

ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

SCUOLA DI SCIENZE
Corso di Laurea in Ingegneria e Scienze Informatiche

L'algoritmo bionomico per il Traveling Salesman Problem

Relazione finale in
RICERCA OPERATIVA

Relatore:
Prof. Aristide Mingozzi

Presentata da:
Luca Accorsi

II sessione di laurea
Anno Accademico 2014 - 2015

Indice

Elenco delle figure	3
Elenco delle tabelle	4
1 Gli algoritmi genetici	7
1.1 Breve storia dei genetici	7
1.2 Dalla biologia all'informatica	8
1.3 L'algoritmo	10
1.3.1 La rappresentazione dell'informazione	10
1.3.2 La funzione di fitness	11
1.3.3 Strategie di selezione ed elitismo	12
1.3.4 Gli operatori	16
1.3.4.1 Il crossover ed il problema dell'ammissibilità	16
1.3.4.2 La mutazione	19
1.3.4.3 L'inversione	19
1.3.4.4 La maturazione	21
1.3.5 La definizione della nuova popolazione	21
1.4 Lo <i>schema</i> di una soluzione	21
1.5 Lo Schema Theorem	23
1.6 Perché un algoritmo genetico?	25
1.7 La programmazione genetica	26
2 Metodi euristici per il TSP	29
2.1 Breve introduzione al TSP	29
2.2 Algoritmi per la risoluzione del TSP	30
2.3 Gli euristici per il TSP	30

3	L'algoritmo bionomico di Christofides	38
3.1	Introduzione	39
3.2	L'algoritmo	40
3.3	La satellites list	41
3.4	L'algoritmo per il TSP	45
3.4.1	Un esempio di esecuzione	48
3.5	Crossover alternativi	53
3.5.1	Crossover alternativo #1: <i>2-opt</i> crossover	53
3.5.2	Crossover alternativo #2: <i>gen-based</i> crossover	54
3.6	Risultati	55
3.7	Conclusioni	61
	Bibliografia	62

Elenco delle figure

1.1	Operazione di crossover tra due cromosomi	9
1.2	Esempio di roulette in Roulette Wheel Selection	13
1.3	Uniform crossover	17
1.4	Rappresentazione grafica di due soluzioni del TSP	17
1.5	Soluzione non ammissibile del TSP	18
1.6	Spazi di ricerca	19
1.7	Rappresentazione grafica di alcune schemata	22
1.8	Un programma espresso in forma di albero.	26
1.9	Mutazione in un albero rappresentante un programma	27
1.10	Crossover tra due alberi rappresentanti programmi	28
2.1	Relazione tra risultati di euristici, lower bound e soluzione ottima	31
2.2	Una esecuzione di Nearest Neighbor	32
2.3	Rappresentazione grafica dell’algoritmo di Christofides.	33
2.4	Rappresentazione grafica dell’euristico basato sull’albero di costo minimo.	34
2.5	Una mossa di 2-opt	35
2.6	Le sette possibili mosse 3-ottimali	36
2.7	The crossing bridges	36
3.1	Insiemi indipendenti in un grafo.	39
3.2	Rappresentazione grafica di una satellites list.	42
3.3	Una mossa di 2-opt	42
3.4	Implementazione mosse 3-opt: circuito iniziale	43
3.5	Implementazione mosse 3-opt: prima mossa	44
3.6	Implementazione mosse 3-opt: seconda mossa	44
3.7	Implementazione mosse 3-opt: terza mossa	44
3.8	Implementazione mosse 3-opt: quarta mossa	45
3.9	Il processo di crossover standard nel BA	48
3.10	Esempio di crossover: parent set	50

3.11 Esempio di crossover: sequenza di esecuzione	51
3.12 Un grafo degli archi	54
3.13 Crossover alternativo #2	55

Elenco delle tabelle

1.1	Esempio di corrispondenza tra genotipo e fenotipo.	10
1.2	Percentuali di fitness in un esempio di Roulette Wheel Selection	12
3.1	Matrice degli archi di un parents set	54
3.2	Schema delle differenze tra gli algoritmi.	57
3.3	Valori medi delle cinque esecuzioni	58
3.4	Valori migliori delle cinque esecuzioni	59
3.5	Tempi medi delle cinque esecuzioni	60

Abstract

In questa tesi viene presentato un nuovo metaeuristico per la risoluzione del *Traveling Salesman Problem* (TSP) simmetrico. Tale metodo, detto algoritmo bionomico, è una variante dell'algoritmo genetico che usa un metodo innovativo di generazione del parents set. Nella tesi vengono proposti diversi metodi di crossover specifici per il TSP ma che possono essere facilmente estesi per altri problemi di ottimizzazione combinatoria. Tali metodi sono stati sperimentati su un insieme di problemi test, i risultati computazionali mostrano l'efficienza dei metodi proposti. In particolare uno dei metodi domina gli altri sia per la miglior qualità delle soluzioni prodotte che per il minor tempo di calcolo impiegato.

Ringraziamenti

Desidero ringraziare i miei genitori senza i quali non avrei potuto intraprendere e quindi terminare questo percorso.

Un ringraziamento molto speciale va alla mia ragazza, Daniela, per essermi stata vicino, da lontano, ed avermi pazientemente aspettato.

Un grazie è doveroso anche per Antony e Michele senza i quali questi tre anni sarebbero sicuramente stati molto meno divertenti.

Infine un ringraziamento al mio relatore, il prof. Aristide Mingozzi, per le idee, i suggerimenti, l'aiuto e la disponibilità e cortesia dimostrate.

Capitolo 1

Gli algoritmi genetici

Gli algoritmi genetici (GA *genetic algorithm*) appartengono alla categoria degli algoritmi metaeuristici. Un algoritmo euristico tenta di fornire una soluzione in tempi ragionevoli (polinomiali o controllabili dall'utente) ad uno specifico problema difficile senza dare alcuna garanzia che essa sia "buona", molto simile o molto diversa da quella ottima. Un metaeuristico mette a disposizione un approccio generale alla risoluzione di una famiglia di problemi piuttosto che ad uno specifico. Da un punto di vista più scientifico un GA può essere visto come una ricerca casuale intelligente all'interno dello spazio di tutte le possibili soluzioni ammissibili (detto anche *search space* o spazio di ricerca). I GA sono stati principalmente impiegati nell'ambito dell'intelligenza artificiale, game-playing e pattern recognition, ma anche nell'area dell'ottimizzazione combinatoria quando si tratta di risolvere problemi *NP-Hard* di notevoli dimensioni. I GA prendono spunto dalla natura, come verrà illustrato nel paragrafo 1.2, fanno evolvere una popolazione iniziale di soluzioni, generata in modo casuale o output di altri euristici, combinano tra loro le soluzioni e producono nuove soluzioni che andranno a rimpiazzare parzialmente o totalmente le precedenti.

In questo capitolo è definito il GA e viene presentata una panoramica su le più rilevanti possibilità implementative dello stesso.

1.1 Breve storia dei genetici

Verso la fine degli anni 50 ci sono stati i primi tentativi atti a combinare tra loro l'informatica e la scienza dell'evoluzione. Nasce l'idea di applicare i processi evolutivi e di selezione naturale a problemi di ottimizzazione. I GA sono stati inizialmente sviluppati da **Bremermann (1958)**, nel 1962 sono poi stati resi famosi da **Holland (1962)** il quale

voleva studiare il fenomeno dell'adattamento, così come succede in natura, e sviluppare modi in cui questo potesse essere importato all'interno dei calcolatori. Un ulteriore passo avanti nella definizione più teorica di questo tipo di algoritmi si ha in **Holland (1975)** dove viene spiegato lo *Schema Theorem* (paragrafi 1.4 e 1.5) il quale definisce le condizioni di lavoro ideali per il successo di un GA. Secondo Holland i GA sono un metodo che permette di muoversi da una popolazione di *cromosomi* ad un'altra attraverso una qualche tipologia di *selezione naturale*. Nella sezione sottostante è introdotta la terminologia presa in prestito dalla biologia ed utilizzata nei GA. Essi sono in seguito stati utilizzati da **Bagley (1967)** nella realizzazione di un programma per il game-playing, da **Rosemberg (1967)** per la simulazione di processi biologici e da **Cavicchio (1972)** per la risoluzione di problemi di pattern-recognition. Da allora vi è stato un sempre maggior interesse nei GA e sono state sviluppate diverse varianti via via più efficaci e performanti. Si rimanda a **Dianati et al (2004)** per una trattazione più completa della storia dei GA. Oggigiorno i GA vengono impiegati con successo insieme a tecniche di ricerca locale nella risoluzione di complessi problemi di ottimizzazione combinatoria nei quali i metodi esatti sarebbero proibitivi a causa del tempo di calcolo richiesto. Prima dell'avvento dei GA ci sono stati diversi tentativi atti a risolvere problemi tramite algoritmi *ad hoc* che sfruttavano informazioni ottenute da campioni random all'interno dello spazio delle soluzioni (**Roberts and Flores (1966)** e **Nugent et al (1968)**). Queste tecniche svolgono effettivamente una parte di quello che un GA fa ma quest'ultimo, in più, fornisce un approccio flessibile e un framework robusto utilizzabile per la risoluzione di una grande varietà di problemi.

1.2 Dalla biologia all'informatica

L'aggettivo *genetico* nasce dall'analogia tra i componenti fondamentali e i comportamenti di questi algoritmi e la biologia vera e propria, sebbene quest'ultima sia infinitamente più complessa del modello qui discusso.

In un modello molto semplificato della biologia si può supporre che gli organismi siano composti da cellule contenenti un determinato insieme di *cromosomi*. Ognuno di questi codifica una stringa di DNA che racchiude informazioni essenziali dell'organismo stesso. Un cromosoma può essere suddiviso in *geni* ciascuno dei quali codifica un tratto, ad esempio il colore dei capelli o il gruppo sanguigno. I possibili valori di un tratto, capelli biondi piuttosto che neri, vengono detti *alleli*. Ogni gene possiede una posizione o *locus* all'interno del cromosoma. L'insieme dei cromosomi di un organismo forma il *genoma*. Il *genotipo* rappresenta il particolare insieme di geni contenuto in un genoma. Due indi-

vidui si dicono con lo stesso genotipo se hanno gli stessi genomi. Il genotipo dà origine al *fenotipo*, ovvero le caratteristiche fisiche visibili dell'individuo. Il processo che avviene durante la riproduzione è detto *crossover*, coppie di cromosomi vengono combinate tra loro in modo da formare uno o due figli (*offspring*). Questi figli sono poi soggetti a *mutazioni* casuali (*mutation*) nelle quali alcuni bit di DNA vengono alterati a causa, ad esempio, di errori durante la copia. Viene indicata come *funzione di fitness* una funzione che definisce un grado di bontà del cromosoma, per esempio il numero di figli che ha generato o la probabilità che esso ha di sopravvivere.

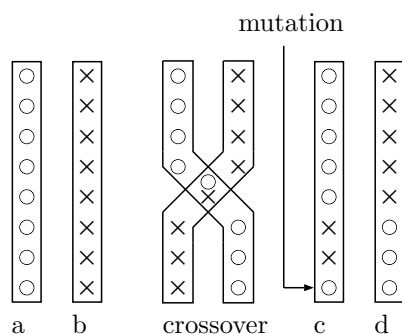


FIGURA 1.1 – L'operazione di crossover tra i due cromosomi a e b crea i figli c e d. La freccia in c indica un errore di copia.

I concetti della biologia sopra presentati in modo molto semplificato vengono mappati in quelli che sono gli elementi di un GA applicato ad un problema di ottimizzazione combinatoria. La *popolazione* (il genoma) è costituita da un insieme di soluzioni (i cromosomi), ciascuna delle quali è codificata come una stringa di bit (il DNA). A seconda della rappresentazione, che verrà successivamente illustrata, un gene può essere rappresentato da un insieme di uno o più bit di questa stringa. I possibili valori assumibili sono gli alleli e rappresentano il dominio del particolare gene all'interno del problema analizzato. Ad esempio nel TSP (paragrafo 2.1) un gene potrebbe essere la città in posizione t all'interno del tour¹, e gli alleli l'insieme di tutte le possibili città. Gli operatori di crossover e mutation possono essere implementati in maniera molto semplice rispettivamente come la ricombinazione di porzioni di stringa di due genitori diversi e la modifica di un valore scelto in modo casuale in una posizione (locus) anch'essa scelta in modo casuale (FIGURA 1.1). Il concetto di fenotipo non ha sempre un rispettivo immediato nel contesto dei GA. Un esempio potrebbe essere l'interpretazione decimale di una stringa di bit (TABELLA 1.1). La popolazione evolve un certo numero di iterazioni (o *generazioni*).

¹Circuito che attraversa ogni città una sola volta e termina in quella di partenza.

GENOTIPO	FENOTIPO
00010110	22

TABELLA 1.1 – Esempio di corrispondenza tra genotipo e fenotipo.

Ora che sono stati illustrati gli ingredienti base dei GA, verrà esposta la sequenza di passi in cui essi si combinano per formare l’algoritmo vero e proprio.

1.3 L’algoritmo

Le fasi svolte dai GA sono fondamentalmente quelle proposte nell’algoritmo generale sottostante, ciascuna delle quali verrà successivamente approfondita individualmente.

1. *Inizializzazione*: genera un insieme di soluzioni iniziali ammissibili in modo random o tramite un euristico.
2. *Loop*: finché non viene raggiunto il limite di generazioni:
 - (a) *Valutazione della fitness*: valuta la bontà di ciascuna soluzione tramite una determinata funzione di fitness.
 - (b) *Selezione dei genitori*: scegli in qualche modo, ad esempio utilizzando la fitness, coppie di soluzioni le quali formeranno i genitori.
 - (c) *Crossover*: utilizza i genitori sopra scelti per generare un determinato numero di figli, normalmente una o due nuove soluzioni per ogni coppia di genitori.
 - (d) *Mutation*: alcune delle nuove soluzioni vengono modificate in maniera casuale. In tal modo si cerca di simulare le variazioni aleatorie che avvengono in genetica al fine di diversificare gli individui.
 - (e) *Definizione della nuova popolazione*: sostituisci tutta o una parte della precedente popolazione utilizzando le nuove soluzioni create.
 - (f) *Torna a Loop*

1.3.1 La rappresentazione dell’informazione

La scelta della rappresentazione dei cromosomi, ovvero le soluzioni della popolazione, è cruciale per la buona riuscita dell’algoritmo. Essa è strettamente dipendente dal problema da risolvere. La prima rappresentazione in assoluto presentata da Holland è stata quella

binaria ovvero una stringa di 1 e 0. Su di essa Holland ha definito gli operatori di crossover, mutation e inversion *standard* e fondato la teoria dello *Schema Theorem*. Purtroppo questa rappresentazione non è adatta alla codifica di qualsiasi problema, pertanto grazie al maggior interesse per l'applicazione dei GA nei più svariati ambiti sono state proposte diverse altre rappresentazioni. Le più usate sono stringhe di caratteri, numeri floating-point, array di numeri, matrici, stringhe binarie in codifica Gray², ecc In un problema come il Knapsack³ una rappresentazione binaria del tipo stringa di bit 1 - 0, dove l'*i*-esimo gene ad 1 indica che l'oggetto *i* è stato preso e 0 che è stato lasciato, è più che adeguata. Nel TSP è necessario ampliare l'alfabeto dei caratteri a disposizione, supponendo che in ogni locus ci sia una città allora i possibili valori sono gli identificativi delle possibili città. Infine nella minimizzazione di una funzione ad una variabile, i cromosomi potrebbero essere i possibili valori del dominio di quella variabile, e quindi dei numeri reali. Per ogni dato problema si ha quindi una rappresentazione più adeguata rispetto ad altre. A seconda della scelta saranno da definire di conseguenza i metodi di crossover e mutation.

1.3.2 La funzione di fitness

La funzione di fitness è anch'essa strettamente legata al problema in questione. Un esempio numerico potrebbe essere la massimizzazione o minimizzazione di una generica funzione ad una variabile del tipo

$$f(x) = y + |\sin(32y)| \text{ dove } 0 \leq y < \pi$$

In questo esempio tratto da *An Introduction to Genetic Algorithms Mitchell (1996)* l'insieme delle soluzioni corrisponde a tutti i valori reali che *y* può assumere tra 0 e π escluso. Numeri come 2.18 o 1.697 sono soluzioni ammissibili di questo problema. In un esempio non numerico come il TSP la fitness di una soluzione *x* potrebbe essere espressa come un costo $c(x) = \frac{1}{\text{length}(x)}$ dove $\text{length}(x)$ è la lunghezza del tour mentre nel Knapsack

come un profitto $p = \sum_i^n (w_i x_i)$ dove w_i indica il valore dell'*i*-esimo oggetto e $x_i = 1$ se questo viene preso o 0 altrimenti. Non sempre è richiesto che una soluzione debba

²Il codice binario ha il seguente problema pratico: valori vicini potrebbero avere una codifica molto diversa. Si considerino ad esempio i valori 31, 32 e 0 mappati in una stringa di 6 bit. Essi sono rispettivamente 011111, 100000 e 000000. Come si nota 32 e 0 hanno un solo bit di differenza pur essendo molto distanti mentre 31 e 32 non hanno alcun bit in comune. Per approfondimenti si consulti **Caruana and Shaffer (1988)** e **Eshelman et al (1989)**.

³Dato uno zaino di capacità *Q* e *N* oggetti ciascuno con un peso w_i e un valore p_i , si devono mettere nello zaino gli oggetti che massimizzino il profitto senza superare la capacità dello stesso.

essere ammissibile ma talvolta ne si deve comunque valutare la fitness. In questo caso essa potrebbe essere calcolata come la combinazione di un costo c e di un grado di non ammissibilità g . Nei problemi reali è spesso complicato discriminare quali sono le caratteristiche che rendono una soluzione più *fit* rispetto ad un'altra e quindi definire una adeguata funzione di fitness.

1.3.3 Strategie di selezione ed elitismo

La selezione è il processo che consiste nello scegliere due soluzioni tra tutte le possibili all'interno della popolazione ed etichettarle come genitori per poi combinarle tra loro (operazione di crossover) e generare uno o due figli. Come si vedrà nel capitolo 3, quello che distingue un GA da un algoritmo bionomico è in primo luogo la quantità di genitori selezionati durante questa fase. L'obiettivo della selezione è quello di permettere la creazione di figli che ereditino dai genitori le migliori caratteristiche ottenendo quindi una fitness sempre migliore man mano che passano le generazioni.

*“Non necessariamente i due genitori migliori producono i figli migliori. Ovvero una popolazione molto diversificata è preferibile, come la storia insegna, di una composta di soli ariani selezionati.”*⁴

Di seguito sono riportati i metodi più utilizzati per la scelta dei genitori.

Roulette Wheel Selection

Il metodo funziona appunto come la ruota in una roulette nella quale le “fette” dove la pallina può fermarsi sono proporzionali in ampiezza alla fitness degli individui della popolazione. Si supponga di avere una popolazione di quattro individui x_1 , x_2 , x_3 e x_4 . Chiamiamo $p_f(x_i)$ la percentuale di fitness dell' i -esimo individuo. Una possibile situazione potrebbe essere la seguente.

x_1	x_2	x_3	x_4
5	45	25	25

TABELLA 1.2 – Fitness in percentuale in un dato istante.

⁴Dispense Prof. A. Mingozzi

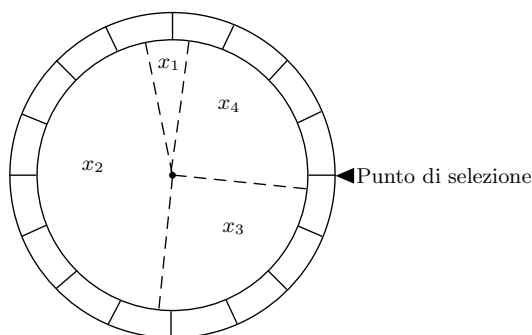


FIGURA 1.2 – La roulette corrispondente alle percentuali in TABELLA 1.2.

Come si può facilmente notare dalla FIGURA 1.2, è poco probabile che la soluzione x_1 venga scelta come genitore, mentre è molto probabile che x_2 venga selezionata anche più volte. Determinata la roulette con le sue fette si genera un numero random. L'individuo per il quale il numero generato risiede all'interno del proprio range viene selezionato come genitore, si ha pertanto che la probabilità per la quale un individuo x_i è scelto tra tutta la popolazione P è pari a $p_f(x_i) = \frac{f(x_i)}{\sum_{x \in P} f(x)}$. Sia l'*expected value* di x_i il rapporto $\frac{f(x_i)}{\sum_{x \in P} f(x)}$ dove n è il numero di elementi della popolazione, allora il metodo può essere implementato in questo modo.

1. Calcola *sum* come la somma degli *expected values* di tutti gli individui.
2. Ogni volta che la roulette viene azionata:
 - (a) Scegli un numero random r tra 0 e *sum*.
 - (b) Scorri gli individui della popolazione sommando il loro *expected value* finché *sum* non diventa maggiore o uguale ad r .
 - (c) La soluzione per la quale questa somma parziale supera o eguaglia *sum* è scelta come genitore.

I migliori individui hanno pertanto più probabilità di essere scelti. Questo replica i processi naturali nei quali gli individui più fit tendono di avere maggior probabilità di sopravvivere e raggiungere la fase di accoppiamento (*mating*). Il processo viene ripetuto finché non è stato generato il numero di genitori desiderato.

Una variante a questo approccio è lo scegliere un genitore mediante Roulette Wheel Selection e l'altro in modo casuale.

Sigma Scaling

Tipicamente la situazione iniziale in un GA è quella in cui si hanno pochi individui con buone caratteristiche e una grande varianza in termini di fitness all'interno della popolazione. Utilizzando metodi che selezionano i genitori in maniera proporzionale alla fitness è molto probabile che nelle prime generazioni vengano scelti sempre gli stessi. Questo porta ad una poca diversificazione, cioè ad una inefficace ricerca nello spazio delle soluzioni e ad una abbastanza prevedibile immobilizzazione in ottimi locali. Questa situazione è conosciuta come *convergenza prematura*. Per sopperire a questo problema sono stati proposti diversi metodi detti di *scaling* che non utilizzano direttamente la fitness *raw* dell'individuo ma considerano anche altri parametri *globali* come la fitness media e la varianza della popolazione. Uno di questi è il *Sigma Scaling*, dovuto a **Forrest (1985)** e **Goldberg (1989)**. La fitness viene mappata in un *expected value* nel seguente modo:

$$ExpVal(x_i, t) = \begin{cases} 1 + \frac{f(x_i) - \overline{f(t)}}{2\sigma(t)} & \text{se } \sigma(t) \neq 0 \\ 1 & \text{se } \sigma(t) = 0 \end{cases}$$

Dove $f(x_i)$ è la fitness di x_i , $\overline{f(t)}$ è la fitness media della popolazione al tempo t e $\sigma(t)$ la deviazione standard delle fitness della popolazione al tempo t .

Boltzmann Selection

Utilizzando lo Sigma Scaling si mantiene la *selection pressure*, per esempio il grado per il quale individui con fitness alta sono più abilitati ad avere figli, costante durante il corso dell'algorithm. Lo scopo di *Boltzmann Selection* è quello di variare questa pressione in base alle necessità. Ad esempio all'inizio dell'esecuzione si potrebbe voler lasciare che gli individui più fit vengano scelti con uno stesso rate di quelli meno fit ma verso la fine del GA concentrarsi solamente sui migliori. L'approccio utilizzato qui ricorda un po' quello del *Simulated Annealing* nel quale inizialmente sono permesse anche soluzioni che peggiorano quella corrente ma man mano che la temperatura scende aumenta la probabilità che vengano scelte solamente quelle migliori. Un'implementazione potrebbe essere

$$ExpVal(x_i, t) = \frac{e^{f(x_i)/T}}{\frac{\sum_{j=1}^n e^{f(x_j)/T}}{n}}$$

dove T rappresenta la temperatura e n la dimensione della popolazione. Per ulteriori approfondimenti si rimanda a **Goldberg (1990)**, **de la Maza and Tidor (1991)**, **de la Maza and Tidor (1993)** e **Prugel-Bennett and Shapiro (1994)**.

Rank Selection

In questa tecnica gli elementi della popolazione vengono classificati in base alla propria fitness. Il *linear ranking* inizialmente proposto da **Baker (1985)** ordina gli individui della popolazione per fitness crescente e calcola l'expected value servendosi del *ranking* piuttosto che della fitness. Utilizzando la posizione in classifica è possibile stabilire se una soluzione è più fit di un'altra senza però conoscere l'effettivo rapporto tra le due. Questo permette di evitare una convergenza prematura ma in alcuni casi può rallentare la ricerca di soluzioni con fitness alta appunto perché gli individui non vengono scelti proporzionalmente alla propria fitness.

Tournament Selection

Vengono scelti in modo casuale due individui dalla popolazione. Il migliore dei due in termini di fitness entra a far parte del set dei genitori mentre il secondo viene *squalificato*. Altre varianti includono la generazione di un numero random tra 0 ed 1, se il numero generato è compreso tra 0 e una certa soglia s (in genere $s > 0.5$) viene scelto il migliore in termini di fitness, altrimenti il peggiore. Le due soluzioni scelte non vengono eliminate dalla popolazione pertanto potrebbero essere selezionate più volte. Questo processo continua finché viene raggiunto il corretto numero di genitori. Dopodiché questi vengono accoppiati per procedere al crossover.

Stady-State Selection

I metodi finora analizzati selezionano un certo numero di genitori che verranno poi rimpiazzati da figli. Questo numero di genitori è la quasi totalità della popolazione. In *Steady-State Selection* viene utilizzato un approccio differente. Vengono selezionati come genitori solamente le soluzioni con fitness più elevata, vengono effettuate le operazioni di crossover e mutation e sostituito un piccolo sottoinsieme di soluzioni peggiori. Per approfondimenti **Syswerda (1989)**, **Syswerda (1991)**, **Whitley (1989)**, e **De Jong and Sarma (1993)**.

L'elitismo

I processi dei GA non garantiscono in alcun modo che i migliori individui sopravvivano alla generazione successiva. Per evitare che essi vengano persi durante il

crossover o che una mutazione possa peggiorarli si può utilizzare una tecnica introdotta da **De Jong (1975)** chiamata *elitismo*. Essa consiste nel copiare una piccola porzione delle migliori soluzioni direttamente nella generazione successiva senza alcuna modifica. Questa scelta permette di evitare che da una generazione all'altra il valore della miglior fitness diminuisca. Diverse ricerche hanno decretato un notevole miglioramento nelle performance dei GA a fronte dell'utilizzo di questa tecnica.

1.3.4 Gli operatori

Le operazioni fondamentali da implementare nella realizzazione di un GA sono il crossover e la mutation. Holland ha definito inizialmente anche una operazione di inversione (*inversion*) la quale però non ha avuto e non ha tuttora un così rilevante ruolo. Gli operatori sono stati originariamente sviluppati per stringhe di bit. I meccanismi per strutture semplici come gli array di numeri sono del tutto analoghi. In seguito si suppone di utilizzare una rappresentazione binaria delle soluzioni.

1.3.4.1 Il crossover ed il problema dell'ammissibilità

Negli anni sono state sviluppate diverse tecniche di crossover, la più semplice è senz'altro il *single-point crossover* (FIGURA 1.1). Dati due genitori x e y , si genera la posizione p di crossover in modo casuale, ad esempio $p = 3$, e si generano due figli z e w nei quali i bit dopo la posizione p sono scambiati tra i due genitori.

$$x = \mathbf{111001001}$$

$$y = 010100100$$

$$z = \mathbf{111}100100$$

$$w = 01\mathbf{0001001}$$

Si è notato però che in questo modo vengono favorite alcune posizioni del cromosoma, i segmenti scambiati contengono sempre gli *endpoint* della stringa. Per sopperire a questa parzialità posizionale si può utilizzare un *two-points crossover*, ovvero, vengono scelti p e q in modo random, con $p < q$ e vengono scambiati fra i genitori gli elementi tra i due punti. In questo modo le porzioni scambiate non contengono, necessariamente, gli estremi finali. Utilizzando i genitori precedenti e assumendo $p = 3$ e $q = 7$ si ottiene:

$$x = \mathbf{111001001}$$

$$y = 010100100$$

$$z = \mathbf{111100101}$$

$$w = 010\mathbf{001}000$$

Il concetto può in maniera analoga essere esteso ad un *n-points crossover*. Si ha infine uno *uniform crossover* nel quale ciascun gene del figlio viene scelto in maniera casuale tra i valori assunti da quello stesso gene nei due genitori (FIGURA 1.3).

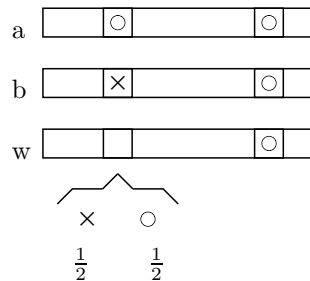


FIGURA 1.3 – Il primo gene valorizzato di w ha la stessa probabilità di assumere croce o cerchio mentre il secondo assumerà certamente cerchio.

Si supponga per un momento di uscire dal dominio delle stringhe binarie e passare alla rappresentazione di array di numeri. Siano x e y due soluzioni delle quali si vuol effettuare un single-point crossover e che rappresentano due circuiti del TSP.

$$x = (1, 2, 3, 4, 5, 0)$$

$$y = (1, 4, 3, 2, 5, 0)$$

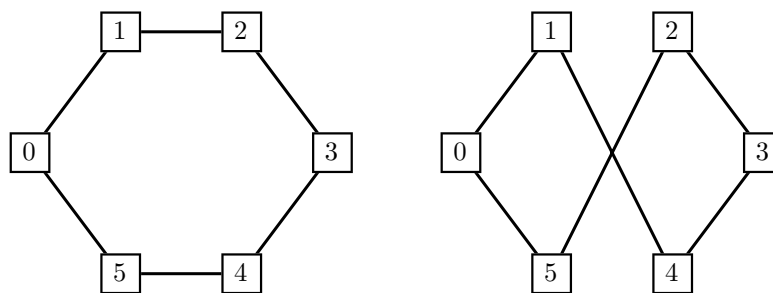


FIGURA 1.4 – Le due soluzioni, x a sinistra e y a destra.

Applicando questo crossover nel punto $p = 3$ si produrranno due figli tra cui il seguente z .

$$x = (1, 2, 3, 4, 5, 0)$$

$$y = (1, 4, 3, 2, 5, 0)$$

$$z = (1, 4, 3, 4, 5, 0)$$

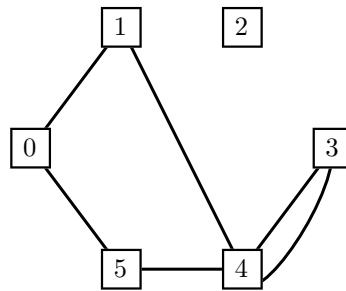


FIGURA 1.5 – La soluzione z , figlio di x e y .

Questa è chiaramente una soluzione non ammissibile. Un operatore di single-point crossover modificato che si può utilizzare per il TSP è il seguente⁵: siano i genitori x e y così definiti

$$x = (1, 0, 2, 3, 4, 5, 6)$$

$$y = (3, 2, 0, 1, 4, 6, 5)$$

e $p = 2$ il punto di crossover, il primo figlio viene generato copiando le prime p città dal primo genitore dopodiché il secondo genitore viene scandito dall'inizio e vengono selezionate tutte quelle città non ancora presenti nel figlio. Si procede analogamente per il secondo figlio scambiando il ruolo dei genitori. I due figli z e w prodotti sono:

$$z = (1, 0, 3, 2, 4, 6, 5)$$

$$w = (3, 2, 1, 0, 4, 5, 6)$$

In generale il risultato di un crossover può non produrre una soluzione ammissibile, è pertanto necessario tenerlo presente e, se richiesto dall'algoritmo utilizzato, renderla tale prima di continuare.

⁵Sono stati cambiati genitori e punto di crossover per realizzare un esempio più significativo e generale in quanto se applicata al precedente questa tecnica produce i figli uguali ai genitori.

1.3.4.2 La mutazione

La mutazione aiuta a mantenere un ragionevole livello di diversità tra i soggetti della popolazione, questo permette di uscire dalle regioni sub-ottimali (minimi o massimi locali) dello spazio delle soluzioni ed andare ad esplorare altri luoghi. Se per il crossover viene assegnata una probabilità di esecuzione P_c molto elevata, talvolta anche pari ad 1, la mutazione ha avuto finora un ruolo secondario, con una probabilità di esecuzione P_m molto bassa. Alcuni studiosi hanno rilevato che la potenza di quest'ultima è stata sottostimata (vedi **Muhlenbein (1992)**) e sarebbe opportuno sfruttarla per raggiungere zone dello spazio delle soluzioni altrimenti non facilmente e velocemente raggiungibili (FIGURA 1.6).

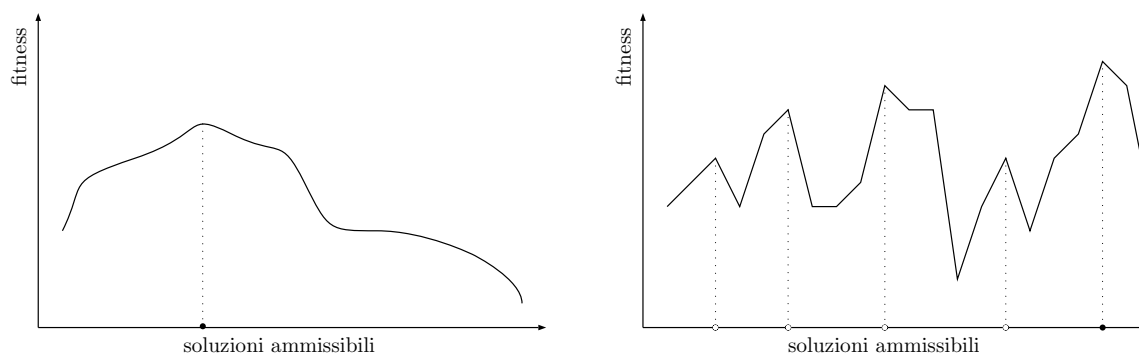


FIGURA 1.6 – Due diversi spazi di ricerca in un caso di massimizzazione. Il cerchietto nero rappresenta l'ottimo globale mentre quelli bianchi ottimi locali. A sinistra un esempio di problema semplice. Esso non presenta alcun ottimo locale, data una qualsiasi soluzione ammissibile controllando la sua neighborhood si riesce sempre a capire dove muoversi per andare verso l'ottimo. A destra un problema difficile con diversi ottimi locali nel quale per raggiungere l'ottimo potrebbe essere necessario passare per zone in cui la fitness è più bassa rispetto a dove si era precedentemente.

1.3.4.3 L'inversione

La versione più semplice dell'inversione consiste nel selezionare un sottoinsieme contiguo di geni, rimuoverlo dal cromosoma e reinserirlo invertito.

100110010

Effettuando l'inversione nella parte evidenziata si ottiene.

100100110

Nella genetica reale la posizione di un gene non influisce sulla funzione complessiva svolta dal cromosoma. L'inversione di una sequenza di geni pertanto non altererebbe il significato iniziale del cromosoma. La versione originale dell'operazione di inversione proposta da Holland si prefiggeva proprio l'obiettivo appena descritto, cioè muovere i geni senza modificare il valore della funzione di fitness. Per far ciò è necessario interpretare gli alleli indipendentemente dalla posizione nel cromosoma. Holland ha proposto di affiancare a ciascun gene un indice che ne indicasse la posizione *reale*. La fitness viene poi calcolata considerando le posizioni reali dei geni e non quelle fittizie dovute agli spostamenti. La soluzione

1101000101

viene memorizzata come

$((0,1)(1,1)(2,0)(3,1)(4,0)(5,0)(6,0)(7,1)(8,0)(9,1))$

dove il primo valore in ciascuna coppia di parentesi interne rappresenta la posizione reale del gene ed il secondo il valore assunto. In questo modo un'operazione di inversione dei geni nelle posizioni dalla 4 alla 8 genera

$((0,1)(1,1)(2,0)(3,1)(8,0)(7,1)(6,0)(5,0)(4,0)(9,1))$

e quindi la soluzione

1101010001

Sia la soluzione iniziale che quella invertita hanno la stessa fitness, quello che cambia sono i collegamenti con gli altri geni i quali influenzano il crossover. L'idea di Holland è di creare degli ordinamenti in cui alcuni *schemi* (si vedano i paragrafi 1.4 e 1.5) potessero sopravvivere con una probabilità più alta. Questa indicizzazione dei geni introduce però problemi nell'esecuzione del crossover. Non è detto che un figlio generato, ad esempio tramite single-point crossover, possieda tutte gli indici. Si supponga di combinare x e y selezionando la posizione di crossover $p = 3$ e creare il figlio z .

$x = ((0,1)(1,1)(2,0)(3,1)(4,0)(5,0)(6,0)(7,1)(8,0)(9,1))$

$y = ((0,1)(1,1)(2,0)(9,1)(8,0)(7,0)(6,0)(5,1)(4,0)(3,1))$

$z = ((0,1)(1,1)(2,0)(3,1)(8,0)(7,0)(6,0)(5,1)(4,0)(3,1))$

La soluzione (non ammissibile) z possiede due geni in posizione 3 e nessuno in posizione 9. Holland propone due tecniche per sopperire a questo problema: (1) permettere il crossover solo tra soluzioni che hanno lo stesso ordinamento e (2) selezionare uno dei due genitori ed applicare temporaneamente lo stesso ordinamento anche all'altro. In ogni caso l'inversione è stata presente soltanto nei primi studi di Holland ma in seguito abbandonata. Un'applicazione di questa tecnica è descritta in **Parson and Johnson (1995)**.

1.3.4.4 La maturazione

La maturazione consiste nel migliorare le caratteristiche di una data soluzione. Questo processo viene anche chiamato *ricerca locale*, in quanto si applicano modifiche locali che non interessano la popolazione nella sua totalità. Gli algoritmi di ricerca locale sono strettamente legati al problema che si ha da risolvere, alcuni esempi sono: WalkSAT per il Boolean Satisfiability Problem⁶ e 2-Opt per il TSP. È da tener presente che, le procedure di maturazione migliorano notevolmente le soluzioni, ma d'altra parte tendono a uniformare la popolazione con il rischio di impedire l'esplorazione di zone diverse dello spazio delle soluzioni.

1.3.5 La definizione della nuova popolazione

Una volta definiti i figli è necessario sostituire totalmente o parzialmente gli individui della precedente generazione. L'idea originale di Holland consiste nella sostituzione della popolazione *in blocco* con i figli generati da tale popolazione. Altre versioni suggeriscono di rimpiazzare un membro random della popolazione non appena viene generato un figlio. **De Jong (1975)** ha introdotto l'idea di un *gap* generazionale per evitare che genitori e figli si sovrappongano. Questo gap è stato supportato da studi empirici i quali indicano migliori performance quando diverse generazioni rimangono separate. I metodi più utilizzati in pratica sono la sostituzione totale, la sostituzione dei K peggiori individui con i K migliori figli oppure la sostituzione di K scelti casualmente con K anch'essi scelti in modo random. Come già detto una popolazione molto diversificata è preferibile, in quanto permette di esplorare uno spazio delle soluzioni più vasto.

1.4 Lo schema di una soluzione

Sebbene la maggior parte dei concetti utilizzati nei GA siano empirici e sperimentali, Holland sviluppa una analisi più teorica basata sul concetto di *schema*⁷ inteso con il significato di *forma* (*shape*). La trattazione originale di Holland utilizza cromosomi codificati in forma binaria. Si supponga di avere due cromosomi, ad esempio:

1010001

1111010

⁶Problema nel quale si deve determinare se esiste una interpretazione che renda vera una formula booleana.

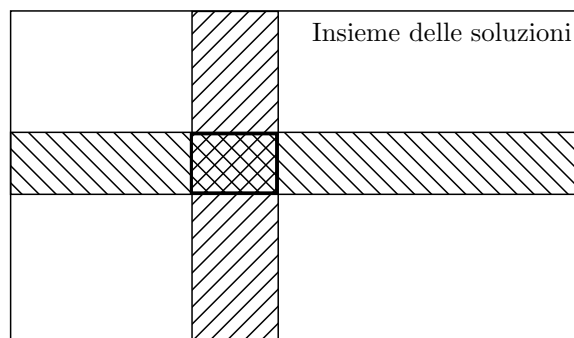
⁷da un verbo greco al tempo passato che significa *avere*.

essi, dice Holland, sono entrambi istanze degli schemi

$1*1*0**$

$1***0**$

(e di molti altri), dove * o *wild card* può essere sostituito da 0 o 1 (FIGURA 1.7).






-  Istanze dello schema $S1$
-  Istanze dello schema $S2$
-  Istanze di entrambe le schemata $S1$ e $S2$

FIGURA 1.7 – Rappresentazione grafica di alcune schemata tratta da *Adaptation in Natural and Artificial Systems* **Holland (1975)**.

Un modo per definire un insieme di cromosomi con caratteristiche simili è quello di definire dei sottoinsiemi di cromosomi che appartengono allo stesso schema. Come visto nell'esempio precedente un cromosoma appartiene a più *schemata* (plurale di schema). In generale si ha che data una stringa di DNA binaria di lunghezza n , ogni cromosoma è istanza di 2^n diverse schemata, questo perchè ogni posizione può assumere o il valore corrente o *. Ad esempio, sia $n = 2$ e si supponga che un particolare cromosoma abbia il seguente codice genetico 10. Esso sarà un'istanza delle quattro seguenti schemata: 10, $1*$, $*0$, $**$. In teoria una popolazione di M elementi può contenere fino a $M2^n$ schemata, in pratica però molte di queste occorreranno più volte sovrappendosi: alcune saranno presenti in più copie mentre altre saranno del tutto assenti. Ogni volta che si calcola la funzione di fitness di un dato cromosoma si stanno raccogliendo informazioni riguardanti la fitness media di ognuna schemata in cui la soluzione è un'istanza. Si può perciò affermare che si sta testando un gran numero di possibilità attraverso un'unica prova. Holland definisce *intrinsic parallelism* o *implicit parallelism* la proprietà dei GA di permettere

l'analisi di tanti concetti diversi in un colpo solo. È possibile dimostrare che processando una popolazione di M elementi in una iterazione, vengono analizzate $O(M^3)$ schemata. Si rimanda a **Reeves (1993)** per la dimostrazione. Holland ha osservato inoltre, come gli operatori (principali) dei GA, ovvero crossover e mutation, influiscono sul numero di istanze che rappresentano un determinato schema all'interno della popolazione. In particolare, lo *Schema Theorem* presentato nel prossimo paragrafo, predice che istanze di un determinato schema aumenteranno o diminuiranno in accordo con la sua fitness e indipendentemente da quelle delle altre schemata.

1.5 Lo Schema Theorem

Lo Schema Theorem permette di stimare la probabilità che un determinato schema ha di sopravvivere da una generazione all'altra. È necessario, però, prima introdurre alcuni concetti. Essi sono la *lunghezza* l , l'*ordine* k e la *fitness ratio* f di uno schema. La lunghezza indica la distanza tra il primo e l'ultimo elemento definito, ovvero che non è un wild card. L'ordine specifica il numero di elementi definiti. Si supponga di avere il seguente schema $S * 1 ** 0 * 1$, esso ha lunghezza $l(S) = 5$ e ordine $k(S) = 3$. La fitness ratio di uno schema S al tempo t è definita come il rapporto $f(S, t) = \frac{f(S)}{f(P)}$ con $\overline{f(S)}$ fitness media dello schema e $\overline{f(P)}$ fitness media della popolazione. I tre Lemma sotto descritti, permettono di capire quando uno schema viene *distrutto* e quali sono le migliori condizioni di lavoro di un GA.

Lemma 1 Selezione dei genitori

Se i genitori sono scelti in proporzione alla loro fitness, l'expected number di istanze di uno schema S all'iterazione $t + 1$ è dato da

$$E(S, t + 1) = f(S, t)N(S, t)$$

dove $f(S, t)$ è la fitness ratio per lo schema S e $N(S, t)$ è il numero di istanze di S al tempo t .

Dimostrazione

Il risultato è conseguenza immediata della definizione delle modalità riproduttive. La probabilità che uno schema sopravviva all'iterazione corrente dipende in primo luogo dal fatto che esso venga scelto come genitore. La situazione descritta è analoga alla modalità di selezione di Roulette Wheel Selection (paragrafo 1.3.3). Sia $f(S, t)$ la probabilità dell'evento "scelgo lo schema S " allora essa va sommata per ogni istanza dello schema all'interno della popolazione ovvero $N(S, t)$ volte.

Lemma 2 Crossover

Se il crossover è applicato al tempo t con probabilità P_c ad uno schema S di lunghezza $l(S)$, allora la probabilità che S verrà rappresentato nella prossima popolazione al tempo $t + 1$ è inferiormente limitata da

$$P[S, t + 1] \geq 1 - \frac{P_c l(S)}{n - 1} (1 - P[S, t])$$

dove n è la lunghezza del cromosoma.

Dimostrazione

La probabilità che lo schema S venga distrutto è data dalla probabilità dell'evento congiunto (1) "il crossover viene effettuato" P_c , (2) "il crossover avviene all'interno dello schema" $\frac{l(S)}{n - 1}$ e (3) "il partner non ha lo schema S " $1 - P[S, t]$. Sia infatti $P[S, t]$ la probabilità che il partner abbia lo schema S al tempo t allora $1 - P[S, t]$ è l'evento contrario. Pertanto la probabilità che lo schema S sopravviva al crossover è l'evento contrario tra la congiunzione di (1), (2) e (3). Esso definisce un *lower bound* in quanto S potrebbe essere generato all'iterazione t dalla combinazione di altre schemata e quindi esistere all'iterazione $t + 1$ indipendentemente da questa probabilità.

Lemma 3 Mutation

Se la mutazione è applicata all'iterazione t con probabilità P_m ad uno schema S di ordine $k(S)$, allora la probabilità che S sia rappresentata al tempo $t + 1$ è inferiormente limitata da

$$P[S, t] = 1 - P_m k(S)$$

Dimostrazione

Tutti i geni del cromosoma hanno la stessa probabilità di essere scelti per la mutazione (probabilità uniforme), la probabilità che venga quindi selezionato uno dei geni che interessa lo schema S è data dall'evento congiunto $P_m k(S)$ pertanto la probabilità che lo stesso rimanga inalterato è $1 - P_m k(S)$.

La combinazione di questi tre Lemma forma lo Schema Theorem.

Teorema Schema Theorem

Siano P_c e P_m rispettivamente le probabilità di crossover e mutation, S uno schema di ordine $k(S)$, lunghezza $l(S)$ e fitness ratio $f(S, t)$ al tempo t , allora l'expected number di istanze dello schema S all'iterazione $t + 1$ è dato da

$$E(S, t + 1) \geq \left\{ 1 - \frac{P_c l(S)}{n-1} (1 - P[S, t]) - P_m k(S) \right\} f(S, t) N(S, t)$$

Questa espressione mostra che mediamente i cromosomi che posseggono S aumenteranno a condizione che

$$f(S, t) \geq 1 + \frac{l(S)}{n-1} + P_m k(S)$$

Pertanto, come può anche sembrare intuitivo, schemi corti e di basso ordine saranno rappresentati da sempre più cromosomi mentre schemi lunghi con alto ordine tenderanno a scomparire. La situazione ideale di lavoro di un GA è quindi quella in cui si hanno piccoli schemi di basso ordine che combinati tra loro formano soluzioni sempre migliori. L'assunzione, empiricamente confermata, che in questa situazione un GA produca effettivamente buoni risultati è chiamata da **Goldberg (1989)** la *building-block hypothesis*.

1.6 Perché un algoritmo genetico?

Molti problemi reali consistono nella ricerca di una soluzione (ottima) all'interno di un insieme delle soluzioni davvero molto vasto nel quale una enumerazione completa richiederebbe un tempo inimmaginabile. Quello che questi problemi richiedono è una buona strategia per selezionare quali soluzioni valutare ed un parallelismo che permetta la simultanea valutazione delle stesse. I GA mettono a disposizione gli strumenti per assolvere questi compiti. Tramite gli operatori di crossover e mutation è possibile esplorare vaste

e distanti regioni dello spazio delle soluzioni, in più i GA vengono definiti *embarrassing parallel* ovvero il parallelismo è talmente evidente che non si ha da fare il minimo sforzo per separare il problema in un certo numero di compiti paralleli. La maggior parte delle operazioni da svolgere sono *indipendenti* ovvero non si hanno interazioni tra le soluzioni (se non durante alcune tipologie di crossover, ad esempio se si rimuovono i genitori dalla popolazione quando scelti, e durante la fase di sostituzione genitori-figli). La situazione ideale è quella in cui si ha una unità di calcolo per ciascun individuo così da processare un'intera popolazione al tempo di una singola soluzione. I GA vantano inoltre semplicità di realizzazione, gli operatori sono tanto semplici da implementare quanto efficaci e potenti nell'*aggredire* problemi complessi. Inoltre il framework generale messo a disposizione è *problem-independent* ovvero può essere utilizzato su una vasta gamma di problemi. L'aggiunta di eventuali algoritmi di ricerca locale implica però l'aggiunta di conoscenza *ad hoc* sul problema in esame.

1.7 La programmazione genetica

Nel settore dell'intelligenza artificiale vi sono stati diversi tentativi di programmazione automatica ovvero programmi che scrivono programmi. Uno di questi tentativi è dovuto a **Koza (1992)** che ha proposto la programmazione genetica (*GP genetic programming*) la quale consiste nell'applicazione delle operazioni classiche dei GA a popolazioni di programmi piuttosto che a semplici dati. La FIGURA 1.8 mostra come un programma possa essere espresso come un albero.

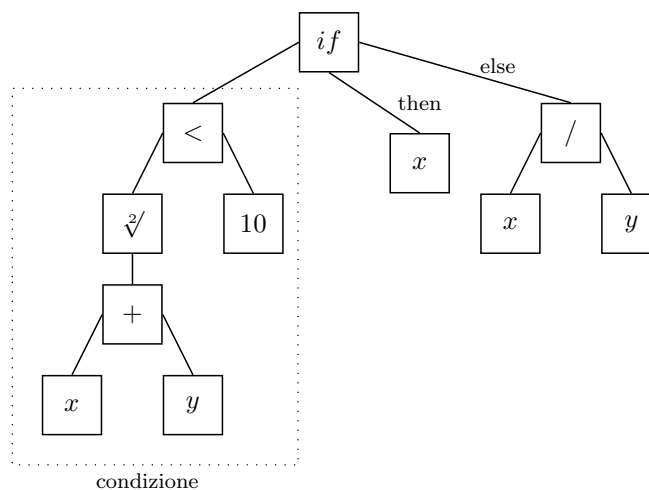


FIGURA 1.8 – Un programma espresso in forma di albero.

Koza utilizza il Lisp, un linguaggio di programmazione interpretato. Lo schema generale di un algoritmo di GP è simile a quello di un GA ed è il seguente:

1. *Inizializzazione*: genera una popolazione iniziale di programmi.

2. *Loop*: finché non viene raggiunto il limite di generazioni:
 - (a) *Valutazione della fitness*: esegui ciascun programma ed assegnagli un valore di fitness in accordo con quanto bene riesce a risolvere il problema.
 - (b) *Crossover e Mutation*: seleziona un certo numero di genitori ed applica gli operatori di crossover e mutation in modo da creare dei programmi figli (figure 1.9 e 1.10).
 - (c) *Definizione della nuova popolazione*: sostituisci tutta o una parte della precedente popolazione utilizzando le nuove soluzioni create.
 - (d) *Torna a Loop*

La miglior soluzione trovata è il risultato dell'algoritmo.

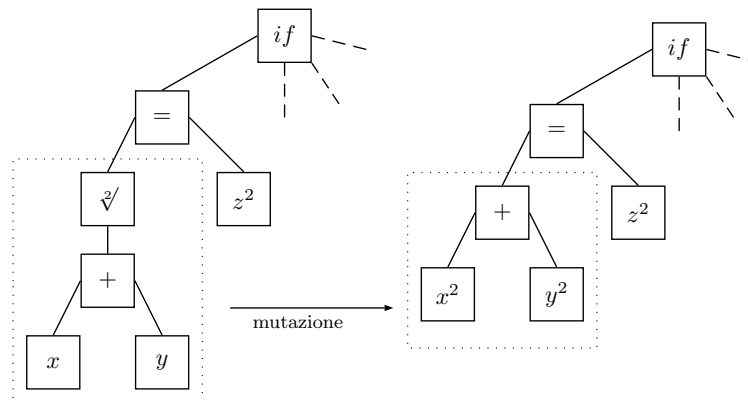


FIGURA 1.9 – Un esempio di mutazione.

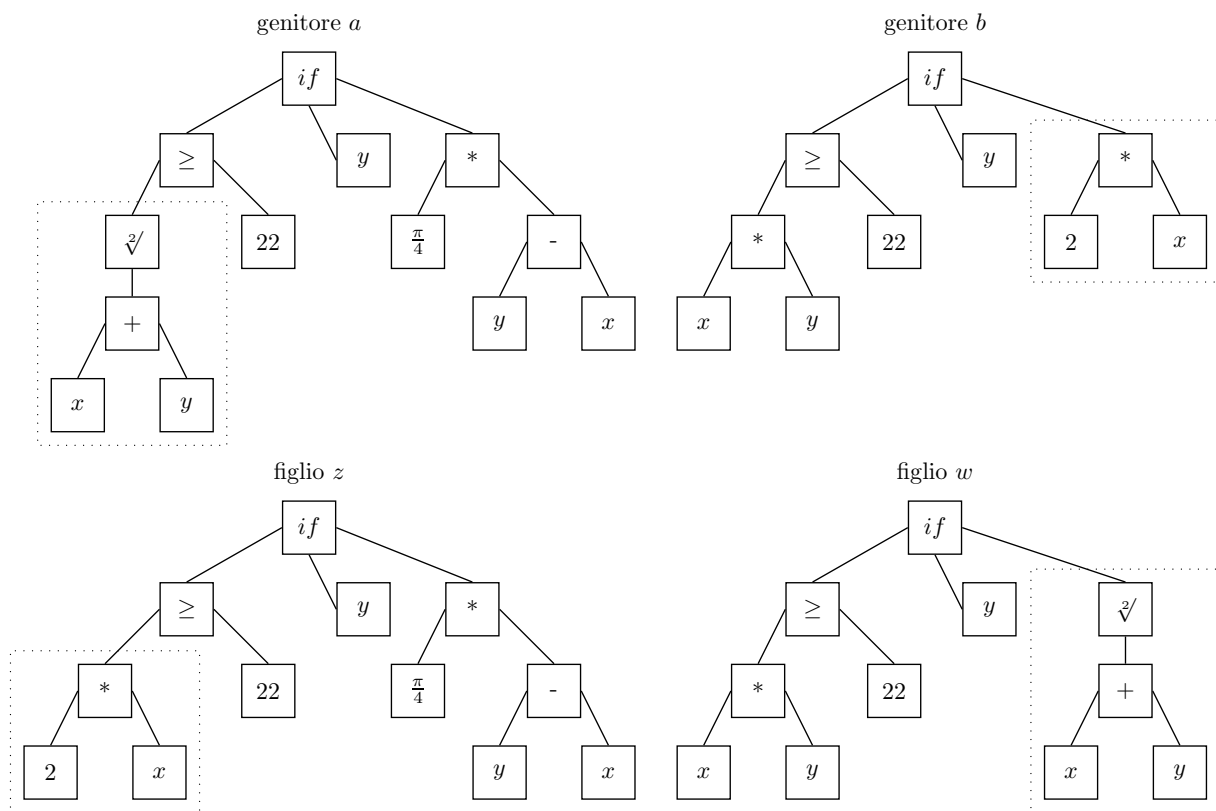


FIGURA 1.10 – un esempio di crossover.

Capitolo 2

Metodi euristici per il TSP

Per il problema del TSP esistono un'infinità di algoritmi di risoluzione. Esso è NP-Hard pertanto utilizzare algoritmi esatti per problemi di elevata dimensione risulta troppo dispendioso in termini di tempo.

In questo capitolo, dopo una breve introduzione al TSP verranno analizzati i principali metodi euristici di risoluzione.

2.1 Breve introduzione al TSP

Il TSP (*Travelling Salesman Problem*) o problema del Commesso Viaggiatore è uno dei più famosi problemi di ottimizzazione combinatoria. Esso è tanto semplice da formulare e definire quanto complesso da risolvere. Il problema consiste nel trovare un circuito che consenta ad un commesso viaggiatore di partire da una sorgente, attraversare un certo numero di città e tornare alla posizione di partenza in modo da visitare ciascuna città esattamente una volta e minimizzare la lunghezza del circuito stesso. Il TSP si dice simmetrico o STSP (*Symmetric TSP*) quando date due città A e B la distanza tra A e B è equivalente a quella tra B e A o asimmetrico o ATSP (*Asymmetric TSP*) quando questo non è vero. Il TSP è un modello di molti problemi reali, oltre a quello del commesso viaggiatore vero e proprio, come la perforazione di circuiti stampati, la cristallografia a raggi x, il routing di veicoli, ecc. Per ulteriori approfondimenti sulle applicazioni si consulti **Matai et al (2010)**.

2.2 Algoritmi per la risoluzione del TSP

Gli algoritmi per risolvere un problema di ottimizzazione combinatoria, in questo caso il TSP, possono essere suddivisi in due categorie:

Algoritmi esatti

Essi garantiscono di trovare la soluzione ottima ma richiedano molta potenza di calcolo e tempo di esecuzione. I metodi più utilizzati quando si cercano soluzioni intere sono i piani di taglio (*cutting plane*), *facet finding* e tecniche di *branch and bound*. Per approfondimenti **Laporte (1992)**.

Algoritmi euristici (o approssimati)

Essi permettono di generare *buone* soluzioni in tempi relativamente inferiori rispetto agli algoritmi esatti senza però garantire che l'ottimo venga effettivamente trovato. Alcuni di questi algoritmi restituiscono soluzioni che in media differiscono solo di una piccola percentuale rispetto alla soluzione ottima.

2.3 Gli euristici per il TSP

La valutazione di un algoritmo euristico si può effettuare fondamentalmente in base a due parametri: la velocità di esecuzione e la distanza delle soluzioni trovate rispetto a quella ottima. Se non si possiede la soluzione ottima ad un dato problema ma si vuole comunque stimare la bontà della soluzione generata si può utilizzare, nel caso di minimizzazione, un algoritmo per il calcolo di un *lower bound* cioè un numero che stima il valore della soluzione ottima senza però restituire una soluzione ammissibile. La FIGURA 2.1 mostra la relazione tra il risultato di un euristico, la soluzione ottima ed il lower bound.

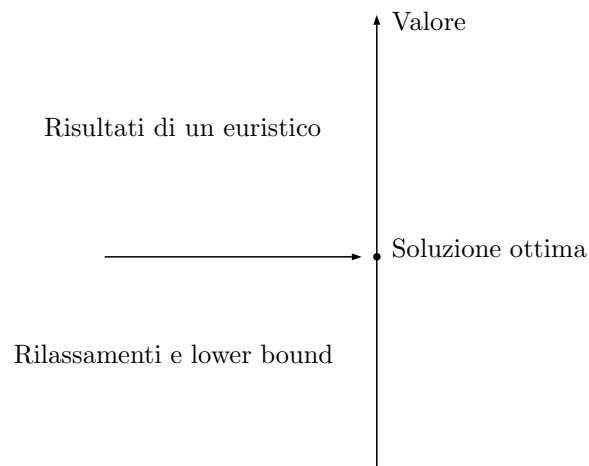


FIGURA 2.1 – In questa immagine tratta da *Modern Heuristic Techniques for Combinatorial Problems* (Reeves (1993)) è illustrata la relazione tra le possibili soluzioni prodotte da un algoritmo euristico, quelle prodotte da un lower bound e l'ottimo in un caso di minimizzazione.

Gli euristici per il problema del Commesso Viaggiatore possono essere ulteriormente raggruppati in due categorie:

Algoritmi di costruzione del tour

Essi generano iterativamente un tour aggiungendo in qualche modo una nuova città ad ogni passo. Tra di essi i più rilevanti sono:

Nearest Neighbor

L'idea di questo semplice algoritmo è quella di selezionare una città di partenza casuale dopodiché definire il tour spostandosi di volta in volta nella città più vicina non ancora visitata. Questo approccio *greedy* porta in genere a lunghe distanze nella fase finale dell'algoritmo. Il costo computazionale è $O(n^2)$ ed in FIGURA 2.2 è presente un esempio di esecuzione.

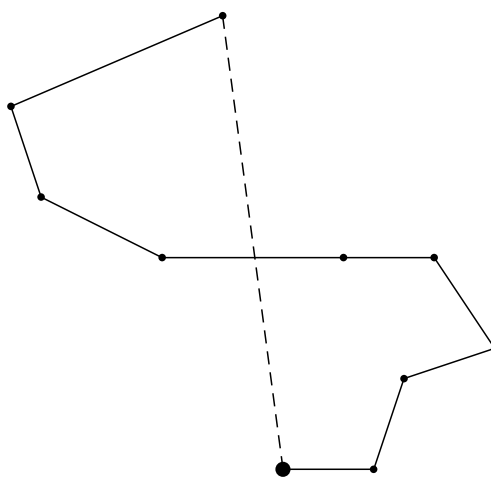


FIGURA 2.2 – Una esecuzione di Nearest Neighbor. La città iniziale è individuata dal cerchietto più grande mentre l'arco tratteggiato è l'ultimo inserito. Si nota come questo approccio non tenga conto della notevole distanza che si presenta dall'ultima città a quella di partenza.

Greedy

Questo euristico consiste nell'ordinare tutti gli archi del grafo associato alle città e aggiungere iterativamente al tour quelli che hanno un costo minore a patto che non venga creato un ciclo con meno di n archi (supponendo n sia il numero di città) e che ciascun nodo abbia al più grado 2. Questo algoritmo restituisce, in genere, risultati migliori di Nearest Neighbor ed ha un costo computazionale di $O(n^2 \log_2(n))$.

Euristico di Christofides

Dato un problema definito su un grafo completo nel quale la matrice dei costi degli archi soddisfa la disuguaglianza triangolare, ovvero $c_{ik} \leq c_{ij} + c_{jk} \forall i, j, k$ Christofides dimostra che è possibile costruire un tour che è al massimo $\frac{3}{2}$ l'ottimo. L'algoritmo è il seguente:

1. Costruisci un albero di costo minimo utilizzando l'insieme delle città.
2. Calcola il matching¹ di costo minimo per l'insieme dei nodi di grado

¹Dato un insieme di cardinalità pari di nodi, si definisce *matching* una collezione di archi tale che ogni nodo è il terminale di uno e un solo arco. Supponendo che ciascun arco abbia un costo, un matching si dice di costo minimo se la somma dei costi degli archi è minima.

- dispari².
3. Unisci gli archi dell'albero di costo minimo e gli archi del matching. Il grafo risultante è euleriano.
 4. La soluzione del TSP si ottiene attraversando il circuito euleriano evitando i nodi già visitati.

Lo step (2) è quello che ha il maggior costo computazionale cioè $O(n^3)$ e determina quindi il costo complessivo dell'algoritmo. La FIGURA 2.3 mostra un esempio di esecuzione.

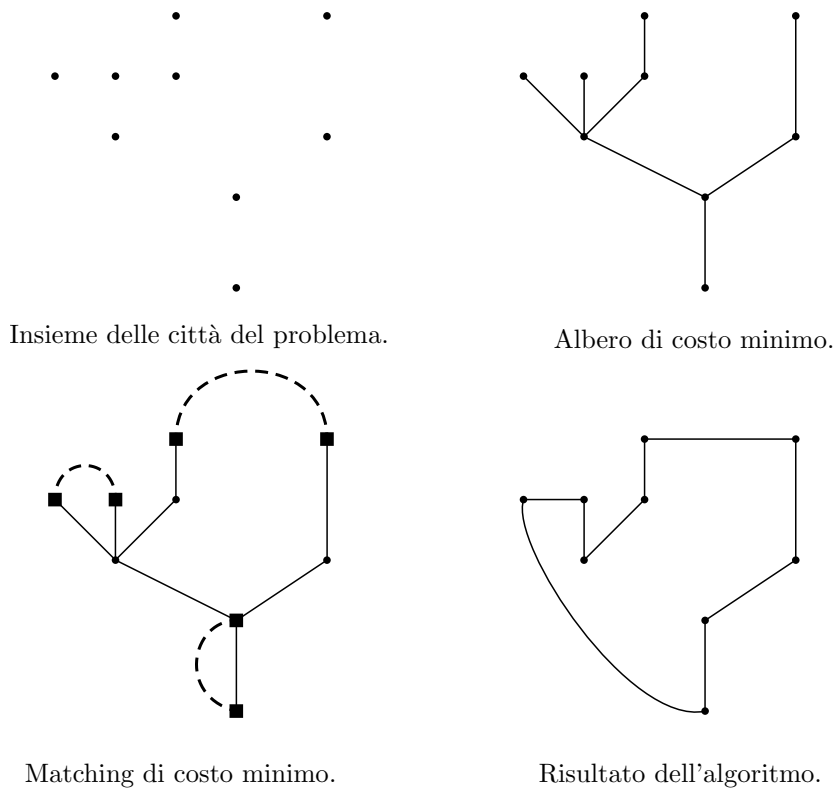


FIGURA 2.3 – Rappresentazione grafica dell'algoritmo di Christofides.

Euristico basato sull'albero di costo minimo

L'algoritmo di Christofides sopra presentato è l'evoluzione di quello che viene

²L'insieme dei nodi di grado dispari in un grafo è pari.

chiamato *euristico basato sull'albero di costo minimo*. Esso ha un *worst case* di 2 volte l'ottimo, un costo computazionale di $O(n^2 \log_2(n))$ ed è definito come segue:

1. Costruisci un albero di costo minimo utilizzando l'insieme delle città.
2. Duplica ogni arco ottenendo un circuito euleriano.
3. La soluzione del TSP si ottiene attraversando il circuito evitando i nodi già visitati.

La FIGURA 2.4 rappresenta graficamente l'algoritmo.

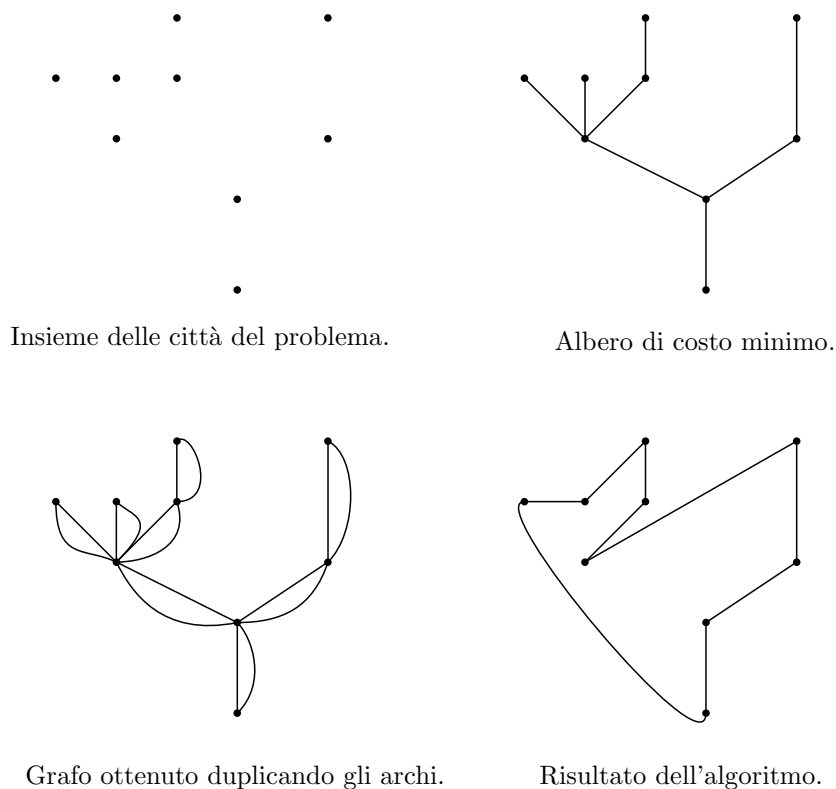


FIGURA 2.4 – Rappresentazione grafica dell'euristico basato sull'albero di costo minimo.

Algoritmi per il miglioramento di una soluzione ammissibile

Gli algoritmi sopra descritti terminano non appena è stato generato il tour. Quelli presenti in questa sezione ne trattano invece il miglioramento. Questa tipologia

ha avuto più successo rispetto alla precedente sicuramente anche a causa della maggior efficacia nel trovare tour di qualità superiore. Sono descritti ora alcuni dei più rilevanti algoritmi.

2-opt, 3-opt e k-opt

Gli algoritmi 2-opt e 3-opt sono forse le procedure di ricerca locale più comunemente utilizzate per il TSP. Il 2-opt consiste nel rimuovere due archi e ricollegarli nell'unico modo possibile (per ottenere un tour ammissibile) se la lunghezza totale del nuovo circuito risulta inferiore. Questa operazione può essere ripetuta per ogni coppia di archi per un costo computazionale di $O(n^2)$. Un tour si dice *2-ottimale* se non è possibile effettuare uno scambio di due archi che permetta di migliorare il circuito. La FIGURA 2.5 mostra una mossa che rende 2 ottimale una soluzione iniziale.

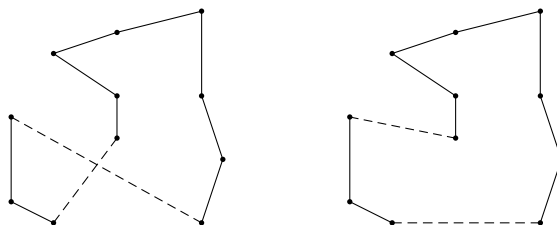


FIGURA 2.5 – Una mossa di 2-opt che riduce la lunghezza totale del circuito. A sinistra la soluzione di partenza e a destra la soluzione ottenuta.

L'algoritmo 3-opt di costo $O(n^3)$ lavora in modo analogo su 3 archi per verificare se si ha un miglioramento del tour in uno dei 7 modi possibili di collegamento. La FIGURA 2.6 mostra le possibili mosse di 3-opt. Siccome 3-opt contiene le mosse di 2-opt se un tour è *3-ottimale* è senz'altro anche 2-ottimale.

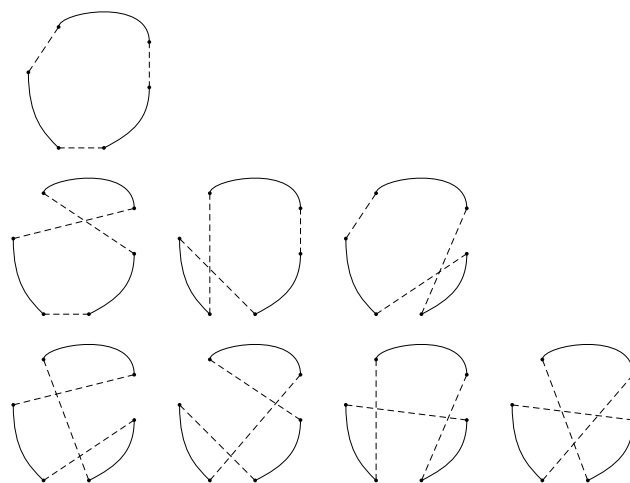


FIGURA 2.6 – Le possibili mosse 3-ottimali. La prima riga contiene il tour iniziale, la seconda le tre mosse 2-ottimali presenti all'interno di 3-opt mentre l'ultima le mosse 3-ottimali vere e proprie.

É possibile generalizzare il numero di archi da considerare. L'algoritmo k -ottimale consiste appunto nel rimuovere k archi per un costo di $O(n^k)$. La FIGURA 2.7 mostra una mossa di 4-opt chiamata *the crossing bridges*.

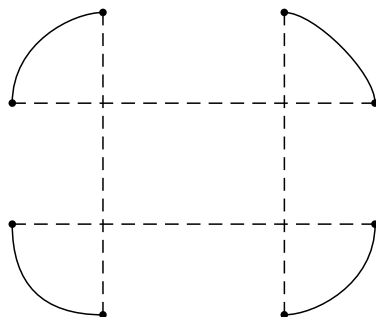


FIGURA 2.7 – *The crossing bridges*. Le mosse finora illustrate possono essere definite come un insieme di mosse di 2-opt. Questa ha invece la particolarità di non poter essere effettuata come una sequenza di mosse di 2-opt legali.

Dato l'aumento del costo computazionale al crescere di k in pratica si applica al più il 3-ottimale.

Algoritmo di Lin-Kernighan

Questo è uno dei più sofisticati ed efficaci algoritmi euristici per il TSP. Esso consiste nell'applicare iterativamente ad una soluzione gli algoritmi k -ottimali sopra descritti decidendo ad ogni iterazione il k più adatto (anche 5-ottimali). Per l'articolo originale si consulti **Lin and Kernighan (1973)**. L'implementazione di questo algoritmo è tutt'altro che banale, **Helsgam (1998)** fornisce un'implementazione efficiente. L'algoritmo ha un costo computazionale di $O(n^{2.2})$, il che lo rende leggermente più lento di un normale 2-opt ma le soluzioni prodotte sono notevolmente migliori.

Incrementare la velocità di 2-opt e 3-opt

La complessità degli algoritmi k -ottimali risiede nel dover analizzare per tutte le possibili mosse se si ha un miglioramento del tour. È possibile velocizzare questa ricerca mantenendo una lista statica di vicini per ciascuna città in modo da effettuare confronti solamente con gli archi relativi ad essi. Questo causa un'ulteriore occupazione di memoria ma consente di restringere la ricerca dei possibili archi da scambiare. L'incremento di velocità invalida però la garanzia di 2 o 3 ottimalità ma scegliendo la quantità di vicini da considerare in modo accurato è possibile accelerare notevolmente l'algoritmo senza diminuire troppo la qualità del tour.

I GA ed altre tecniche metaeuristiche come Tabu Search, Simulated Annealing, Ant Colony Optimization, ecc... possono essere classificate come algoritmi per il miglioramento del tour.

Held-Karp lower bound

Un valido lower bound, con cui confrontare la soluzione ottenuta da un euristico, è descritto da **Held and Karp (1970)**. Questo lower bound corrisponde all'ottimo del rilassamento lineare della classica formulazione, detta a due indici, del TSP dove vengono rilassati i vincoli di interezza. La soluzione può essere calcolata con il metodo del Simplex se il numero di città è inferiore a 100. Per problemi di più grande dimensione è stato sviluppato un approccio iterativo, detto del rilassamento lagrangiano, che risulta più efficace del Simplex.

Capitolo 3

L'algoritmo bionomico di Christofides

L'algoritmo bionomico (BA *bionomic algorithm*) è stato proposto da Christofides e presentato alla conferenza di Savona del 1994 **Christofides (1994)**. Non molto altro materiale è stato pubblicato sull'argomento. Gli articoli più rilevanti riguardano la risoluzione del *Capacitated p -Median Problem*¹ (CPMP) nel quale viene provata l'efficacia di questo quando affiancato a buone tecniche di ottimizzazione locale e confrontata con i migliori euristici finora esistenti (*HEUMED*, *OC*, *MB1* e *MB2*). I risultati trovati mostrano che per alcuni insiemi di problemi definiti difficili il BA performa meglio degli altri euristici ed è l'unico a trovare la soluzione ottima. Per approfondimenti si rimanda all'articolo originale di **Maniezzo et al (1998)**. Nel 2000 Christofides ha utilizzato il BA per risolvere due problemi riguardanti l'ingegneria finanziaria: *corporate tax structuring* e *scenario selection*. Anche in questo caso il BA risulta superiore rispetto ad altre tecniche quali Simulated Annealing e GA **Christofides (2000)**. Un ulteriore risultato dell'applicazione del BA si ha nel *Shortest Path Finding Problem* **Beg et al (2011)**. In questo caso gli autori confrontano le prestazioni di questo algoritmo con quelle di un GA. Essi confermano l'abilità del BA nell'evitare ottimi locali e la sua rapidità di convergenza in termini di numero di iterazioni anche se a costo di un tempo computazionale superiore a causa della maggior complessità del BA.

In questo capitolo viene descritto l'algoritmo generale e l'adattamento utilizzato per la risoluzione del TSP. Ad oggi, in letteratura, non è presente alcuna applicazione del BA al TSP.

¹locazione ottima di p strutture per servire n entità.

3.1 Introduzione

Il BA è una particolare variante del GA. Il BA fornisce un metodo di ottimizzazione globale che va adattato al particolare problema da risolvere. Il BA e il GA condividono la maggior parte degli step. Come per il GA, il BA agisce su una popolazione di soluzioni che ad ogni iterazione viene aggiornata. I figli vengono generati partendo da un insieme di genitori e su di essi possono agire alcune mutazioni casuali. Il BA al contrario del GA necessita obbligatoriamente di una qualche tecnica di ottimizzazione locale sulle soluzioni (la fase di maturazione descritta in 1.3.4.4). Questo approccio, ora praticato anche all'interno dei GA, è stato utilizzato raramente fino agli ultimi anni 80 mentre oggi è diventato lo standard. Le fasi per le quali un BA si differenzia da un GA sono essenzialmente (1) la selezione dei genitori e di conseguenza (2) il crossover. Data una popolazione P ad una certa generazione g , il BA definisce un grafo di similarità tra tutte le soluzioni appartenenti a P . Viene generato un grafo i cui vertici corrispondono alle soluzioni della popolazione. I vertici di due soluzioni vengono collegate da un arco se dette soluzioni sono considerate simili in base ad una qualche funzione di similarità. Costruito questo grafo si va ad identificare un insieme indipendente (massimale) di soluzioni (tramite un euristico), cioè si vanno a prendere tutte quelle soluzioni per le quali non si ha un arco che le collega (si veda la FIGURA 3.1). Queste vengono identificate come primo insieme dei genitori (*parents set*), esse verranno poi combinate per ottenere un certo numero di figli, il BA ammettono la possibilità di aver l'insieme delle soluzioni di dimensione variabile, ad esempio nel CPMP sopra accennato viene effettuato un mapping di n genitori in un unico figlio. Questa strategia introduce uno speciale criterio di diversificazione che permette di esplorare zone molto distanti dello spazio delle soluzioni ed evitare ottimi locali.

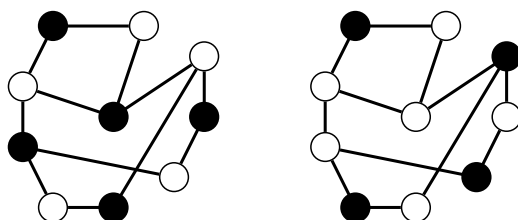


FIGURA 3.1 – Esempi di insiemi indipendenti in un grafo. In nero sono evidenziati i nodi appartenenti all'insieme indipendente. A sinistra un insieme indipendente di cardinalità massima, a destra un altro insieme indipendente nello stesso grafo.

3.2 L'algoritmo

La struttura generale del BA è la seguente:

sia $g = 1, \dots, g_{max}$ l'indice delle generazioni, $s = 1, \dots, s_g$ il numero delle soluzioni per ciascuna generazione, x_i^g l' i -esima soluzione nell'iterazione g rappresentata nel modo più adeguato al problema e infine $f(x_i^g)$ la valutazione della funzione di fitness della soluzione x_i^g .

1. Inizializzazione

- (a) Setta $g = 1$.
- (b) Scegli s_1 , il numero di soluzioni iniziali.
- (c) Crea l'insieme delle soluzioni, cioè s_1 soluzioni ammissibili (distinte) in maniera random o usando un euristico.
- (d) Sia quindi $X^1 = \{x_i^1 : i = 1, \dots, s_1\}$ l'insieme delle soluzioni iniziale.

2. Maturazione

- (a) Migliora ciascuna soluzione utilizzando un algoritmo di ricerca locale adatto al problema.
- (b) Sia x^* la migliore soluzione (ottimo locale) associata ad X^i alla corrente iterazione.
- (c) Sia $z(x^*)$ il costo di x^* .

3. Definizione dell'insieme dei genitori

- (a) Sia $\overline{X^g}$ l'insieme delle soluzioni maturato nello Step 2.
- (b) Definisci per ogni soluzione x_i^g una *frequenza di inclusione* θ_i^g la quale mappa la fitness della soluzione $f(x_i^g)$ in un numero intero positivo.
- (c) Per ogni soluzione calcola una funzione di similarità (distanza) con tutte le altre soluzioni.
- (d) Scegli una distanza Δ oltre la quale due soluzioni vengono definite diverse.
- (e) Definisci un grafo di adiacenza (o similarità) $G(X^g)$ in accordo con la sopra definita distanza. Due soluzioni sono simili e quindi collegate da un arco se la loro distanza non è maggiore di Δ .
- (f) Definisci l' r -esimo insieme di genitori P_r^g come un insieme indipendente (massimale) di $G(X^g)$.

- (g) Per ogni soluzione scelta, quindi $\forall x_i^g \in P_r^g$, aggiorna la frequenza di inclusione θ_i^g decrementandola di un valore intero adeguato (ad esempio 1 o il valore minimo attribuito). Se $\theta_i^g = 0$ rimuovi x_i^g da $G(X^g)$.
- (h) Ripeti (f) e (g) finché il grafo contiene elementi e non è completo.
- (i) Sia $P^g = \{P_r^g : r = 1, \dots, r_g\}$ l'insieme di tutti i parent sets.

4. Crossover, definizione dell'insieme dei figli

- (a) Sia $\pi(S, \varepsilon)$ una funzione che mappa un insieme $S \subseteq X^g$ in una soluzione $x \in X^{g+1}$ dove ε è un vettore random che affligge questo mapping *many to one*. La nuova generazione sarà quindi definita come un insieme $X^{g+1} = \{x | \pi(P_r^g, \varepsilon_j), r = 1, \dots, r_g, j = 1, \dots, \eta_r\}$, dove η_r è il numero di figli (offspring) del parent set r .

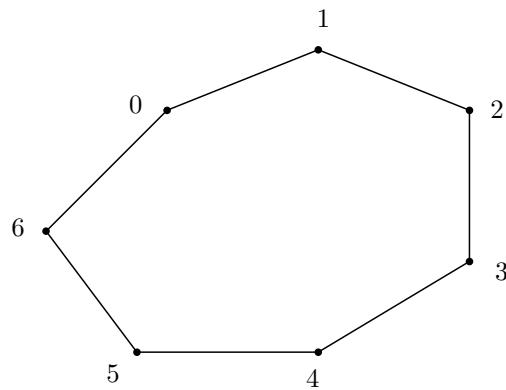
5. Terminazione

- (a) Ripeti gli Step dal 2 al 4 finché non è raggiunto il limite di generazioni.
- (b) Seleziona la miglior soluzione trovata.

I parametri utilizzati nell'algoritmo sono g_{max} , il numero di generazioni, s_1 , il numero di individui della prima generazione, Δ , la distanza utilizzata per la definizione del grafo di similarità ed η_r , il numero di offsprings di ogni parent set. È inoltre necessario definire la frequenza di inclusione che mappa $f(x_i^g)$ in θ_i^g e la funzione $\pi(S, \varepsilon)$ che definisce come ottenere i figli partendo dal set dei genitori. Nella fase di maturazione viene citato l'utilizzo di un algoritmo di ricerca locale, il quale è chiaramente *problem-dependent*.

3.3 La satellites list

La struttura dati utilizzata per la rappresentazione di ciascuna soluzione nell'algoritmo realizzato è la *satellites list* di **Osterman and Rego (2003)**, struttura definita *ad hoc* per il TSP simmetrico. Essa permette di effettuare operazioni di inversione (*reverse*) di un sottopercorso, le quali sono la base degli algoritmi k -ottimali, in tempo costante mantenendo una occupazione di memoria analoga alla *linked list*. Dato il seguente insieme di città 0,1,2,3,4,5,6 supponendo di generare un tour con il medesimo ordine, una satellites list è definita come nella FIGURA 3.2.



Indice satellite	0	1	2	3	4	5	6	7	8	9	10	11	12	13	
Array	2	13	4	1	6	3	8	5	10	7	12	9	0	11	
Città	N_0	P_0	┌───┐		┌───┐		┌───┐		┌───┐		┌───┐		┌───┐		
	└───┘	└───┘	1	2	3	4	5	6							
	0														

FIGURA 3.2 – Rappresentazione grafica di una satellites list. In posizione 0 si ha 2, esso indica che il satellite successivo N_0 ha indice 2. In posizione 1 si ha l'indice del satellite precedente P_0 . Dato un satellite con indice i , per capire in quale città ci si trova è sufficiente effettuare $\frac{i}{2}$ oppure $i \gg 2$ (right shift) se nel linguaggio usato sono disponibili le operazioni bitwise. Dato l'indice di N_0 , in questo caso lo 0, è possibile ricavare l'indice di P_0 (e viceversa) tramite $i + 1 - 2(i \bmod 2)$ oppure $i \wedge 1$ (XOR).

Una operazione 2-ottimale consiste nello scambiare gli archi $ab-cd$ con gli archi $ac-bd$. Questo implica l'inversione del sottopercorso bc (FIGURA 3.2).

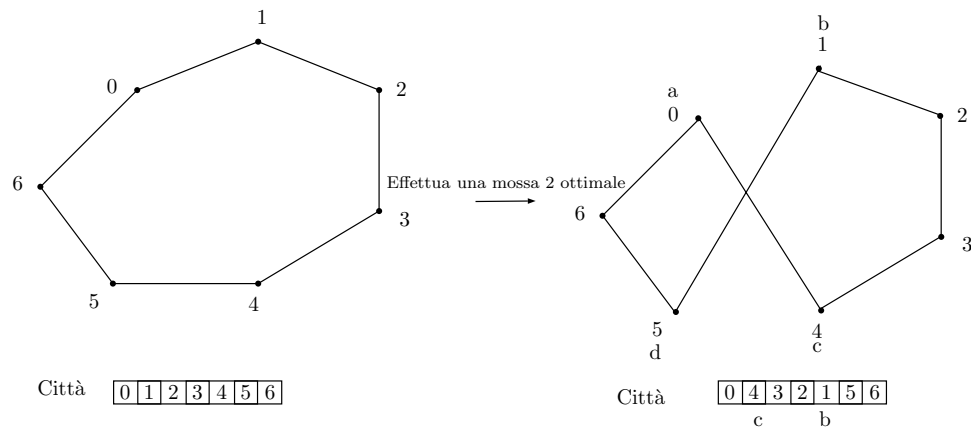


FIGURA 3.3 – Una mossa di 2-opt. Si noti l'inversione del sottopercorso bc .

Per effettuare questa operazione nella linked list è necessario scambiare tutti i puntatori alla successiva e precedente città all'interno del sottopercorso *bc*. Nella satellites list è sufficiente scambiare 4 puntatori (o indici), indipendentemente dal numero di città in *bc*. Il LISTATO 3.1 contiene l'implementazione degli autori.

LISTATO 3.1 – Implementazione mossa 2-ottimale da Osterman e Rego (2003)

```

1  /* Replaces arcs ab and cd with arcs ac and bd.
2     Implies the bc-path is reversed. */
3
4  void Reverse_Subpath(   long  satellite_index_a,
5                          long  satellite_index_b,
6                          long  satellite_index_c,
7                          long  satellite_index_d )
8  {
9      SATELLITES [satellite_index_a] = satellite_index_c^1;
10     SATELLITES [satellite_index_c] = satellite_index_a^1;
11     SATELLITES [satellite_index_d^1] = satellite_index_b;
12     SATELLITES [satellite_index_b^1] = satellite_index_d;
13     // There are no intermediate pointers to flip
14 }

```

Le mosse 3-ottimali, che non sono semplici mosse 2-ottimali, sono 4. Esse possono essere implementate come un insieme di scambi 2-ottimali come segue.

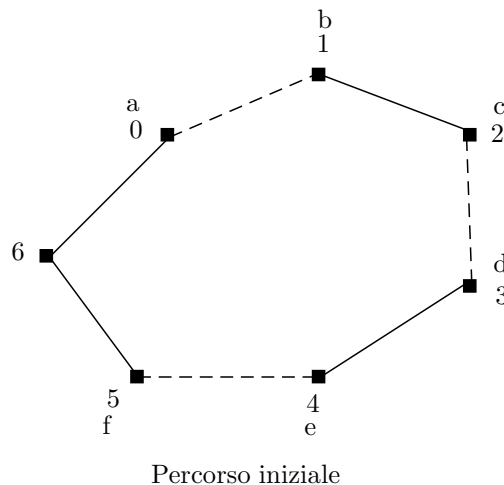


FIGURA 3.4 – Il circuito iniziale. Gli archi tratteggiati *ab*, *cd* ed *ef* sono quelli che si vuole rimuovere.

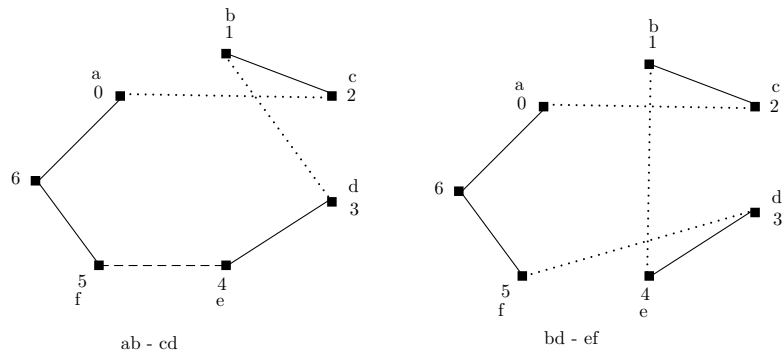


FIGURA 3.5 – Primo scambio 3-ottimale, si effettui un 2-ottimale prima sugli archi $ab - cd$ dopodiché su $bd - ef$.

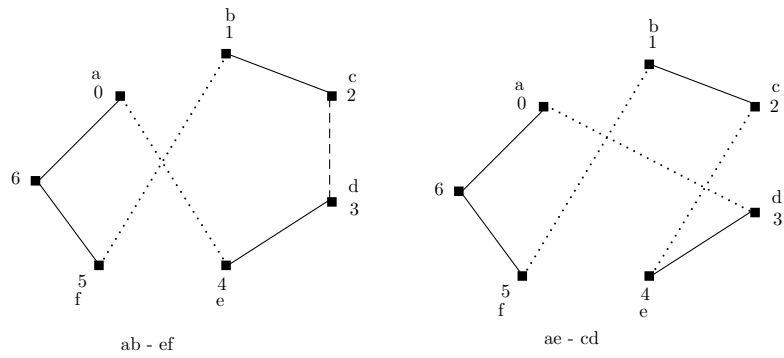


FIGURA 3.6 – Secondo scambio 3-ottimale, si effettui un 2-ottimale prima sugli archi $ab - ef$ dopodiché su $ae - cd$.

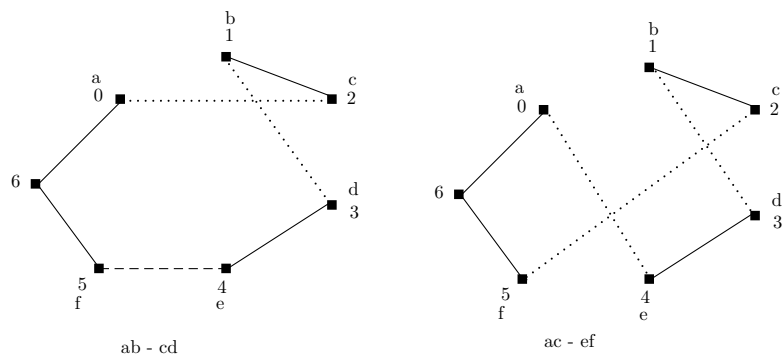


FIGURA 3.7 – Terzo scambio 3-ottimale, si effettui un 2-ottimale prima sugli archi $ab - cd$ dopodiché su $ac - ef$.

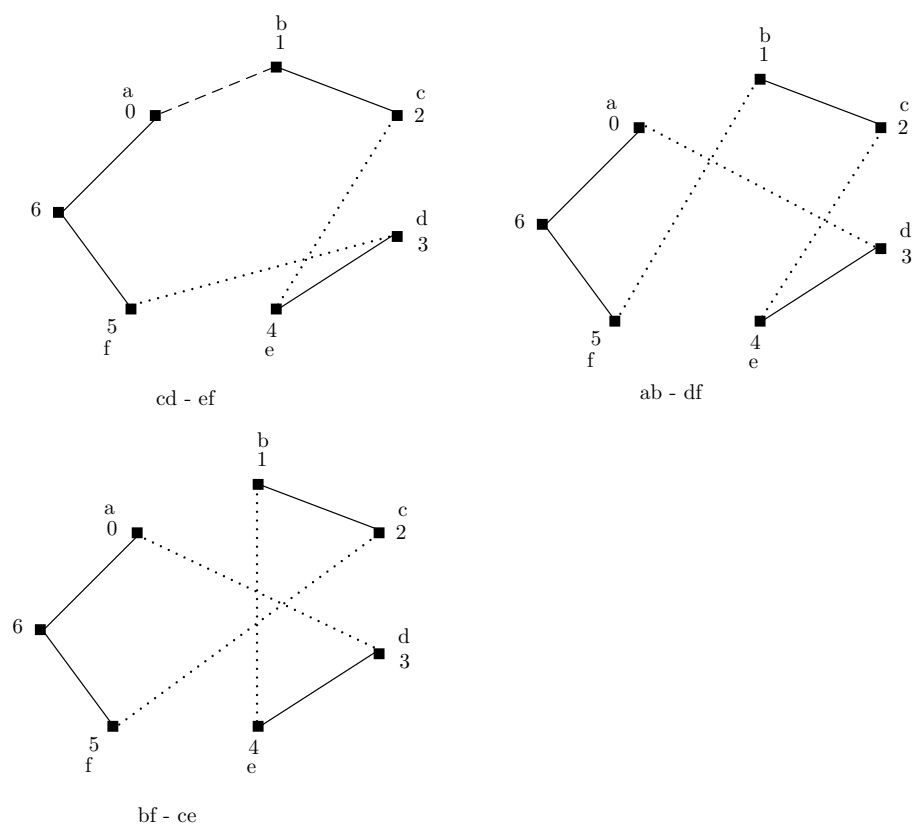


FIGURA 3.8 – Ultimo scambio 3-ottimale, è l'unico che necessita di tre scambi 2-ottimali in ordine sugli archi $cd - ef$, $ab - df$ ed infine $bf - ce$.

3.4 L'algoritmo per il TSP

In seguito viene descritto il BA implementato per il TSP simmetrico. La struttura dati utilizzata per la rappresentazione di ciascun cromosoma è la satellites list esattamente come descritta nella sezione precedente. La funzione di fitness utilizzata è l'inverso della lunghezza del circuito e si è scelta una popolazione di dimensione fissa p . La struttura dell'algoritmo rispecchia quella presentata nel paragrafo 3.2 ed è la seguente.

Sia $g = 1, \dots, 20$ l'indice delle generazioni, $p = 200$ il numero di soluzioni della popolazione, c il numero di città considerate, $X = \{x_i : i = 1, \dots, p\}$ l'insieme delle soluzioni dove ogni x_i contiene un tour di c città.

1. *Inizializzazione*

- (a) Seleziona c città in modo random o leggendole da un file.
- (b) Setta il contatore delle iterazioni $g = 1$.
- (c) Crea l'insieme delle soluzioni, ovvero p tour ammissibili di c città disposte in maniera random utilizzando l'algoritmo Fisher-Yates shuffle².
- (d) Sia quindi X^1 l'insieme delle soluzioni iniziale.

2. *Maturazione*

- (a) Se $g > 1$ allora esegui su ciascuna soluzione l'algoritmo 2-opt.
- (b) Se $g > 1$ allora esegui su ciascuna soluzione l'algoritmo di *customer relocation* (3-opt semplificato).
- (c) Se $g = g_{max}$ allora esegui su ciascuna soluzione l'algoritmo 3-opt e termina dopo i passi (c) e (d).
- (d) Sia x^* la migliore soluzione (ottimo locale) associata ad X^g .
- (e) Sia $z(x^*)$ il costo di x^* e $d(x^*)$ la lunghezza dell'attuale miglior tour.

3. *Definizione dell'insieme dei genitori*

- (a) Sia $\overline{X^g}$ l'insieme delle soluzioni maturato nello Step 2.
- (b) Orienta tutti i tour nella direzione che minimizza $\sum_{i=0}^c i * x[i]$ dove $x[i]$ indica la città in posizione i nella soluzione x .
- (c) Per ogni soluzione calcola la funzione di similarità (distanza) con tutte le altre soluzioni. Sia $diff_{ij}$ il numero di archi differenti tra x_i e x_j allora $\delta_{ij} = \frac{diff_{ij}}{c}$.
- (d) Scegli $\Delta = avg - 0.05 \text{ std}$ dove la media $avg = \frac{\sum_{i=1}^p \sum_{j=1}^p \delta_{ij}}{(p * p) - p}$ $i \neq j$ e lo scarto quadratico medio $std = \sum_{i=1}^p \sum_{j=1}^p \sqrt{\frac{(x_{ij} - avg)^2}{p}}$ $i \neq j$.
- (e) Definisci un grafo di adiacenza (o similarità) $G(X^g)$ in accordo con la sopra definita distanza. Due soluzioni vengono dette simili se la loro distanza non è maggiore di Δ .
- (f) Ordina le soluzioni per fitness.

²L'algoritmo è conosciuto anche come Knuth shuffle. É un metodo per generare una permutazione casuale di un insieme finito.

- (g) Definisci una frequenza di inclusione θ_i^g per ogni soluzione x_i^g classificando per valore crescente di $f(x_i^g)$ la popolazione ed impostando $\theta_i^g = \lceil \frac{\text{rank}(x_i^g)}{5} \rceil$
- (h) Definisci l' r -esimo parents set P_r^g come un insieme indipendente nel seguente modo:
 - i. Scegli un numero casuale n_r tale che $1 \leq n_r \leq p$.
 - ii. Inserisci nell'insieme indipendente P_r^g , ora vuoto, la soluzione x_{n_r} .
 - iii. Prosegui in modo circolare da $n_r + 1$ a $n_r - 1$ ed aggiungi le soluzioni analizzate se non esiste alcun arco tra loro e le soluzioni già presenti in P_r^g e la somma delle fitness degli elementi selezionati è inferiore ad $\frac{1}{10}$ della somma della fitness di tutti gli individui della popolazione (per limitare la dimensione del parents set).
- (i) Decrementa θ_i^g per ogni soluzione x_i^g selezionata e rimuovila da $G(X^g)$ se $\theta_i^g = 0$.
- (j) Ripeti (g), (h) e (i) finché il numero totale dei genitori selezionati non è p .
- (k) Sia $P = \{P_r^g : r = 1, \dots, r_g\}$ l'insieme di tutti i parent sets.

4. Crossover, definizione dell'insieme dei figli

- (a) Sia k la dimensione dell' r -esimo parent set P_r^g .
- (b) Genera k soluzioni figlie ammissibili utilizzando una quantità proporzionale ($\frac{c}{k}$) di città da ogni genitore all'interno di P_r^g evitando di prendere città duplicate.
- (c) Sia i l'indice del figlio che si sta generando allora inizia a selezionare le città da utilizzare partendo dal parent i e continuando in maniera circolare finché l' $(i - 1)$ -esimo non è raggiunto (FIGURA 3.9).
- (d) Ripeti il procedimento per tutti gli insiemi di genitori generati.
- (e) Una volta generati p figli sostituisci in blocco la popolazione della generazione precedente.

5. Mutazione

- (a) Effettua una mutazione ad ogni elemento creato con una probabilità $P_m = 0.1$. La mutazione consiste nello scambio di due città scelte casualmente.

6. Terminazione

- (a) Ripeti gli Step dal 2 al 5 finché non è raggiunto il limite di generazioni.

(b) Seleziona la miglior soluzione trovata.

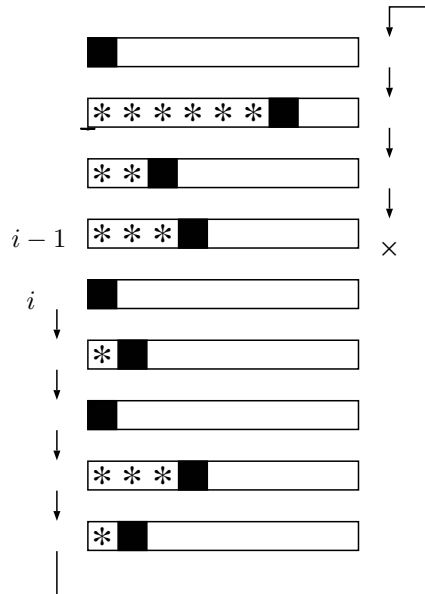


FIGURA 3.9 – Crossover standard. Il processo inizia dal genitore i e prosegue fino all' $i - 1$. I quadratini scuri indicano città scelte per essere inserite nel figlio mentre gli asterischi città saltate in quanto già presenti. Questo processo genera una soluzione ammissibile.

3.4.1 Un esempio di esecuzione

Verrà ora descritto un esempio di crossover, ovvero una delle parti peculiari del BA, in modo da chiarire ancora meglio l'algoritmo sopra definito. Si supponga di aver un insieme di 14 città indicizzate a 0 a 13 ed una popolazione di 10 elementi. L'esempio è verosimile in quanto verrà rilassato il vincolo per cui la somma della fitness in ciascun parents set debba essere minore o uguale di un decimo della fitness totale. Se non venisse eliminato questo vincolo non verrebbero prodotti parents set significativi in quanto 10 elementi non sono sufficienti, ma lo sono decisamente per un essere umano. I dati presentati sono reali e provengono da una esecuzione dell'algoritmo senza il vincolo.

La matrice delle distanze tra le città è la seguente:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0		2	5	7	9	6	4	1	1	3	1	5	3	4
1	2		4	6	9	6	5	2	3	4	3	5	4	4
2	5	4		2	7	4	4	5	6	8	6	3	5	2
3	7	6	2		5	3	4	6	7	10	7	2	5	3
4	9	9	7	5		3	5	8	9	11	9	4	6	6
5	6	6	4	3	3		2	5	6	8	6	1	3	2
6	4	5	4	4	5	2		3	4	7	4	2	1	2
7	1	2	5	6	8	5	3		1	4	1	4	2	3
8	1	3	6	7	9	6	4	1		2	0	6	3	5
9	3	4	8	10	11	8	7	4	2		3	8	5	7
10	1	3	6	7	9	6	4	1	0	3		5	3	5
11	5	5	3	2	4	1	2	4	6	8	5		3	2
12	3	4	5	5	6	3	1	2	3	5	3	3		3
13	4	4	2	3	6	2	2	3	5	7	5	2	3	

La popolazione, dove per semplicità è omessa la generazione, è così composta:

$$\begin{aligned}
 x_0 &= (0, 9, 8, 10, 7, 12, 6, 11, 5, 4, 3, 2, 13, 1) & f(x_0) &= 0.0333 & \theta_0 &= 2 \\
 x_1 &= (0, 9, 8, 10, 7, 12, 6, 13, 11, 5, 4, 3, 2, 1) & f(x_1) &= 0.0333 & \theta_1 &= 2 \\
 x_2 &= (0, 9, 8, 10, 7, 12, 6, 13, 5, 4, 11, 3, 2, 1) & f(x_2) &= 0.0333 & \theta_2 &= 2 \\
 x_3 &= (0, 9, 8, 10, 7, 12, 6, 5, 4, 11, 3, 2, 13, 1) & f(x_3) &= 0.0333 & \theta_3 &= 2 \\
 x_4 &= (0, 10, 8, 9, 1, 2, 3, 11, 4, 5, 13, 6, 12, 7) & f(x_4) &= 0.0333 & \theta_4 &= 2 \\
 x_5 &= (0, 9, 8, 10, 7, 12, 6, 13, 5, 4, 11, 3, 2, 1) & f(x_5) &= 0.0333 & \theta_5 &= 2 \\
 x_6 &= (0, 7, 12, 6, 5, 4, 11, 3, 2, 13, 1, 9, 8, 10) & f(x_6) &= 0.0333 & \theta_6 &= 1 \\
 x_7 &= (0, 10, 8, 9, 1, 2, 13, 11, 3, 4, 5, 6, 12, 7) & f(x_7) &= 0.0322 & \theta_7 &= 1 \\
 x_8 &= (0, 10, 8, 9, 12, 6, 4, 5, 11, 3, 2, 13, 7, 1) & f(x_8) &= 0.0322 & \theta_8 &= 1 \\
 x_9 &= (0, 9, 8, 10, 7, 12, 6, 11, 3, 4, 5, 13, 2, 1) & f(x_9) &= 0.0322 & \theta_9 &= 1
 \end{aligned}$$

La matrice delle diversità (o distanza tra soluzioni) è la seguente. Essa è chiaramente simmetrica rispetto alla diagonale principale.

	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9
x_0		0.50	0.43	0.29	1.00	0.43	1.00	1.00	0.93	0.43
x_1	0.50		0.29	0.50	1.00	0.29	1.00	1.00	0.93	0.43
x_2	0.43	0.29		0.50	1.00	0.00	1.00	1.00	0.93	0.43
x_3	0.29	0.50	0.50		1.00	0.50	1.00	1.00	0.93	0.50
x_4	1.00	1.00	1.00	1.00		1.00	1.00	0.43	0.79	1.00
x_5	0.43	0.29	0.00	0.50	1.00		1.00	1.00	0.93	0.43
x_6	1.00	1.00	1.00	1.00	1.00	1.00		1.00	1.00	1.00
x_7	1.00	1.00	1.00	1.00	0.43	1.00	1.00		0.79	0.79
x_8	0.93	0.93	0.93	0.93	0.79	0.93	1.00	0.79		0.93
x_9	0.43	0.43	0.43	0.50	1.00	0.43	1.00	0.79	0.93	

La media delle diversità avg è pari a 0.76 mentre lo scarto quadratico medio std è 0.86 quindi scegliendo Δ come indicato nell'algoritmo si ottiene 0.72. Se non fosse stato rilassato il vincolo sulla fitness totale dei parents se si avrebbero 10 insiemi di 1 genitore ciascuno a causa delle poche soluzioni presenti nella popolazione e della struttura del problema che ha di per se poche città.

Sia quindi $P = \{P_0, P_1, P_2, P_3, P_4\}$ l'insieme *verosimile* dei parents set così composto:

$$P_0 = \{x_7, x_8, x_9\}$$

$$P_1 = \{x_3, x_4, x_6\}$$

$$P_2 = \{x_2, x_4\}$$

$$P_3 = \{x_0\}$$

$$P_4 = \{x_4\}$$

Ora definiti i parents set si procede al crossover per creare i figli. Verrà illustrato il procedimento per P_0 (FIGURA 3.10) e forniti i risultati per i successivi insiemi. La cardinalità di P_0 è 3 quindi verranno selezionate da ogni genitore la parte intera di $\frac{14}{3}$ città ovvero $k = 4$.

x_7	0	10	8	9	1	2	13	11	3	4	5	6	12	7
x_8	0	10	8	9	12	6	4	5	11	3	2	13	7	1
x_9	0	9	8	10	7	12	6	11	3	4	5	13	2	1

FIGURA 3.10 – Il parent set P_0 .

ESECUZIONE DELL'ALGORITMO DI CROSSOVER

1. *Definizione primo figlio*

- (a) Seleziona il corrispondente genitore iniziale. Per il primo figlio x'_0 viene selezionato il primo genitore del parent set cioè x_7 .
- (b) Seleziona le prime 4 città di x_7 (**0,10,8,9**) ed inseriscile in x'_0 .

$$x'_0 \quad \boxed{0} \boxed{10} \boxed{8} \boxed{9} \boxed{} \boxed{} \boxed{} \boxed{} \boxed{} \boxed{} \boxed{} \boxed{} \boxed{} \boxed{} \boxed{} \boxed{}$$

- (c) Passa al genitore successivo x_8 .
- (d) Salta le prime 4 città (**0, 10, 8, 9**) in quanto già presenti nel figlio ed aggiungi **12, 6, 4, 5**.

$$x'_0 \quad \boxed{0} \boxed{10} \boxed{8} \boxed{9} \boxed{12} \boxed{6} \boxed{4} \boxed{5} \boxed{} \boxed{} \boxed{} \boxed{} \boxed{} \boxed{} \boxed{} \boxed{}$$

- (e) Prosegui al genitore x_9 .
- (f) Ignora le prime 4 città, scegli **7**, ignora **12** e **6**, aggiungi **11, 3**, ignora **4** e **5** ed aggiungi **13**.

$$x'_0 \quad \boxed{0} \boxed{10} \boxed{8} \boxed{9} \boxed{12} \boxed{6} \boxed{4} \boxed{5} \boxed{7} \boxed{11} \boxed{3} \boxed{13} \boxed{} \boxed{} \boxed{} \boxed{}$$

- (g) Riparti da x_7 (questo si intende per *in modo circolare*).
- (h) Salta le città già presenti **0,10,8,9** ed aggiungi **1** e **2**.

$$x'_0 \quad \boxed{0} \boxed{10} \boxed{8} \boxed{9} \boxed{12} \boxed{6} \boxed{4} \boxed{5} \boxed{7} \boxed{11} \boxed{3} \boxed{13} \boxed{1} \boxed{2} \boxed{} \boxed{}$$

La FIGURA 3.11 mostra la sequenza temporale di esecuzione (i numeri romani) ed all'interno di ciascuna di esse la posizione che la città andrà ad occupare in x'_0 .

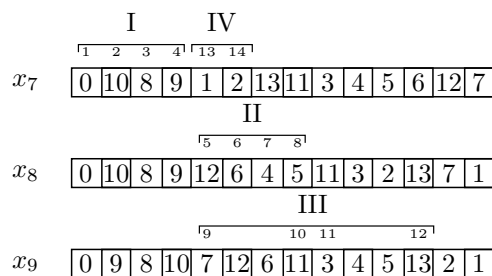


FIGURA 3.11 – La sequenza di città selezionate ad ogni passaggio.

2. *Definizione secondo figlio*

- (a) Seleziona come genitore di partenza x_8 .
- (b) Ripeti l'algoritmo esattamente come sopra.
- (c) ...

$$x'_1 \quad \boxed{0} \boxed{10} \boxed{8} \boxed{9} \boxed{7} \boxed{12} \boxed{6} \boxed{11} \boxed{1} \boxed{2} \boxed{13} \boxed{4} \boxed{5} \boxed{3}$$

3. *Definizione terzo figlio*

- (a) Seleziona come genitore di partenza x_9 .
- (b) Ripeti l'algoritmo esattamente come sopra.
- (c) ...

$$x'_2 \quad \boxed{0} \boxed{9} \boxed{8} \boxed{10} \boxed{1} \boxed{2} \boxed{13} \boxed{11} \boxed{6} \boxed{4} \boxed{5} \boxed{3} \boxed{7} \boxed{12}$$

I figli relativi a P_1 sono:

$$x'_3 = (0, 9, 8, 10, 1, 2, 3, 11, 7, 12, 6, 5, 4, 13)$$

$$x'_4 = (0, 10, 8, 9, 7, 12, 6, 5, 4, 11, 3, 2, 1, 13)$$

$$x'_5 = (0, 7, 12, 6, 9, 8, 10, 5, 1, 2, 3, 11, 4, 13)$$

I figli relativi a P_2 sono:

$$x'_6 = (0, 9, 8, 10, 7, 12, 6, 1, 2, 3, 11, 4, 5, 13)$$

$$x'_7 = (0, 10, 8, 9, 1, 2, 3, 7, 12, 6, 13, 5, 4, 11)$$

In questo caso sono state scelte 7 città da ciascun genitore.

Il figlio relativo a P_3 è:

$$x'_8 = (0, 9, 8, 10, 7, 12, 6, 11, 5, 4, 3, 2, 13, 1)$$

I figli relativi a P_4 sono:

$$x'_9 = (0, 10, 8, 9, 1, 2, 3, 11, 4, 5, 13, 6, 12, 7)$$

Gli ultimi due figli x'_8 e x'_9 sono stati ricopiati senza alcuna modifica causata dal crossover in quanto erano in parents set di cardinalità 1. Un'alterazione potrebbe occorrere in caso di mutazione.

3.5 Crossover alternativi

Come già detto diverse volte le componenti peculiari del BA sono (1) la selezione di genitori multipli e (2) il crossover. Il criterio di selezione dei genitori finora adottato è più che ragionevole. Selezionando un insieme indipendente di soluzioni dal grafo delle similarità si ottiene un parents set *sicuramente* diversificato. Per quanto riguarda il crossover si apre un ampio ventaglio di possibilità ed alternative. È complicato capire quando e perché una tecnica di crossover raggiunge risultati migliori rispetto ad altre. Nei prossimi due paragrafi sono descritti, in ordine temporale, due diverse metodologie di crossover implementate.

3.5.1 Crossover alternativo #1: 2-opt crossover

Il crossover descritto nel paragrafo 3.4 consiste nel selezionare un certo sottoinsieme di città dai vari genitori e creare iterativamente una nuova soluzione ammissibile. Questo processo analizzato singolarmente assomiglia in qualche modo a quello che fa un *algoritmo di costruzione del tour* descritto nel paragrafo 2.3. Un'idea alternativa è quella di un crossover che usa gli archi dei genitori per generare una soluzione figlio. Il seguente algoritmo di crossover è una alternativa per il punto (4) nella descrizione dell'algoritmo nel paragrafo 3.4.

4. ALT.1 Crossover, definizione dell'insieme dei figli

- (a) Sia k la dimensione dell' r -esimo parent set P_r^g .
- (b) Genera un grafo contenente gli archi delle k soluzioni presenti in P_r^g (FIGURA 3.12).
- (c) Sia i l'indice del figlio da generare, $1 \leq i \leq k$.
- (d) Copia il tour del genitore i nel figlio i .
- (e) Effettua un 2-opt sul figlio i considerando solo gli archi del grafo generato al punto (b).
- (f) Ripeti i passi da (b) a (e) per ogni soluzione di ciascun membro del parents set in modo da generare k figli.
- (g) Una volta ripetuto l'algoritmo per ciascun parents set saranno stati generati p figli che sostituiranno in blocco la popolazione della generazione precedente.

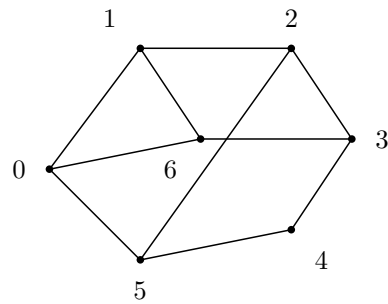


FIGURA 3.12 – Un esempio di grafo degli archi in un certo parents set.

	0	1	2	3	4	5	6
0		1	0	0	0	1	1
1	1		1	0	0	0	1
2	0	1		1	0	1	0
3	0	0	1		1	0	0
4	0	0	0	1		1	0
5	1	0	1	0	1		0
6	1	1	0	0	0	0	

TABELLA 3.1 – La matrice associata contiene 1 se esiste un arco tra le città in incrocio tra riga e colonna, 0 altrimenti.

Inoltre, per evitare che al crescere delle generazioni il crossover tenda ad un 2-opt standard si è impostata la probabilità di mutazione $P_m = 1$. Successivamente ci si riferirà all'algoritmo implementato con questa variante con *algoritmo bionomico ALT.1*

3.5.2 Crossover alternativo #2: *gen-based crossover*

Questo crossover assomiglia di più a quello standard ovvero quello che si cerca di ottenere è che ciascun figlio erediti una qualche caratteristica da tutti i genitori (sperando che sia buona). Piuttosto che selezionare piccole porzioni di geni da ciascun genitore viene effettuato una operazione di crossover modificato³ *iterativa* con ciascuna soluzione appartenente al parents set (FIGURA 3.13). Questa operazione permette di generare tanti figli quanti sono gli elementi del parents set. L'algoritmo, d'ora in avanti chiamato *algoritmo bionomico ALT.2*, è il seguente:

4. ALT.2 Crossover, definizione dell'insieme dei figli

³Da qui nasce il nome di *gen-based*.

- (a) Sia k la dimensione dell' r -esimo parent set P_r^g .
- (b) Sia i l'indice del figlio z da generare, $1 \leq i \leq k$.
- (c) Effettua un crossover modificato tra i genitori in posizione⁴ i e $i + 1$ e scegli tra uno dei due figli generati in modo casuale. Sia z^1 quello scelto.
- (d) Effettua iterativamente un crossover modificato tra la soluzione temporanea z^α con $1 \leq \alpha \leq k$ e i genitori successivi ($i + 2, i + 3, \dots, i - 1$) scegliendo tra una delle due soluzioni generate in modo casuale.
- (e) Sia z la soluzione generata.
- (f) Ripeti i passi da (b) a (d) per ogni soluzione di ciascun membro del parents set in modo da generare k figli.
- (g) Una volta ripetuto l'algoritmo per ciascun parents set saranno stati generati p figli che sostituiranno in blocco la popolazione della generazione precedente.

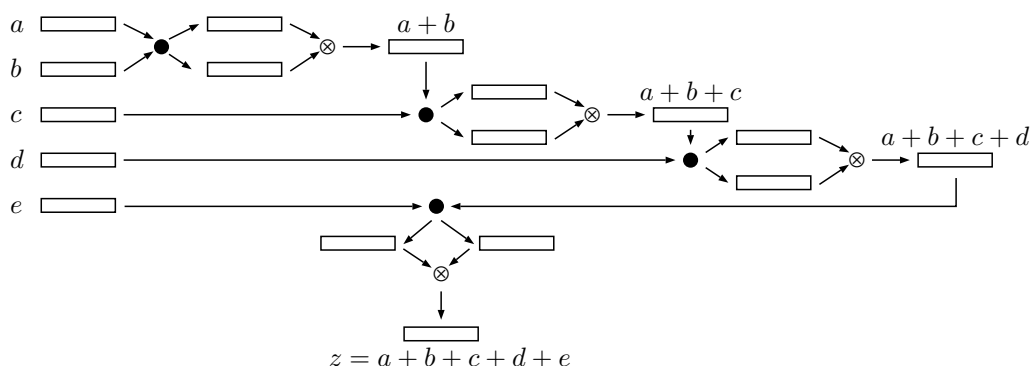


FIGURA 3.13 – Crossover alternativo #2. In figura è mostrata la generazione del figlio z . I pallini neri pieni indicano l'operazione di crossover modificato come indicata nel paragrafo 1.3.4.1 mentre i pallini con la x indicano una selezione casuale tra i due cromosomi.

3.6 Risultati

Il BA implementato è stato testato su diversi problemi euclidei simmetrici presenti in TSPLIB al link <http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/>. Il BA è stato realizzato in versione *standard* e con crossover alternativi #1 e #2. Inoltre

⁴Come al solito considerando il parents set in maniera circolare, quindi se $i + 1 = 11$ ma l'insieme ha cardinalità 10 si prenda la soluzione iniziale.

questi sono stati confrontati con un GA del tutto analogo nel quale le uniche differenze sono (1) nella selezione dei genitori, dove viene utilizzato Roulette Wheel Selection, (2) e nell'operatore crossover, dove viene utilizzato un single-point crossover modificato definito come in 1.3.4.1. Sono state realizzate tre tabelle contenenti le seguenti colonne:

- TABELLA 3.3: *lunghezza media dei tour nelle cinque esecuzioni.*

- *Prob*: identificativo dell'istanza del problema, esso è composto da un nome ed un numero identificativo della quantità delle città presenti;
- z^* : valore della soluzione ottima.
- \bar{g} : valore medio del GA calcolato su 5 esecuzioni.
- \bar{b} : valore medio del BA calcolato su 5 esecuzioni.
- \bar{b}_1 : valore medio del BA ALT.1 calcolato su 5 esecuzioni.
- \bar{b}_2 : valore medio del BA ALT.2 calcolato su 5 esecuzioni.
- $d_{\bar{g}}$: distanza in percentuale di \bar{g} da z^* .
- $d_{\bar{b}}$: distanza in percentuale di \bar{b} da z^* .
- $d_{\bar{b}_1}$: distanza in percentuale di \bar{b}_1 da z^* .
- $d_{\bar{b}_2}$: distanza in percentuale di \bar{b}_2 da z^* .

- TABELLA 3.4: *lunghezza migliore dei tour nelle cinque esecuzioni.*

- *Prob*: identificativo dell'istanza del problema.
- z^* : valore della soluzione ottima.
- g^* : valore della miglior soluzione trovata per il GA.
- b^* : valore della miglior soluzione trovata per il BA.
- b_1^* : valore della miglior soluzione trovata per per il BA ALT.1 .
- b_2^* : valore della miglior soluzione trovata per per il BA ALT.2 .
- d_{g^*} : distanza in percentuale di g^* da z^* .
- d_{b^*} : distanza in percentuale di b^* da z^* .
- $d_{b_1^*}$: distanza in percentuale di b_1^* da z^* .
- $d_{b_2^*}$: distanza in percentuale di b_2^* da z^* .

- TABELLA 3.5: *tempo medio nelle cinque esecuzioni (in secondi).*

- \bar{t}_g : tempo di esecuzione medio per il GA.

- \bar{t}_b : tempo di esecuzione medio per per il BA.
- \bar{t}_{b_1} : tempo di esecuzione medio per per il BA ALT.1 .
- \bar{t}_{b_2} : tempo di esecuzione medio per per il BA ALT.2 .

La TABELLA 3.2 riassume le differenze tra gli algoritmi. L'algoritmo è stato implementato in C ed eseguito su un computer avente una CPU Intel i5 dual-core da 1.70 GHz ciascuno. Sono state utilizzate le librerie OMP per svolgere in parallelo le operazioni possibili. L'arrotondamento utilizzato per la misura delle distanze è quello descritto nella documentazione di TSPLIB, ovvero *nearest int*. I risultati sono approssimati alla seconda cifra decimale per quanto riguarda i tempi mentre alla prima per le medie delle distanze. L'ultima riga della tabelle valori aggregati come media e somma. Le lunghezze in grassetto sono le migliori trovate tra i quattro algoritmi mentre le distanze dall'ottimo in grassetto indicano dove è stato raggiunto l'ottimo.

	g	b	b1	b2
Selezione genitori	Roulette Wheel Selection	Insieme indipendente	Insieme indipendente	Insieme indipendente
Crossover	single-point modificato	<i>standard</i>	2-opt	gen-based
Probabilità di mutazione	10 %	10 %	100 %	10 %

TABELLA 3.2 – Schema delle differenze tra gli algoritmi.

<i>Prob</i>	z^*	\bar{g}	\bar{b}	\bar{b}_1	\bar{b}_2	$d_{\bar{g}}$	$d_{\bar{b}}$	$d_{\bar{b}_1}$	$d_{\bar{b}_2}$
a280	2579	2632.4	2637.0	2647.8	2582.4	2.07	2.25	2.67	0.13
berlin52	7542	7542.0	7542.0	7542.0	7542.0	0.00	0.00	0.00	0.00
bier127	118282	118648.8	118876.6	118714.8	118317.8	0.31	0.50	0.36	0.03
ch130	6110	6159.0	6158.0	6168.6	6116.4	0.80	0.78	0.96	0.10
ch150	6528	6581.8	6579.2	6567.2	6528.0	0.82	0.78	0.81	0.00
d198	15780	15879.0	15881.4	15926.6	15790.2	0.63	0.64	0.92	0.06
d493	35002	35838.0	35832.2	35925.0	35312.0	2.39	2.37	2.64	0.88
eil51	426	427.4	427.0	427.0	426.4	0.32	0.23	0.23	0.09
eil76	538	541.0	543.0	542.2	538.0	0.56	0.93	0.78	0.00
eil101	629	636.6	636.6	640.2	629.8	1.21	1.21	1.78	0.13
fl417	11861	11948.0	11989.8	11973.2	11890.0	0.73	1.08	0.94	0.24
gil262	2378	2409.2	2418.0	2425.8	2387.8	1.31	1.68	2.01	0.41
kroA100	21282	21301.0	21286.0	21286.6	21282.0	0.09	0.02	0.02	0.00
kroB100	22141	22171.8	22161.0	22168.4	22141.0	0.14	0.09	0.12	0.00
kroC100	20749	20779.4	20758.6	20754.6	20749.0	0.15	0.05	0.02	0.00
kroD100	21294	21368.8	21372.8	21325.8	21310.0	0.35	0.37	0.14	0.07
kroE100	22068	22148.4	22134.2	22148.0	22068.0	0.36	0.30	0.36	0.00
kroA150	26524	26677.2	26724.2	26723.4	26546.4	0.55	0.75	0.75	0.08
kroB150	26130	26322.0	26301.2	26294.4	26165.6	0.73	0.65	0.63	0.14
kroA200	29368	29633.6	29589.2	29671.0	29453.8	0.90	0.75	1.03	0.29
kroB200	29437	29729.4	29746.4	29831.0	29451.2	0.99	1.05	1.34	0.05
lin105	14379	14383.4	14379.0	14379.0	14379.0	0.03	0.00	0.00	0.00
lin318	42029	42716.4	42632.0	42919.0	42337.4	1.63	1.43	2.11	0.73
pcb442	50778	52034.4	52198.6	52166.8	51424.4	2.47	2.80	2.73	1.27
pr76	108159	108258.4	108249.0	108237.8	108205.0	0.09	0.08	0.07	0.04
pr107	44303	44428.6	44425.0	44395.2	44311.8	0.28	0.27	0.21	0.02
pr124	59030	59030.0	59048.4	59030.0	59030.0	0.00	0.03	0.00	0.00
pr136	96772	97406.2	97458.6	97520.8	96846.8	0.65	0.71	0.77	0.08
pr144	58537	58537.0	58537.0	58537.0	58537.0	0.00	0.00	0.00	0.00
pr152	73682	73683.6	73738.8	73720.6	73682.8	0.00	0.08	0.05	0.00
pr226	80369	80707.0	80417.0	80537.6	80369.8	0.42	0.05	0.21	0.00
pr264	49135	49213.2	49259.8	49323.0	49149.4	0.16	0.25	0.38	0.03
pr299	48191	49109.4	49441.8	48843.6	48398.6	1.90	2.59	1.35	0.43
rat99	1211	1219.8	1217.2	1213.8	1211.8	0.72	0.51	0.23	0.07
rat195	2323	2365.8	2367.2	2374.6	2337.6	1.84	1.90	2.22	0.63
rd100	7910	7919.8	7917.8	7926.2	7916.8	0.12	0.10	0.20	0.08
st70	675	675.2	676.0	675.4	675.0	0.03	0.15	0.06	0.00
ts225	126643	126792.4	126765.6	126738.8	126709.4	0.12	0.09	0.07	0.05
tsp225	3916	4011.4	4023.8	4014.0	3948.0	2.42	2.75	2.50	0.82
u159	42080	42443.8	42626.6	42494.2	42080.0	0.86	1.30	0.98	0.00
<i>Media</i>						0.73	0.79	0.82	0.17

TABELLA 3.3 – Valori medi delle cinque esecuzioni

<i>Prob</i>	z^*	g^*	b^*	b_1^*	b_2^*	d_{g^*}	d_{b^*}	$d_{b_1^*}$	$d_{b_2^*}$
a280	2579	2622	2608	2634	2579	1.67	1.12	2.13	0.00
berlin52	7542	7542	7542	7542	7542	0.00	0.00	0.00	0.00
bier127	118282	118313	118510	118313	118282	0.03	0.19	0.03	0.00
ch130	6110	6129	6137	6135	6110	0.31	0.44	0.40	0.00
ch150	6528	6558	6550	6528	6528	0.46	0.34	0.00	0.00
d198	15780	15861	15864	15905	15781	0.51	0.53	0.79	0.01
d493	35002	35660	35687	35823	35248	1.88	1.96	2.34	0.70
eil51	426	427	426	427	426	0.23	0.00	0.23	0.00
eil76	538	538	542	541	538	0.00	0.74	0.56	0.00
eil101	629	634	633	638	629	0.79	0.63	1.43	0.00
fl417	11861	11924	11964	11954	11877	0.53	0.87	0.78	0.13
gil262	2378	2393	2413	2418	2382	0.63	1.47	1.68	0.17
kroA100	21282	21282	21282	21282	21282	0.00	0.00	0.00	0.00
kroB100	22141	22146	22141	22141	22141	0.02	0.00	0.00	0.00
kroC100	20749	20753	20749	20749	20749	0.02	0.00	0.00	0.00
kroD100	21294	21309	21343	21294	21294	0.07	0.23	0.00	0.00
kroE100	22068	22097	22073	22068	22068	0.13	0.02	0.00	0.00
kroA150	26524	26642	26665	26563	26524	0.44	0.53	0.15	0.00
kroB150	26130	26287	26261	26278	26137	0.60	0.50	0.57	0.03
kroA200	29368	29556	29521	29647	29435	0.64	0.52	0.95	0.23
kroB200	29437	29666	29581	29611	29439	0.78	0.49	0.59	0.01
lin105	14379	14379	14379	14379	14379	0.00	0.00	0.00	0.00
lin318	42029	42670	42566	42583	42157	1.52	1.28	1.32	0.30
pcb442	50778	51718	52084	52006	51283	1.85	2.57	2.41	0.99
pr76	108159	108159	108159	108159	108159	0.00	0.00	0.00	0.00
pr107	44303	44358	44326	44347	44303	0.12	0.05	0.10	0.00
pr124	59030	59030	59030	59030	59030	0.00	0.00	0.00	0.00
pr136	96772	97098	97223	97062	96798	0.34	0.47	0.30	0.03
pr144	58537	58537	58537	58537	58537	0.00	0.00	0.00	0.00
pr152	73682	73682	73682	73682	73682	0.00	0.00	0.00	0.00
pr226	80369	80397	80369	80455	80369	0.03	0.00	0.11	0.00
pr264	49135	49135	49144	49212	49135	0.00	0.02	0.16	0.00
pr299	48191	48992	49170	48756	48318	1.66	2.03	1.17	0.26
rat99	1211	1217	1212	1211	1211	0.49	0.08	0.00	0.00
rat195	2323	2349	2357	2365	2323	1.11	1.46	1.81	0.00
rd100	7910	7910	7917	7911	7910	0.00	0.09	0.01	0.00
st70	675	675	675	675	675	0.00	0.00	0.00	0.00
ts225	126643	126726	126726	126726	126643	0.06	0.06	0.06	0.00
tsp225	3916	3995	4009	4008	3936	2.02	2.37	2.35	0.51
u159	42080	42371	42396	42257	42080	0.69	0.75	0.42	0.00
<i>Media</i>						0.49	0.54	0.57	0.08

TABELLA 3.4 – Valori migliori delle cinque esecuzioni

<i>Prob</i>	\bar{t}_g	\bar{t}_b	\bar{t}_{b_1}	\bar{t}_{b_2}
a280	473.90	571.81	414.52	105.40
berlin52	1.08	2.04	1.25	0.76
bier127	24.44	38.13	24.05	8.82
ch130	23.64	39.43	23.40	9.45
ch150	46.70	69.89	41.09	15.51
d198	137.21	173.23	125.53	42.74
d493	4593.90	5175.01	4066.17	1138.90
eil51	0.86	1.74	1.13	0.72
eil76	3.53	7.23	4.05	2.06
eil101	11.10	17.62	11.17	4.24
fl417	1640.00	2205.52	1750.16	506.67
gil262	410.11	512.69	353.14	106.85
kroA100	9.74	17.72	10.26	4.72
kroB100	10.42	17.47	11.16	5.57
kroC100	8.64	16.70	9.73	4.59
kroD100	10.35	19.56	11.97	5.06
kroE100	10.67	18.03	11.55	4.90
kroA150	42.53	68.71	43.50	15.92
kroB150	49.00	72.77	48.37	17.47
kroA200	133.80	187.46	121.56	40.13
kroB200	144.83	194.25	133.73	45.57
lin105	9.09	14.69	11.34	5.34
lin318	817.34	996.25	733.34	217.63
pcb442	2611.38	3134.28	2630.27	631.33
pr76	4.50	7.03	4.83	2.30
pr107	18.83	19.86	17.70	7.90
pr124	16.15	26.22	15.88	8.95
pr136	33.98	51.41	33.73	12.02
pr144	33.15	42.90	30.61	12.47
pr152	49.09	58.56	41.16	31.46
pr226	165.42	169.23	148.88	60.88
pr264	384.84	494.40	393.79	116.09
pr299	825.94	1008.73	718.39	277.94
rat99	10.39	17.38	10.76	6.48
rat195	123.42	169.28	116.12	36.56
rd100	11.28	17.49	11.66	4.58
st70	3.05	5.75	3.48	1.79
ts225	223.45	253.27	199.51	93.51
tsp225	219.11	304.10	207.20	57.22
u159	43.06	71.05	48.27	16.23
Σ	13389.92	16288.89	12594.41	3686.73

TABELLA 3.5 – Tempi medi delle cinque esecuzioni

Come si evince dalle tabelle dei risultati tutti e quattro gli algoritmi restituiscono valori soddisfacenti. Le distanze in percentuale dall'ottimo raggiungono un picco massimo di 2.80 % (pcb442 BA) per quanto riguarda la media e 2.57 % (pcb442 BA) per la miglior soluzione. In media le soluzioni trovate dagli algoritmi distano meno dell'1% da quella ottima. Le migliori prestazioni sia per quanto riguarda i tempi che i valori trovati sono del BA ALT.2 il quale raggiunge l'ottimo in 28 problemi sui 40 testati ed ha in media una distanza dall'ottimo di 0.17 % per quanto riguarda i valori medi delle cinque esecuzioni (≈ 4 volte inferiore rispetto agli altri algoritmi) mentre dello 0.08 % per il miglior valore trovato (≈ 6 volte migliori rispetto agli altri). Inoltre i tempi di esecuzione sono inferiori di ≈ 3 volte rispetto al GA e al BA ALT.1 e di ≈ 4 volte rispetto al BA (converge in meno iterazioni).

3.7 Conclusioni

In questa tesi sono state sperimentate tre diverse tecniche di crossover per il BA. I risultati mostrano come il crossover sia effettivamente una componente cruciale dell'algoritmo. I crossover del BA *standard* e del BA ALT.1 non dominano il GA mentre il crossover implementato nel BA ALT.2 produce soluzioni notevolmente migliori di tutti gli altri tre algoritmi. Per riassumere il BA ALT.2 raggiunge i seguenti risultati.

- Raggiunge l'ottimo 5 volte su 5 in 12 problemi su 40.
- Raggiunge l'ottimo in 28 problemi su 40.
- La media delle distanze delle soluzioni generate è ≈ 4 volte inferiore rispetto a quella prodotta dagli altri algoritmi.
- La media delle distanze delle miglior soluzioni generate è ≈ 6 migliore rispetto a quella prodotta dagli altri algoritmi.
- Ha un tempo di esecuzione ≈ 3 volte minore rispetto al GA e al BA ALT.1 e ≈ 4 volte inferiore rispetto al BA.

L'algoritmo di Lin-Kernighan accennato nel paragrafo 2.3 produce l'ottimo in tutti i problemi testati ed è molto più stabile rispetto a qualsiasi GA o BA ma è specifico per il TSP come anche il crossover alternativo implementato in BA ALT.1. I metodi di crossover *standard* e *gen-based* implementati nel BA sono invece potenzialmente estendibili ad un qualsiasi problema di ottimizzazione combinatoria. In generale i GA e i BA forniscono una struttura robusta e flessibile per la risoluzione di un qualsiasi problema di ottimizzazione combinatoria nel quale non si ha la necessità della garanzia di trovare una soluzione ottima.

Bibliografia

- [1] Bagley, J.D. (1967) The behaviour of adaptive systems which employ genetic and correlation algorithms. Doctoral dissertation, University of Michigan.
- [2] Baker, J. E. (1985) Adaptive selection methods for genetic algorithms. In J. J. Grefenstette, ed., Proceedings of the First International Conference on Genetic Algorithms and Their Applications. Erlbaum.
- [3] Beg, S. et al (2001) Performance Evaluation of Bionomic Algorithm (BA) in Comparison with Genetic Algorithm (GA) for Shortest Path Finding Problem. Computer Science Department, COMSATS Institute of Information and Technology, Islamabad, 46000, Pakistan.
- [4] Bremermann, H. (1958) The evolution of intelligence: The nervous system as a model of its environment. Seattle, Wash: University of Washington, Dept. of Mathematics.
- [5] Caruana, R.A., and Schaffer, J.D. (1988) Representation and hidden bias: Gray vs. binary coding for genetic algorithms . Proc. 5th International Conference on Machine Learning. Morgan Kaufmann, Los Altos, CA.
- [6] Cavicchio, D.J. (1972) Adaptive search using simulated evolution. Doctoral dissertation, University of Michigan.
- [7] Christofides, N. (1994). The Bionomic Algorithm. AIRO'94 Conference. Savona, Italy.
- [8] Christofides, N. (2000) The Bionomic Algorithm and its Financial Applications. Centre for Quantitative Finance, Imperial College, London SW7 2BX
- [9] de la Maza, M., and Tidor, B. (1991). Boltzmann Weighted Selection Improves Performance of Genetic Algorithms. A.I. Memo 1345, MIT Artificial Intelligence Laboratory.

- [10] de la Maza, M., and Tidor, B. (1993). An analysis of selection procedures with particular attention paid to proportional and Boltzmann selection. In S. Forrest, ed., Proceedings of the Fifth International Conference on Genetic Algorithms. Morgan Kaufmann.
- [11] De Jong K.A. (1975) An analysis of the behaviour of a class of genetic adaptive systems. Doctoral dissertaion, University of Michigan.
- [12] De Jong, K. A., and Sarma, J. (1993) Generation gaps revisited. In L. D. Whitley, Foundations of Genetic Algorithms 2. Morgan Kaufmann.
- [13] Dianati M. et al. (2004) An Introduction to Genetic Algorithms and Evolution Strategies. 200 Univ. Ave. West, University of Waterloo, Ontario, N2L 3G1, Canada.
- [14] Eshelman L.J. et al (1989) Biases in the crossover landscape . J. David Schaffer (Ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 10-19.
- [15] Forrest, S. (1985) Scaling fitnesses in the genetic algorithm . In Documentation for PRISONERS DILEMMA and NORMS Programs That Use the Genetic Algorithm. Unpublished manuscript.
- [16] Goldberg, D. E. (1989) Genetic Algorithms in Search, Optimization, and Machine Learning .
- [17] Goldberg, D. E. (1990) A note on Boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing . Complex Systems 4: 445-460.
- [18] Held,M., Karp,R.M. (1970) The Traveling Salesman Problem and Minimum Spanning Trees, Operations Research 18, 1970, 1138–1162.
- [19] Helsgaun, K. (1998) An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic. DATALOGISKE SKRIFTER (Writings on Computer Science), No. 81, 1998, Roskilde University.
- [20] Holland, J. H. (1962) “Outline for a logical theory of adaptive systems”. Journal of the Association for Computing Machinery, 3, 1962, pp. 297 - 314.
- [21] Holland, J. H. (1975) In Adaptation in Natural and Artificial Systems Holland . University of Michigan Press, Ann Arbor.

-
- [22] Laporte, G. (1992) The Traveling Salesman Problem: An overview of exact and approximate algorithms, Centre de recherche sur les transports, Université de Montréal, C.P. 6128, Station A, Montreal, Canada H3C M7.
- [23] Lin, S. & Kernighan, B. W. (1973) An Effective Heuristic Algorithm for the Traveling-Salesman Problem, *Oper. Res.* 21, 498-516 (1973).
- [24] Maniezzo, V et al. (1998) “A Bionomic Approach to the Capacitated p-Median Problem”, *Journal of Heuristics*, Vol 4, pp 263–280, Kluwer Academic Publishers, 1998.
- [25] Matai, R et al (2010) Traveling Salesman Problem: an Overview of Applications, Formulations, and Solution Approaches, *Traveling Salesman Problem, Theory and Applications*, Prof. Donald Davendra (Ed.), ISBN: 978-953-307-426-9, InTech.
- [26] Mitchell M. (1996) *An Introduction to Genetic Algorithms* . The MIT Press.
- [27] Mühlenbein, H. (1992) How genetic algorithms really work: 1. Mutation and hill-climbing. In R. Männer and B. Manderick, eds., *Parallel Problem Solving from Nature 2*. North-Holland.
- [28] Nugent, C.E. et al. (1968) An experimental comparison of techniques for the assignment of facilities to locations. *Ops. Res.*, 16, 150-173.
- [29] Osterman, C., Rego, C. (2003) *The Satellite List and New Data Structures for Symmetric Traveling Salesman Problems*. University of Mississippi.
- [30] Parson, R., Johnson, M.E. (1995) *DNA Sequence Assembly and Genetic Algorithms New Results and Puzzling Insights*. University of Central Florida. Orlando, FL US.
- [31] Prügel-Bennett, A., and Shapiro, J. L. (1994) An analysis of genetic algorithms using statistical mechanics. *Physical Review Letters* 72, no. 9: 1305-1309.
- [32] Reeves, C. (1993) *Modern Heuristic Techniques for Combinatorial Problems*.
- [33] Roberts, S.M., and Flores, B. (1966) An engineering approach to the travelling salesman problem . *Man.Sci.*, 13, 269-288.
- [34] Rosemberg, R.S. (1967) *Simulation of genetic population with biochemical properties* . Doctoral dissertation, University of Michigan.

- [35] Syswerda, G. (1989) Uniform crossover in genetic algorithms. In J. D. Schaffer, ed., Proceedings of the Third International Conference on Genetic Algorithms. Morgan Kaufmann.
- [36] Syswerda, G. (1991) A study of reproduction in generational and steady-state genetic algorithms. In G. Rawlins, ed., Foundations of Genetic Algorithms. Morgan Kaufmann.
- [37] Whitley, L. D. (1989) The Genitor algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In J. D. Schaffer, ed., Proceedings of the Third International Conference on Genetic Algorithms. Morgan Kaufmann.