]=='2'),]
I3 =dati2[which(dati2[,3]=='3'),]
I10=rep(0,50)
for (i in 1:50 ){
for( j in 1: dim(I1)[[1]]) {
if( I1[i,2]== j){
I10[j] = I10 [j]+1
}}}
I20=rep(0,50)
for (i in 1:50 ){
for( j in 1: dim(I2)[[1]]) {
if( I2[i,2]== j){
I20[j] = I20 [j]+1
}}}
I30=rep(0,50)
for (i in 1:50 ){
for( j in 1: dim(I3)[[1]]) {
if( I3[i,2]== j){
```

```
I30[j] = I30 [j]+1
}}}
I10
I20
I30
CL1=c()
CL2=c()
CL3=c ()
for (i in 1:50 ) {
if(   max(I10[i], I20[ i ],I30 [i ])==I10[i]) {
CL1=c(CL1,i)
}
if(   max(I10[i], I20[ i ],I30 [i ])==I20[i] ){
CL2=c(CL2,i)
}
if(    max(I10[i], I20[ i ],I30 [i ])==I30[i]){
CL3=c(CL3,i)
}
}
CL1
CL2
CL3
for(q in 1:length(CL1)){
for(w in 1:length(CL2)){
for(t in 1:length( CL3)){
if(CL1[q]  != CL3[ t] & CL2[w] != CL1[q]  & CL3[t] != CL2[ w ]){
A=P[,(5*  CL1[q] -4): (5*CL1[q])]
B=P[,(5*  CL2[w] -4): (5*CL2[w])]
C=P[,(5*  CL3[t] -4): (5*CL3[t])]
}
}
}
}

A=cbind(A,c(0,cumsum(A[1:dim(A)[[1]]-1,3])))
```

```
B=cbind(B,c(0,cumsum(B[1:dim(B)[[1]]-1,3])))
C=cbind(C,c(0,cumsum(C[1:dim(C)[[1]]-1,3])))
n= dim(A) [[1]]
A=A[,-5]
B=B[,-5]
C=C[,-5]
A
B
C
ta=rep(0,5)
tb=rep(0,5)
tc=rep(0,5)
tab=rep(0,5)
tcb=rep(0,5)
tac=rep(0,5)
for (k in 1:n){
for ( j in 1:n){
if (A[k,1]==A[j,1] & A[k,2]==A[j,2]  & j!=k | A[k,1]==A[j,2] & A[k,2]==A[j,1 ] & j!=k){
ta= rbind(ta, A[k,],A[j,])}
if (B[k,1]==B[j,1] & B[k,2]==B[j,2] & j!=k  | B[k,1]==B[j,2] & B[k,2]==B[j,1 ] & j != k){
tb= rbind(tb, B[k,],B[j,])
}
if (C[k,1]==C[j,1] & C[k,2]==C[j,2] & j!=k | C[k,1]==C[j,2] & C[k,2]==C[j,1 ] & j!=k ){
tc= rbind(tc, C[k,],C[j,])
}
if (A[k,1]==B[j,1] & A[k,2]==B[j,2] | A[k,1]==B[j,2] & A[k,2]==B[j,1 ]){
tab= rbind(tab, A[k,],B[j,])
}
if (C[k,1]==B[j,1] & C[k,2]==B[j,2] | C[k,1]==B[j,2] & C[k,2]==B[j,1 ]){
tcb= rbind(tcb, C[k,],B[j,])
}
if (A[k,1]==C[j,1] & A[k,2]==C[j,2] | A[k,1]==C[j,2] & A[k,2]==C[j,1 ]){
tac= rbind(tac, A[k,],C[j,])
}
}}
```

```
#ta=ta[which(duplicated(ta[,5])==FALSE),]
#tb=tb[which(duplicated(tb[,5])==FALSE),]
#tc=tc[which(duplicated(tc[,5])==FALSE),]
#tab=tab[which(duplicated(tab[,5])==FALSE),]
#tcb=tcb[which(duplicated(tcb[,5])==FALSE),]
#tac=tac[which(duplicated(tac[,5])==FALSE),]
tempiarchiuguali=rbind(ta,tb,tc,tab,tcb,tac)
tempiarchiuguali
TEMPIARCHIUGUALI= toz(tempiarchiuguali)
T=toz(tempiarchiuguali)
T=T[which(duplicated(T[,5])==FALSE),]
PROF= sum(T[,4])
T= T[order(T[,5]),]
PROFITTOSINGL=sum(A[,4])+sum(B[,4])+sum(C[,4])-sum(T[,4])
for (k in 1:dim(T)[[1]]){
for ( j in k:dim(T)[[1]]){
if (T[k,1]==T[j,1] & T[k,2]==T[j,2]  & j!=k | T[k,1]==T[j,2] & T[k,2]==T[j,1 ] & j!=k){
if(T[k,5]<T[j,5]){
DeltaT=T[j,5]-T[k,5]
T[j,4]=min((DeltaT/90)*T[j,4],T[j,4])
}
else if (T[k,5]>T[j,5]){
DeltaT=-T[j,5]+T[k,5]
T[j,4]=min((DeltaT/90)*T[j,4],T[j,4])
}
}
}
}
PROFTURNOVE= sum(T[,4])
PROFITTO_TOTALE_DELUXE= PROFTURNOVE + PROFITTOSINGL
D=list( PROFITTO_TOTALE_DELUXE)
PROFITTO_TOTALE_DELUXE
if(PROFITTO_TOTALE_DELUXE < 10){
Profitto(P)
}
```

```
else{
return(D)
}
}
Y=0
for(i in 1:50 ){
Y=c(Y,Profitto(P))
}



#Codice per il clustering


Pze= c()
for (i in 1:(dim(P)[[2]]/5  ) ){
Pze=rbind(Pze,P[,(5*i-4):(i*5)])
}
Pze
n=dim(P)[[1]]
Pzer=cbind(Pze,c(rep(c(1:50),times=c(rep(30,50)))))
Pzerc= Pzer[order(Pzer[,5]),]
ZPZer=Pzerc[,5:6]
k=3
dati=ZPZer
K=kmeans(dati,k)$cluster
dati2=cbind(dati,K)
I1 =dati2[which(dati2[,3]=='1'),]
I2 =dati2[which(dati2[,3]=='2'),]
I3 =dati2[which(dati2[,3]=='3'),]
I10=rep(0,50)
for (i in 1:50 ){
for( j in 1: dim(I1)[[1]]) {
if( I1[i,2]== j){
I10[j] = I10 [j]+1
}}}
```

```
I20=rep(0,50)
for (i in 1:50 ){
for( j in 1: dim(I2)[[1]]) {
if( I2[i,2]== j){
I20[j] = I20 [j]+1
}}}
I30=rep(0,50)
for (i in 1:50 ){
for( j in 1: dim(I3)[[1]]) {
if( I3[i,2]== j){
I30[j] = I30 [j]+1
}}}
I10
I20
I30
CL1=c()
CL2=c()
CL3=c ()
for (i in 1:50 ) {
if(   max(I10[i], I20[ i ],I30 [i ])==I10[i]) {
CL1=c(CL1,i)
}
if(   max(I10[i], I20[ i ],I30 [i ])==I20[i] ){
CL2=c(CL2,i)
}
if(    max(I10[i], I20[ i ],I30 [i ])==I30[i]){
CL3=c(CL3,i)
}
}
CL1
CL2
CL3
for(q in 1:length(CL1)){
for(w in 1:length(CL2)){
for(t in 1:length( CL3)){
```

```
if(CL1[q]  != CL3[ t] & CL2[w] != CL1[q]  & CL3[t] != CL2[ w ]){
A=P[,(5*  CL1[q] -4): (5*CL1[q])]
B=P[,(5*  CL2[w] -4): (5*CL2[w])]
C=P[,(5*  CL3[t] -4): (5*CL3[t])]
}
}
}
}
```

A.5 Il codice in R del Genetico

```
#CODICE PER GENERARE I CAMMINI CON IL PACCHETTO IGRAPH

get.random.loop <- function(graph, Cc) {
s=dim(Cc)[[1]]
Estart=sample(1:(s-1),1,replace=F)
start=Cc[Estart,1]
ENd=Cc[Estart,2]
path=end=start
while(TRUE){
end<-sample(neighbors(graph,end),1,replace=F)
path=c(path,end)
if(end==ENd) break
}
D=list(path=path,b=c(start,end,ENd))
return(D)
}

#CODICE PER LA FITNESSS

fitness<-function(S){
fits=c()
```

```
for (i in 1:(dim(S)[[2]]/5 ) ){
fits=cbind(fits,S[,5*i-1])
F=colSums(fits)
}
return(F)
}

#CODICE PER L ALGORITMO GENETICO

GENETIC<-function(U_uA,U_uB,U_uC,U_uP){
UP=max(U_uP)
fitnA=fitness(U_uA)
fitnB=fitness(U_uB)
fitnC=fitness(U_uC)
ProA=fitnA/sum(fitnA)
ProB=fitnB/sum(fitnB)
ProC=fitnC/sum(fitnC)
PROBA=(fitnA+fitnB+fitnC)/((sum(fitnA)+sum(fitnB)+sum(fitnC)))
All=sample(5*(1:m),2,replace=FALSE,prob=PROBA)
Al1=All[1]
Al2=All[2]
A=U_uA[,(Al1-4):Al1]
B=U_uB[,(Al1-4):Al1]
C=U_uC[,(Al2-4):Al2]
T=rbind(A,B,C)
T=toz(T)
T= T[order(T[,5]),]
T=T[which(duplicated(T[,1:5])==FALSE),]
for (k in 1:dim(T)[[1]]){
for ( j in k:dim(T)[[1]]){
if (T[k,1]==T[j,1] & T[k,2]==T[j,2]  & j!=k | T[k,1]==T[j,2] & T[k,2]==T[j,1 ] & j!=k){
if(T[k,5]<T[j,5]){
DeltaT=T[j,5]-T[k,5]
T[j,4]=min((DeltaT/90)*T[j,4],T[j,4])
}
```

```
else if (T[k,5]>T[j,5]){
DeltaT=-T[j,5]+T[k,5]
T[j,4]=min((DeltaT/90)*T[j,4],T[j,4])
}
}
}
}
PROFITTOTOta= sum(T[,4])
D=list( A=A,B=B,C=C, Profitt=PROFITTOTOta,Al1=Al1,Al2=Al2)
PROFITTOTOta
return(D)
}
GENETIC(U_uA,U_uB,U_uC,U_uP)

#CODICE PER LE GENERAZIONI

LRGr<-function(U_uA,U_uB,U_uC,U_uP)
{
Aa=c()
Bb=c()
Cc=c()
Al1=c(0)
Al2=c(0)
Ppp=c()
for(i in 1:m){
E=GENETIC(U_uA,U_uB,U_uC,U_uP)
for(j in length(Al1)){
if(Al1[j]==E$Al1 | Al2[j]==E$Al2){
E=GENETIC(U_uA,U_uB,U_uC,U_uP)
Aa=cbind(Aa,E$A)
Bb=cbind(Bb,E$B)
Cc=cbind(Cc,E$C)
Ppp=c(Ppp,E$Profitt)
j=j-1
}
```

```
else{
Al1=c(Al1,E$Al1)
Al2=c(Al2,E$Al2)
Aa=cbind(Aa,E$A)
Bb=cbind(Bb,E$B)
Cc=cbind(Cc,E$C)
Ppp=c(Ppp,E$Profitt)
}
}
}
xyja=which(max(Ppp)==Ppp)
Axyja=Aa[,(((xyja[1]*5))-4):((xyja[1]*5))]
Bxyja=Bb[,(((xyja[1]*5))-4):((xyja[1]*5))]
Cxyja=Cc[,(((xyja[1]*5))-4):((xyja[1]*5))]
Pxyja=max(Ppp)
U_uA=Aa
U_uB=Bb
U_uC=Cc
AL1=Al1[xyja]
AL2=Al2[xyja]
D=list(U_uA=U_uA,U_uB=U_uB,U_uC=U_uC, U_uP=U_uP,Axyja=Axyja,Bxyja=Bxyja,Cxyja=Cxyja,Pxyja=Pxyja,Al
return(D)}

#CODICE PER LE MUTAZION

MUTAZ<- function(gra, C){
Cc=toz(C)
data=gra[,1:2]
write.table(data,file="e.net.txt",append=FALSE,row.names=FALSE,col.names=FALSE )
graph<-read.graph('e.net.txt',directed=FALSE)
B=get.random.loop(graph,Cc)
while(TRUE){
if(length(B$path)>7){
B=get.random.loop(graph,Cc)
}
```

```
else break
}
S=PERC(B$path)
if(is.matrix(S)==TRUE){
S4=matrix(0, dim(S)[[1]],5)
}else {
S4=matrix(0, 1,5)
S4[1]=S[1]
S4[2]=S[2]
}
if(is.matrix(S)==TRUE){
S4[,1] = S[,1]
S4[,2] = S[,2]
for(i in 1:dim(S)[[1]])
for(j in 1:dim(gra)[[1]])
{
{
if(S[i,1]==gra[j,1]& S[i,2]==gra[j,2] | S[i,2]==gra[j,1]& S[i,1]==gra[j,2]){
S4[i,3]= gra[j,3]
S4[i,4]= gra[j,4]
S4[i,5]= gra[j,5]
}
}
}
}
S=S4
x=B$b[-2]
for(i in 1:dim(C)[[1]]){
if(C[i,2] ==x[2] & C[i,1] ==x[1]) {
C1=C[-i,]
C3=rbind(C1[1:i-1,],S,C1[i: dim(C1)[[1]],])
C3[1,5]=0
for ( i in 2:dim(C3)[[1]]){
C3[i,5]=C3[i-1,3]+C3[(i-1),5]
}
```

```
if(  sum(C3[,3])>240){
MUTAZ(gra,C)
}
else{
return(C3=C3[1:30,])
}
}
}}
```

Appendice B

Dati relativi all'istanza reale

B.1 Istanza reale codifica

```
12 18 VIA GIORGIO JAN
28 19 CORSO BUENOS AIRES
19 8 VIA GIUSEPPE BROGGI
33 5 VIA GIORGIO JAN
5 26 VIA GIORGIO JAN
26 12 VIA GIORGIO JAN
30 6 VIA GIOVANNI BATTISTA MORGAGNI
35 20 PIAZZALE LAVATER
20 36 PIAZZALE LAVATER
37 36 PIAZZALE LAVATER
20 37 PIAZZALE LAVATER
37 38 PIAZZALE LAVATER
39 31 PIAZZALE LAVATER
39 40 PIAZZALE LAVATER
40 34 PIAZZALE LAVATER
45 47 PIAZZALE BACONE
46 47 PIAZZALE BACONE
49 14 PIAZZALE BACONE
50 14 PIAZZALE BACONE
49 44 PIAZZALE BACONE
44 14 PIAZZALE BACONE
14 51 PIAZZALE BACONE
```

48 51 PIAZZALE BACONE

10 52 VIA BARTOLOMEO EUSTACHI

53 10 VIA BARTOLOMEO EUSTACHI

53 52 VIA BARTOLOMEO EUSTACHI

54 10 VIA BARTOLOMEO EUSTACHI

55 11 PIAZZA LIMA

55 56 VIA PLINIO

60 56 VIA PLINIO

57 56 VIA PLINIO

9 58 VIA BARTOLOMEO EUSTACHI

59 9 VIA BARTOLOMEO EUSTACHI

61 56 VIA PLINIO

48 62 VIA CARLO MATTEUCCI

62 51 PIAZZALE BACONE

2 63 VIA LAZZARO SPALLANZANI

1 2 VIA GIOVANNI OMBONI

3 4 CORSO BUENOS AIRES

7 8 VIA GIOVANNI MASERA

11 3 CORSO BUENOS AIRES

12 13 VIA BARTOLOMEO GIULIANO

16 15 VIA GIUSEPPE BROGGI

40 17 VIA ANTONIO STOPPANI

13 7 VIA GIOVANNI MASERA

6 16 VIA GIUSEPPE BROGGI

4 51 VIA GASPARE SPONTINI

21 22 VIA FRANCESCO REDI

23 24 VIA FRANCESCO REDI

22 23 VIA FRANCESCO REDI

15 23 VIA LODOVICO ZAMBELETTI

17 15 VIA LODOVICO ZAMBELETTI

17 25 VIA ANTONIO STOPPANI

13 57 VIA GIOVANNI MASERA

18 57 VIA ULISSE ALDROVANDI

43 18 VIA ULISSE ALDROVANDI

28 27 CORSO BUENOS AIRES

8 5 VIA GIUSEPPE BROGGI

7 26 VIA FRANCESCO REDI

27 7 VIA FRANCESCO REDI

16 22 VIA FILIPPO DE FILIPPI

34 16 VIA FILIPPO DE FILIPPI

1 28 CORSO BUENOS AIRES

26 29 VIA FRANCESCO REDI

29 21 VIA GIOVANNI BATTISTA MORGAGNI

5 30 VIA GIUSEPPE BROGGI

6 21 VIA GIOVANNI BATTISTA MORGAGNI

30 29 VIA GIOVANNI BATTISTA MORGAGNI

31 6 VIA GIOVANNI BATTISTA MORGAGNI

32 30 VIA GIOVANNI BATTISTA MORGAGNI

32 31 PIAZZALE LAVATER

33 32 PIAZZALE LAVATER

31 34 PIAZZALE LAVATER

2 35 VIA GIOVANNI OMBONI

35 33 PIAZZALE LAVATER

36 32 PIAZZALE LAVATER

38 39 PIAZZALE LAVATER

29 43 VIA GIOVANNI BATTISTA MORGAGNI

43 42 VIA GIOVANNI BATTISTA MORGAGNI

42 41 VIA GIOVANNI BATTISTA MORGAGNI

41 45 VIA GIOVANNI BATTISTA MORGAGNI

42 44 VIA GIOVANNI BATTISTA MORGAGNI

45 46 PIAZZALE BACONE

14 47 PIAZZALE BACONE

47 48 PIAZZALE BACONE

11 49 VIA FEDERICO OZANAM

3 50 VIA AMILCARE PONCHIELLI

50 49 PIAZZALE BACONE

52 46 VIA BARTOLOMEO EUSTACHI

41 53 VIA PLINIO

24 54 VIA BARTOLOMEO EUSTACHI

56 42 VIA PLINIO

58 24 VIA BARTOLOMEO EUSTACHI

25 59 VIA ACHILLE MAIOCCHI

27 60 CORSO BUENOS AIRES

63 19 VIA LAZZARO SPALLANZANI

21 64 VIA GIOVANNI BATTISTA MORGAGNI

64 41 VIA GIOVANNI BATTISTA MORGAGNI

1 2 VIA GIOVANNI OMBONI

3 4 CORSO BUENOS AIRES

7 8 VIA GIOVANNI MASERA

11 3 CORSO BUENOS AIRES

12 13 VIA BARTOLOMEO GIULIANO

16 15 VIA GIUSEPPE BROGGI

40 17 VIA ANTONIO STOPPANI

13 7 VIA GIOVANNI MASERA

6 16 VIA GIUSEPPE BROGGI

4 51 VIA GASPARE SPONTINI

21 22 VIA FRANCESCO REDI

23 24 VIA FRANCESCO REDI

22 23 VIA FRANCESCO REDI

15 23 VIA LODOVICO ZAMBELETTI

17 15 VIA LODOVICO ZAMBELETTI

17 25 VIA ANTONIO STOPPANI

13 57 VIA GIOVANNI MASERA

18 57 VIA ULISSE ALDROVANDI

43 18 VIA ULISSE ALDROVANDI

28 27 CORSO BUENOS AIRES

8 5 VIA GIUSEPPE BROGGI

7 26 VIA FRANCESCO REDI

27 7 VIA FRANCESCO REDI

16 22 VIA FILIPPO DE FILIPPI

34 16 VIA FILIPPO DE FILIPPI

1 28 CORSO BUENOS AIRES

26 29 VIA FRANCESCO REDI

29 21 VIA GIOVANNI BATTISTA MORGAGNI

5 30 VIA GIUSEPPE BROGGI

6 21 VIA GIOVANNI BATTISTA MORGAGNI

30 29 VIA GIOVANNI BATTISTA MORGAGNI

31 6 VIA GIOVANNI BATTISTA MORGAGNI

32 30 VIA GIOVANNI BATTISTA MORGAGNI

32 31 PIAZZALE LAVATER

33 32 PIAZZALE LAVATER

31 34 PIAZZALE LAVATER

2 35 VIA GIOVANNI OMBONI

35 33 PIAZZALE LAVATER

36 32 PIAZZALE LAVATER

38 39 PIAZZALE LAVATER

29 43 VIA GIOVANNI BATTISTA MORGAGNI

43 42 VIA GIOVANNI BATTISTA MORGAGNI

42 41 VIA GIOVANNI BATTISTA MORGAGNI

41 45 VIA GIOVANNI BATTISTA MORGAGNI

42 44 VIA GIOVANNI BATTISTA MORGAGNI

45 46 PIAZZALE BACONE

14 47 PIAZZALE BACONE

47 48 PIAZZALE BACONE

11 49 VIA FEDERICO OZANAM

3 50 VIA AMILCARE PONCHIELLI

50 49 PIAZZALE BACONE

52 46 VIA BARTOLOMEO EUSTACHI

41 53 VIA PLINIO

24 54 VIA BARTOLOMEO EUSTACHI

56 42 VIA PLINIO

58 24 VIA BARTOLOMEO EUSTACHI

25 59 VIA ACHILLE MAIOCCHI

27 60 CORSO BUENOS AIRES

63 19 VIA LAZZARO SPALLANZANI

21 64 VIA GIOVANNI BATTISTA MORGAGNI

64 41 VIA GIOVANNI BATTISTA MORGAGNI

B.2 Istanza reale

```
12 18 9.9512 450
28 19 0.7024 0
19 8 5.4856 250
33 5 16.017 700
5 26 6.0026 250
26 12 9.0988 400
30 6 11.929 550
35 20 7.5522 350
20 36 3.5458      150
37 36 6.372        300
20 37 0.5712 0
37 38 5.2988 250
39 31 0.6454 0
39 40 2.719        100
40 34 0.598        0
45 47 0.8968 0
46 47 0.8632 0
49 14 0.7706 0
50 14 0.7466 0
49 44 0.6718 0
44 14 4.8754 200
14 51 2.0166 0
48 51 9.3988 400
10 52 0.6528 0
53 10 0.647        0
53 52 0.597  0
54 10 0.6168 0
55 11 0.8726 0
55 56 1.287 0
60 56 9.2532 400
57 56 0.5422 0
9 58 0.3812 0
59 9 2.563 100
61 56 0.9094 0
```

```
48 62 0.1794 0
62 51 9.2962 400
2 63 11.849 550
1 2 9.0996 400
3 4 18.015 800
7 8 13.1314 600
11 3 15.657 650
12 13 29.6168 1400
16 15 16.574 750
40 17 25.5936 1200
13 7 25.0998 1200
6 16 15.0812 700
4 51 54.8102 2550
21 22 21.08         1000
23 24 33.2344 1550
22 23 31.5576 1500
15 23 16.9476 800
17 15 40.62 1900
17 25 27.9718 1300
13 57 23.4404 1100
18 57 35.7736 1700
43 18 15.0174  700
28 27 14.1702  650
8 5 31.668 1500
7 26 32.6418 1550
27 7 17.2074 800
16 22 15.971 750
34 16 25.0094 1150
1 28 43.4536 2050
26 29 31.05   1500
29 21 14.9396 700
5 30 13.0534 600
6 21 14.9712 700
30 29 17.971 850
31 6 34.0534 1600
```

```
32 30 36.033 1700
32 31 17.9576 850
33 32 15.047 700
31 34 19.0562 900
2 35 26.8678 1250
35 33 9.6152 450
36 32 8.2512 400
38 39 11.8178 550
29 43 38.8052 1850
43 42 17.1448 800
42 41 6.9246 300
41 45 45.9674 2150
42 44 53.9094 2550
45 46 9.7726 450
14 47 20.2114 950
47 48 33.9126 1600
11 49 37.0902 1650
3 50 45.1434 2100
50 49 10.6592 500
52 46 33.2888 1500
41 53 34.312 1600
24 54 47.7412 2250
56 42 49.7448 2350
58 24 16.3042 750
25 59 36.0158 1700
27 60 29.057 1300
63 19 26.6532 1250
21 64 21.538 1000
64 41 21.36  1000
1 2 9.0996 400
3 4 18.015 800
7 8 13.1314 600
11 3 15.657 650
12 13 29.6168 1400
16 15 16.574 750
```

```
40 17 25.5936 1200
13 7 25.0998 1200
6 16 15.0812 700
4 51 54.8102 2550
21 22 21.08        1000
23 24 33.2344 1550
22 23 31.5576 1500
15 23 16.9476 800
17 15 40.62   1900
17 25 27.9718 1300
13 57 23.4404 1100
18 57 35.7736 1700
43 18 15.0174 700
28 27 14.1702 650
8 5 31.668 1500
7 26 32.6418 1550
27 7 17.2074 800
16 22 15.971 750
34 16 25.0094 1150
1 28 43.4536 2050
26 29 31.05   1500
29 21 14.9396 700
5 30 13.0534 600
6 21 14.9712 700
30 29 17.971 850
31 6 34.0534 1600
32 30 36.033 1700
32 31 17.9576 850
33 32 15.047 700
31 34 19.0562 900
2 35 26.8678 1250
35 33 9.6152 450
36 32 8.2512 400
38 39 11.8178 550
29 43 38.8052 1850
```

```
43 42 17.1448 800
42 41 6.9246 300
41 45 45.9674 2150
42 44 53.9094 2550
45 46 9.7726 450
14 47 20.2114 950
47 48 33.9126 1600
11 49 37.0902 1650
3 50 45.1434 2100
50 49 10.6592 500
52 46 33.2888 1500
41 53 34.312 1600
24 54 47.7412 2250
56 42 49.7448 2350
58 24 16.3042 750
25 59 36.0158 1700
27 60 29.057 1300
63 19 26.6532 1250
21 64 21.538 1000
64 41 21.36   1000
```

B.3 Istanza reale: percorsi soluzione mattino

```
Percorso primo controllore
12 18 9.9512 450 0
18 43 15.0174 700 9.9512
43 42 17.1448 800 24.9686
42 56 49.7448 2350 42.1134
56 42 49.7448 2350 91.8582
42 43 17.1448 800 141.603
43 18 15.0174 700 158.7478
18 12 9.9512 450 173.7652
Percorso secondo controllore
12 18 9.9512 450 0
18 57 35.7736 1700 9.9512
```

```
57 13 23.4404 1100 45.7248
13 57 23.4404 1100 69.1652
57 18 35.7736 1700 92.6056
18 57 35.7736 1700 128.3792
57 18 35.7736 1700 164.1528
18 43 15.0174 700 199.9264
43 18 15.0174 700 214.9438
18 12 9.9512 450 229.9612
Percorso terzo controllore
12 18 9.9512 450 0
18 43 15.0174 700 9.9512
43 29 38.8052 1850 24.9686
29 26 31.05 1500 63.7738
26 7 32.6418 1550 94.8238
7 13 25.0998 1200 127.4656
13 57 23.4404 1100 152.5654
57 18 35.7736 1700 176.0058
18 12 9.9512 450 211.7794
```

B.4 Istanza reale: percorsi soluzione pomeriggio

```
Percorso primo controllore
12 18 9.9512 450 0
18 57 35.7736 1700 9.9512
57 18 35.7736 1700 45.7248
18 57 35.7736 1700 81.4984
57 13 23.4404 1100 117.272
13 57 23.4404 1100 140.7124
57 13 23.4404 1100 164.1528
13 12 29.6168 1400 187.5932
12 18 9.9512 450 217.21
Percorso secondo controllore
12 18 9.9512 450 0
18 43 15.0174 700 9.9512
43 29 38.8052 1850 24.9686
```

```
29 30 17.971 850 63.7738
30 29 17.971 850 81.7448
29 26 31.05 1500 99.7158
26 29 31.05 1500 130.7658
29 43 38.8052 1850 161.8158
43 18 15.0174 700 200.621
18 12 9.9512 450 215.6384
Percorso terzo controllore
12 18 9.9512 450 0
18 12 9.9512 450 9.9512
12 13 29.6168 1400 19.9024
13 57 23.4404 1100 49.5192
57 18 35.7736 1700 72.9596
18 57 35.7736 1700 108.7332
57 18 35.7736 1700 144.5068
18 12 9.9512 450 180.2804
```

Bibliografia

[1] Maurizio Bruglieri, Alberto Colorni, Alessandro Lué. *The Parking Warden Tour Problem*. 4OR, 2009.

[2] Feng Chu, Nacima Labadi, Christian Prins. *A Scatter Search for the periodic capacitated arc routing problem*. European Journal of Operational Research 169, 2006.

[3] Federico Lia. *Ottimizzazione dei percorsi dei controllori della sosta irregolare. Applicazione ad un'area di Milano*. Tesi di Laurea Specialistica, a.a. 2008/2009, Politecnico di Milano.

[4] Maurizio Bruglieri, Alberto Colorni. *Ricerca Operativa*, Zanichelli, in corso di pubblicazione.

[5] Gianni Di Pillo. *Metodi di ottimizzazione per le decisioni*, Atti della scuola CIRO, Massan, Milano, 1994.

[6] Holland J. *Adaptation in Natural and Artificial Systems*, University of Michigan Press, 1976.

[7] Claudia Archetti, DominiqueFeillet, AlainHertz, M.GraziaSperanza. *The undirected capacitated arcrouting problem with profits*, Computers Operations Research 37, 2010.

[8] Richard A. Johnson, Dean W. Wichern. *Applied multivariate statistical analysis*, sesta edizione, Pearson Education Inc., Upper Saddle River, NJ, 2007.

[9] Harvey M. Deitel, Paul J. Deitel. *C Corso completo di programmazione*, terza edizione, Apogeo, Milano, 2007.

[10] F. S. Hiller, G. .J. Liebermann. *Introduzione alla Ricerca Operativa*, nona edizione, Franco Angeli s.r.l., Milano, 1999.

[11] M. Fischetti. *Lezioni di Ricerca Operativa*, seconda edizione, Edizioni Libreria Progetto, Padova, 1999.

[12] M. Dell'Amico. *120 Esercizi di Ricerca Operativa*, Pitagora Editrice, Bologna, 1996.

[13] E. Tarantino De Falco, A. Della Cioppa. *I meccanismi di mutazione negli Algoritmi Genetici: verso una migliore imitazione della natura*,Workshop italiano sulla Vita Artificiale, 5Ð6 Settembre 2003, Cosenza, Italia.

[14] Daniele Rampoldi *I meccanismi di mutazione negli Algoritmi Genetici: verso una migliore imitazione della natura*, tesi di Laurea Specialistica, aa. 2005/2006, Università degli studi di Padova.

[15] http://www.graphviz.org/

[16] http://www.r-project.org/

[17] http://disi.unitn.it/ montreso/asd/appunti/argomenti/22-pnp.pdf

[18] http://www.wikipedia.it

[19] Appunti del corso di Modelli per le decisioni del Prof. Alberto Colorni, http://corsi.metid.polimi.it/.

[20] Claudio Agostinelli., *Introduzione a R*, versione 0,3, Paodva, ottobre 2003.

[21] Fabio Frascati. *Formulario di statistica con R*, versione 1.3.1, Firenze, novembre 2005.

[22] Appunti del corso di Fondamenti di Ricerca Operativa del Prof. Edoardo Amaldi , http://home.dei.polimi.it/amaldi/FRO-Mi-10-11.html.

[23] Appunti del corso di Ottimizzazione del Prof. Edoardo Amaldi, http://home.dei.polimi.it/amaldi/OTT-10-11.shtml.

[24] http://www2.mokabyte.it/cms/article.run?articleId=VAK-3CL-R65-N6H$_7$$f000001_7$715235$_5$$e8b$$f$$c48http$: //$igraph.sourceforge.net$/