G.A.M.P. Genetic Algorithm for Motion Planning

Nicola Carlon Matteo Finotto Francesco Fornasiero Giovanni Melis Marco Visonà

Tesina per il corso di Intelligenza Artificiale, Prof.ssa Silvana Badaloni
Università degli Studi di Padova, Facoltà di Ingegneria

A.A. 2008/2009

***Sommario* - G.A.M.P. è un software sviluppato in Java che cerca di risolvere problemi di motion planning in ambiente statico non conosciuto, utilizzando gli Algoritmi Genetici. Nell'implementazione sono state previste diverse varianti riguardanti in particolare il processo genetico (selezione, crossover e mutazioni) al fine di confrontarne le prestazioni e stabilire la miglior combinazione di scelte e di parametri per la soluzione del nostro problema.**

1 Introduzione

Gli Algoritmi Genetici (GA) [1] sono un tipo di algoritmi di *ricerca informata* che lavorano simulando il processo naturale evolutivo. Essi sono classificati tra gli algoritmi *meta-euristici*, definiti come un'evoluzione degli algoritmi di *ricerca locale* poiché si differenziano da questi ultimi per il fatto di aggiungere tecniche che permettano di evitare di terminare in ottimi locali. Gli algoritmi meta-euristici consentono di calcolare mosse peggiorative che permettano però di allontanarsi dagli ottimi locali e dirigersi auspicabilmente verso l'ottimo globale del problema.

I GA operano simulando il processo genetico di evoluzione per mezzo della selezione naturale, pertanto, per risolvere un problema computazionale, è necessario operare una mappatura dei concetti specifici del problema in questione in concetti specifici degli algoritmi genetici. Una soluzione ad un problema è costituita da un certo numero di *variabili* che la caratterizzano: essa viene dunque associata al concetto di individuo biologico (o meglio ai suoi cromosomi), mentre le variabili che la costituiscono sono i geni all'interno dei cromosomi. A partire da una popolazione iniziale di n individui, l'algoritmo opera ordinando gli individui dal migliore al peggiore, attraverso opportune politiche basate su un voto dato ad ogni individuo. Tale voto, detto *fitness*, riassume in sè la bontà dell'individuo a cui è associato ed è calcolato da una *fitness function*. Attraverso questa fase, chiamata *selezione*, gli individui migliori ottengono una probabilità di tramandare il loro corredo genetico maggiore rispetto a quella degli individui peggiori. A questo punto si entra nella fase di *crossover*, o ricombinazione, in cui si effettua la creazione di una nuova generazione figlia attraverso diverse tecniche di crossover genico. In accoppiata a questo meccanismo viene anche realizzata la possibilità che un individuo subisca, con una certa probabilità, una *mutazione* genetica. Evento sporadico che accade anche in natura, essa consente, nel parallelo algoritmico, di far variare gli individui in modo da allontanarsi da ottimi locali per esplorare altre parti dello spazio delle configurazioni delle variabili e sperabilmente dirigersi verso un ottimo globale. Una volta ottenuti n nuovi individui si itera nuovamente la procedura di selezione, crossover e mutazione appena descritti fintanto che non si raggiunge una condizione per cui la soluzione trovata è soddisfacente.

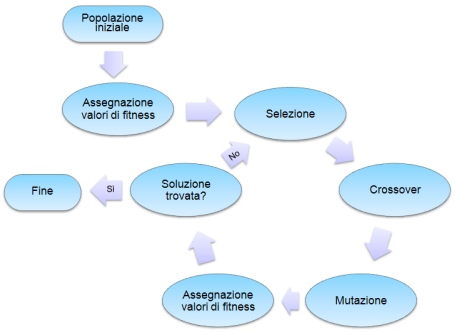

Figura 1.1: Processo evolutivo realizzato da un GA

I GA sono molto flessibili e per questo sono usati in moltissimi campi come l'electronic circuit design [2], i problemi di scheduling, gcc compiler optimization e molti altri. Il Motion Planning (MP) [3] è uno dei più importanti task dei robot autonomi. Questo serve a generare un percorso collision-free in un ambiente con ostacoli e ottimizzarlo secondo un particolare criterio. Più in generale l'ambiente è uno spazio n-dimensionale dove n è il numero di gradi di libertà del robot, chiamato spazio delle configurazioni (C-Space). Esistono molti algoritmi specifici che si adattano alle più diverse tipologie di ambiente: da conosciuto (cioè di cui si conoscono a priori le posizioni degli ostacoli) a non conosciuto; da strutturato (come una strada di città) a non strutturato (come un deserto), da dinamico se gli ostacoli si muovono a statico (se sono immobili), ecc... In generale le principali problematiche da affrontare nel MP sono la gestione della complessità computazionale e degli ottimi locali, nonché l'adattabilità degli algoritmi risolutivi alle diverse situazioni.

2 Il Problema

Il praradigma di programmazione che sfrutta gli Algoritmi Genetici può essere applicato alla risoluzione di diversi problemi computazionali: in questo paper ci occuperemo di motion planning su ambiente statico sconosciuto di un robot *non-olonomo* (car-like) [5] che si muove in uno spazio bidimensionale. Questa restrizione si può comunque generalizzare in un punto che si muove in un C-space.

È importante precisare che l'ambiente sia sconosciuto a priori e quindi al robot non è nota la posizione degli ostacoli. Tale agente conosce solo la propria distanza dal goal e dallo start, fornita ad esempio tramite un segnale GPS oppure, ipotesi meno stringente, da due radiofari la cui distanza può essere calcolata in termini di attenuazione dei segnali. L'obiettivo del robot è trovare la strada che lo conduca al goal, utilizzando un algoritmo genetico che costruisca percorsi che di volta in volta lo avvicinano sempre di più al suo obiettivo.

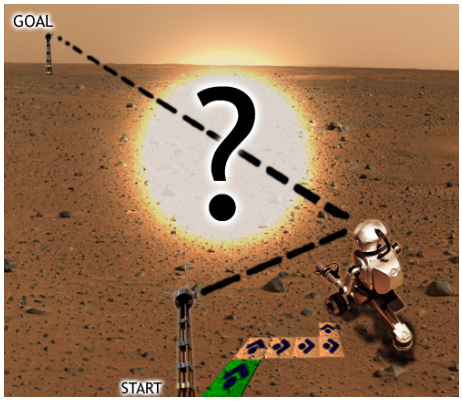

Figura 2.1: Descrizione del problema

Obiettivo del paper è descrivere la realizzazione del simulatore del processo genetico che porta alla creazione di una soluzione al problema descritto. In molti articoli della letteratura si parte da uno start-goal path che attraversa gli ostacoli (*infeasible*) e poi, con l'evoluzione dei GA, si cerca un path *feasible*, cioè privo di collisioni [6]. Il nostro approccio al problema è invece sostanzialmente diverso dagli altri. Come primo passo si è proceduto con una mappatura dei concetti generali propri dei GA nei concetti più specifici del problema in esame: l'*individuo* è un percorso che parte dalla posizione iniziale del robot (che da qui in poi chiameremo *start*), ovvero una soluzione candidata al problema. Una soluzione è quindi un individuo che va dallo start al goal.

Ogni individuo è costituito da *Geni* e questi, nella modellizzazione del problema, corrispondono ai movimenti ammessi dal robot, ovvero Avanti (A), Destra (D), Sinistra (S). Un percorso in definitiva avrà una struttura di questo tipo:

$$ADSDSDAAAASDADSSSDA$$

A causa della natura discreta dei geni utilizzati dai GA si è proceduto ad una tessellazione dello spazio bidimensionale con un *fixed decomposition* [5], in particolare con il metodo della *occupancy-grid* [7].

Figura 2.2: Movimenti atomici

Senza perdita di generalità si è considerato il robot della dimensione pari a quella di una cella e gli ostacoli estesi con il metodo della somma di Minkowski [8].

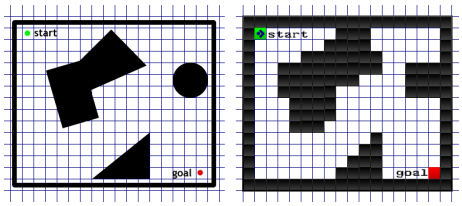

Figura 2.3: Fixed decomposition

Le scelte su come sono stati definiti ambiente e movimenti sono state dettate dallo studio dei movimenti di un robot in nostro possesso [10] in modo tale da poter testare fisicamente il programma.

Per poter affrontare il problema è tuttavia necessario definire altre questioni come l'implementazione di algoritmi per effettuare il *crossover*, la *mutazione* e la *selezione* dei cromosomi, nonché la definizione di una buona *Fitness Function*: la parte fondamentale dell'algoritmo. I prossimi paragrafi tratteranno l'implementazione di questi algoritmi.

3 Fitness Function

La fitness function (**FF**) è uno dei punti cruciali dei GA. Questa è un particolare tipo di funzione obiettivo nei problemi di ricerca. Per fare un parallelo naturale, la FF è ciò che regola la selezione sessuale, un voto che si assegna ad ogni individuo e che valuta le sue qualità: in base a questo voto l'individuo avrà una certa probabilità di riprodursi e perpetuare quindi il suo patrimonio genetico.

La FF deve essere definita in modo tale da assegnare un voto (peggiore) migliore all'individuo che si avvicinerà di (meno) più alla soluzione. In questo modo solo

gli individui che si avvicineranno di più alla soluzione avranno la possibilità di accoppiarsi.

Proprio per come è strutturato il problema, la FF è stata definita con il minimo nella soluzione. Per verificare che la FF sia adeguata bisogna innanzitutto verificare con alcuni test che nell'avanzare delle generazioni i voti degli individui siano sempre migliori, cioè che l'andamento della FF sia decrescente. Per facilitarne la visualizzazione e dare la possibilità di confrontare l'andamento su diversi ambienti è stato necessario effettuare una normalizzazione.

$$FF : I \longrightarrow [0, 1]$$

Il problema principale in questo tipo di definizione è dovuto alla forte presenza di ottimi locali, nel nostro caso ostacoli che costringono il robot ad allontanarsi momentaneamente dal goal. La FF deve avere quindi un certo grado di elasticità evitando di penalizzare troppo gli individui che si allontanano momentaneamente dalla soluzione. Purtroppo non avendo informazioni sulla posizione degli ostacoli, le variabili in gioco nella FF possono essere solo la distanza dal goal (Dg), la distanza dallo start (Ds) e il numero di passi effettuati (Ng). Queste distanze sono definite secondo le possibilità di localizzazione del robot e saranno più accurate se esso sarà dotato di dispositivo GPS (3.2) o meno, essendo disponibile solo un valore di distanza in linea d'aria calcolato grazie ai radiofari.

Per normalizzare la FF sono necessarie altre due informazioni: la distanza in linea d'aria dallo start al goal (Dm) e il numero di geni totali (Nt).

3.1 FF Semplice

Si è iniziato con la FF più semplice possibile:

$$FF = Dg$$

che normalizzata diventa:

$$FF = \frac{Dg}{Dm+Nt}$$

perché la massima distanza dal goal non può essere più grande del numero di geni totali dalla posizione dello start, mentre la minima è ovviamente 0 in caso di soluzione. Questa FF però non premia l'ottimalità della soluzione e ciò è inaccettabile in caso di grande differenza tra distanza minima e numero di geni totali.

Si è cercato quindi di premiare i percorsi brevi che terminano vicino al goal e penalizzare i percorsi lunghi che terminano vicino allo start.

$$FF = \frac{Dg+Ng}{Dm+Ds}$$

Normalizzata diventa:

$$FF = \frac{Dg+Ng}{Dm+Ds} \cdot \frac{Dm}{Dm+2 \cdot Nt}$$

Come si può immaginare questa FF funziona bene solo in caso di assenza di ostacoli, privilegiando percorsi brevi che si avvicinano rapidamente alla soluzione.

3.2 FF elastica

Per considerare anche percorsi lunghi, necessari per arrivare alla soluzione evitando gli ostacoli, è stato definito un secondo termine per la FF:

$$FF = \frac{Dg}{Ds+Ng}$$

Normalizzata diventa:

$$FF = \frac{Dg}{Ds+Ng} \cdot \frac{Nt}{Dm+Nt}$$

Questo termine privilegia i percorsi lunghi che però si allontanano dallo start, avvicinandosi al goal. La FF risultante avrà quindi due termini normalizzati e pesati in modo da poter decidere quanto privilegiare l'elasticità della FF a discapito dell'ottimalità e viceversa.

$$FF = \alpha \cdot \frac{Dg+Ng}{Dm+Ds} \cdot \frac{Dm}{Dm+2 \cdot Nt} + \beta \cdot \frac{Dg}{Ds+Ng} \cdot \frac{Nt}{Dm+Nt}$$

Per capire meglio il significato della FF è utile visualizzare il Fitness Landscape (FL) relativo ad una semplice mappa priva di ostacoli.

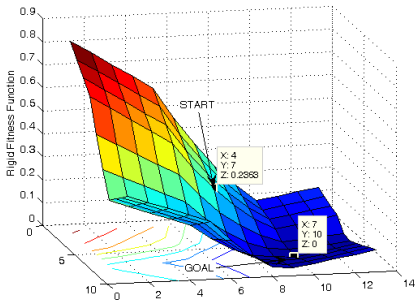

Figura 3.1: Fitness Landscape: Rigid FF.

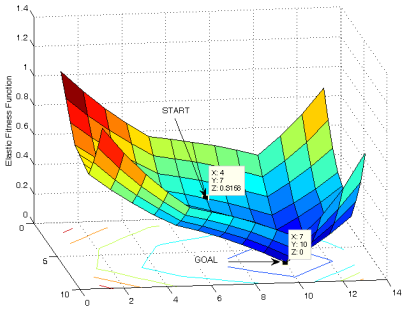

Figura 3.2: Fitness Landscape: Elastic FF.

L'FL non è altro che una funzione 3D che associa a ciascuna cella dell'ambiente il voto assegnato ad un ipotetico individuo con termine proprio su quella cella. Entrambi gli FL sono dipendenti anche dal numero di geni utilizzati dall'individuo per raggiungere la data cella, quindi l'FL (elastico) rigido sarà (meno) più ripido con l'aumentare dell'Ng proprio per (penalizzare) premiare i percorsi brevi. Le informazioni sulla localizzazione del robot sono cruciali per un buon esito dell'algoritmo.

Cenni implementativi Da questo momento in poi per poter implementare gli algoritmi della letteratura e per facilitare l'ordinamento tra individui sarà considerato il reciproco della FF, gestendo l'eccezione di divisione per zero in caso di soluzione. Dopo aver implementato la FF utilizzando la distanza in linea d'aria (grazie ai radiofari) si è introdotta la possibilità di munire il robot di un dispositivo GPS in grado di fornire differenze di latitudine e longitudine rispetto allo start ed al goal, che chiameremo rispettivamente Dg_x, Dg_y , Ds_x e Ds_y. La FF è stata poi ridefinita con $Dg = 2 \cdot max(Dg_x, Dg_y)$, $Ds = 2 \cdot max(max(Ds_x, Ds_y), Ds)$.
Questo permette una valutazione più accurata della FF riuscendo a risolvere ambienti con molti ottimi locali.

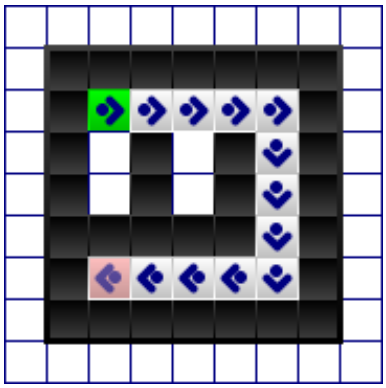

Figura 3.3: Mappa con ottimi locali.

Fornire la possibilità di decidere il grado di elasticità della FF permette di inserire un certo grado di conoscenza a priori dell'ambiente: se ci si aspetta un grande numero di ostacoli ed un ambiente con una struttura simile ad un labirinto si imposterà quindi una FF molto elastica precludendo però la possibilità di trovare una soluzione ottima in caso di errata valutazione dell'ambiente; viceversa se ci si aspetta un ambiente abbastanza libero da ostacoli scegliendo una FF rigida la soluzione si avvicinerà più velocemente alla soluzione ottima del percorso più breve.

4 Selezione

L'operazione di selezione è quella che, nel parallelo biologico, corrisponde alla cosiddetta *selezione naturale* [11]. La natura fornisce agli individui più adattati all'ambiente la possibilità di avere vita più lunga e maggiori probabilità di riprodursi per diffondere il proprio corredo genetico. In modo analogo, un algoritmo genetico, data una popolazione, deve scegliere gli individui migliori e farli accoppiare (o clonare) in modo da ottenere soluzioni più buone delle precedenti, ereditando sperabilmente i geni positivi dai genitori. Questo processo di selezione avviene sulla base della fitness function e consiste essenzialmente nel riempimento del cosiddetto *mating pool*: un bacino in cui vengono inseriti temporaneamente gli individui più promettenti al fine della riproduzione. Esso deve avere la stessa dimensione della popolazione iniziale e può perciò prevedere la presenza di cloni, cioè che un individuo venga inserito più volte. I cloni tuttavia, nella successiva fase di crossover, vengono trattati come se fossero individui distinti. È chiaro che per simulare la selezione naturale, gli individui con migliore valore di fitness devono essere inseriti nel mating pool con maggiore probabilità.

In letteratura esistono diverse tecniche per la selezione, ciascuna delle quali si adatta più o meno efficacemente al problema da risolvere per mezzo dell'algoritmo genetico. Sono state scelte le tre più comuni: la selezione proporzionale alla fitness, quella a torneo e quella basata su un certo rank dato agli individui.

4.1 Selezione proporzionale alla fitness

È lo schema più usato e prevede di assegnare a ciascun individuo una probabilità di essere inserito nel mating pool direttamente proporzionale al suo valore di fitness, normalizzato rispetto alla somma estesa su tutti i valori di fitness della popolazione in oggetto. Una normalizzazione siffatta permette di leggere direttamente i valori di fitness normalizzata di un individuo esattamente come il valore di probabilità da assegnargli. In formule la probabilità dell'individuo i è calcolata come segue [11]:

$$P_i = \frac{F_i}{\sum_{j=1}^{n} F_j}$$

dove F_i è il valore di fitness. Sorge, a questo punto della trattazione, un problema pratico: come fare a scegliere fra un gruppo di oggetti tenendo conto che ciascuno di essi ha una probabilità di essere scelto diversa dagli altri?

Gli oggetti su cui lavorare sono individui ai quali è assegnato un valore di probabilità. L'unico vincolo che è possibile sfruttare deriva dalla normalizzazione (e dalla definizione di probabilità): la somma delle probabilità associate agli individui deve convergere a 1. Se interpretassimo la probabilità come un'ampiezza, potremmo rappresentarla come un settore di corona circolare di dimensione proporzionale alla probabilità ad esso associata. L'intera corona circolare rappresenta tutto il range di valori di probabilità $[0, 1]$ e se generassimo un numero casualmente in tale range, potremmo facilmente simulare l'estrazione dell'oggetto i se il numero cade nel settore circolare a lui corrispondente. Questo meccanismo viene detto comunemente *roulette* per la somiglianza col famoso gioco.

$P(i)$	$P_c(i-1)$	$P_c(i)$	S_i
0.1	0	0.1	$S_1 = [0, 0.1)$
0.2	0.1	0.3	$S_2 = [0.1, 0.3)$
0.3	0.3	0.6	$S_3 = [0.3, 0.6)$
0.4	0.6	1	$S_4 = [0.6, 1)$

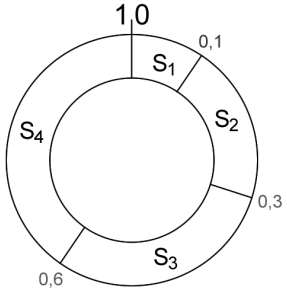

Figura 4.1: Rappresentazione della roulette per l'esempio sintetizzato in tabella

Cenni implementativi A livello implementativo la roulette è stata realizzata in G.A.M.P. usando il concetto di probabilità cumulativa $P_c(i)$ definita ricorsivamente come segue:

$$P_c(i) = \begin{cases} P(i), & i = 1 \\ P(i) + P_c(i-1), & i \geq 2 \end{cases}$$

ove $P_c(i)$ è la probabilità cumulativa dell'oggetto i, mentre $P(i)$ è la probabilità (semplice) relativa all'oggetto i. In questo modo si ottiene per ogni individuo i un range di probabilità (corrispondenti ai valori estremi dei settori circolari) così costruito: $S_i = [P_c(i-1), P_c(i))$ (facendo attenzione a fissare $P_c(i-1) = 0$ per $i = 1$ e $P_c(i) = 1$ per $i = n$, se n è il numero degli individui). Con una griglia di intervalli siffatta, una volta generato un numero random fra 0 e 1, sarà sufficiente stabilire a quale range appartiene per selezionare l'oggetto corrispondente. Si noti che nel meccanismo della roulette così implementato, non ha alcuna importanza l'ordine con cui compaiono gli intervalli degli oggetti all'interno del range $[0, 1]$, perché quello che conta è l'ampiezza degli intervalli, non la loro posizione relativa, visto che il numero pseudo-casuale è generato in modo uniformemente distribuito in $[0, 1]$.

4.2 Selezione a torneo

La *tournament selection* [11] simula un torneo eseguendo una selezione a gironi e prevede diverse varianti. Quella usata in G.A.M.P. stabilisce con un'opportuna funzione la dimensione di ciascun girone (k) in base alla dimensione della popolazione iniziale (n). Ciascun girone viene creato scegliendo in modo casuale k individui dalla popolazione in oggetto. Successivamente viene individuato il miglior individuo del girone confrontando i valori di fitness di tutti gli individui interni al girone e inserito nel mating pool. Creando con questo procedimento n gironi si otterrà un bacino di accoppiamento di n individui, coerentemente col vincolo di mantenere inalterato il numero di individui per l'intero processo genetico. Anche in questo caso è probabile la presenza di cloni all'interno del mating pool, dato che gli individui con buoni valori di fitness function possono risultare i vincitori di più gironi.

Vale la pena notare che la rappresentatività dei gironi dipende dalla loro dimensione. In letteratura non ci sono formule che permettano di calcolare un'ampiezza di girone ottimale. Questo parametro infatti è fortemente dipendente dalla numerosità della popolazione in esame e può variare di molto da problema a problema. Nella fattispecie G.A.M.P. implementa una tecnica che assegna opportune dimensioni di girone a diversi range di numerosità di popolazione. Se non si operasse in questo modo, infatti, si rischierebbe di ricadere in alcuni casi limite: gironi composti da un solo individuo equivalgono a scegliere completamente a caso gli individui da inserire nel mating pool, non eseguendo così una buona selezione; viceversa, impostare una dimensione di girone pari a metà della dimensione della popolazione restituirebbe un bacino di riproduzione con troppe ripetizioni di pochi individui, perdendo così di variabilità genetica.

Cenni implementativi Poiché la dimensione dei gironi dipende dalla numerosità della popolazione in esame e lavorando in genere con popolazioni non troppo numerose (limite massimo fissato sperimentalmente a 50 individui), è stata usata la seguente funzione definita per casi per fissare l'ampiezza dei gironi stessi:

$$k = f(n) = \begin{cases} 2, & n < 10 \\ 3, & 10 \leq n < 20 \\ 4, & 20 \leq n < 40 \\ 5, & n \geq 40 \end{cases}$$

ove k è l'ampiezza del girone e n è la dimensione della popolazione.

4.3 Selezione basata sul rank

L'idea di base della *rank selection* [11] è quella di eseguire un ordinamento della popolazione secondo i valori di fitness function degli individui e successivamente assegnare una probabilità dipendente dall'ordine (rank), pesato secondo una funzione strettamente crescente (o decrescente) a seconda di com'è stato eseguito il sorting.

Sarebbe sbagliato pensare che scegliendo una funzione proporzionale per pesare gli individui si ottenga come caso degenere la tecnica di selezione proporzionale. È vero infatti che l'ordinamento mantiene l'ordine degli individui secondo i loro valori di fitness, ma l'assegnazione della probabilità è basata sul solo rank e quindi non tiene più conto delle differenze (in valore assoluto) dei valori di fitness degli individui.

Cenni implementativi In G.A.M.P. è stata usata una funzione di ranking di tipo parabolico, in quanto quella esponenziale risultava eccessivamente selettiva e

riduceva il mating pool alla ripetizione di pochi individui. Nella fattispecie, la probabilità di essere inserito nel mating pool per il generico individuo i è così calcolata:

1. Calcolo del peso di ciascun individuo secondo la seguente funzione parabolica:

$$W_i = 1 + r_i^2$$

ove r_i è la posizione (rank) dell'individuo i.

2. Trasformazione del peso in una probabilità, normalizzando ciascun peso rispetto alla somma di tutti i pesi:

$$P_i = \frac{W_i}{\sum_{j=1}^{n} W_j} = \frac{W_i}{W_{tot}}$$

L'ordinamento invece è stato eseguito con l'algoritmo *merge sort* (versione iterativa) che garantisce prestazioni $O(n \log n)$ sulla dimensione dell'istanza ed è considerato uno degli algoritmi di sorting più efficienti.

5 Crossover

Una volta selezionati gli individui statisticamente più promettenti, si può applicare la fase di crossover (o incrocio): si prelevano casualmente due individui per poi ricombinarli in modo da generare una nuova coppia. L'operazione viene ripetuta fino al completo svuotamento del mating pool. Il crossover è dunque un operatore binario che opera una trasformazione sulla struttura degli individui. L'idea verso la quale si vorrebbe tendere è quella in cui gli individui figli ereditano le migliori caratteristiche dei due genitori. In generale si nota che il valore medio della fitness delle nuove generazioni tende ad essere migliore di quello delle generazioni precedenti, grazie al fatto che la selezione premia gli individui migliori.

Dalle nostre simulazioni abbiamo notato che è preferibile eseguire il crossover con una probabilità pari al 90%. Nei casi in cui il crossover non viene applicato, i figli sono semplicemente una copia dei genitori, dando loro la possibilità di propagare completamente i propri geni nella generazione successiva.

In base al campo di applicazione dell'algoritmo genetico, si possono definire diversi tipi di crossover.

Crossover a un punto Scelta la coppia di individui, viene fissato casualmente un punto di taglio che divide in due parti ciascun cromosoma e si scambiano le due code.

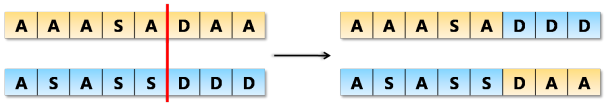

Figura 5.1: Crossover a un punto

Questo crossover tratta alcune parti della sequenza in modo preferenziale: i segmenti scambiati dai due genitori contengono sempre la parte finale delle due stringhe.

Crossover a due punti In questo caso si selezionano due punti di taglio, suddividendo ciascun individuo in tre parti. Per generare i nuovi figli, i due genitori si scambiano la parte di sequenza compresa tra i due tagli.

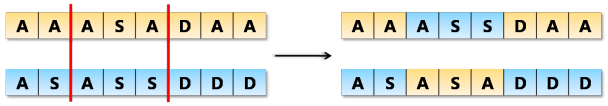

Figura 5.2: Crossover a due punti

Il crossover a due punti riesce a creare molte più combinazioni rispetto al crossover a un punto. Inoltre, i segmenti che vengono scambiati non contengono necessariamente la parte finale delle stringhe.

Crossover uniforme Il crossover uniforme può essere visto come una generalizzazione del crossover a due punti, nel caso in cui i punti di taglio siano $N - 1$, dove N indica la lunghezza del cromosoma. In questo modo i cromosomi dei genitori vengono fusi a livello di gene invece che a livello di segmento. Viene usato un parametro globale, detto *mixing ratio*, che indica, per ogni gene di un discendente, la probabilità che tale gene provenga dal primo o dal secondo individuo. Abbiamo fissato tale valore al 50%, in modo che ogni gene abbia la stessa probabilità di essere scelto da uno dei due genitori.

Sono state implementate due varianti del crossover uniforme: un primo metodo sceglie per ogni posizione del primo discendente qual è il genitore che fornisce il gene. Il secondo discendente riceve il gene dall'altro genitore.

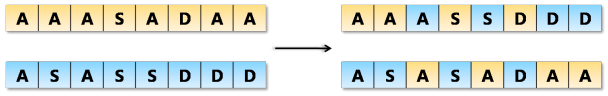

Figura 5.3: Crossover uniforme (1)

Un'altra possibilità genera il secondo figlio indipendentemente dal primo, permettendo in tal modo ad un genitore di fornire lo stesso gene ad entrambi i discendenti.

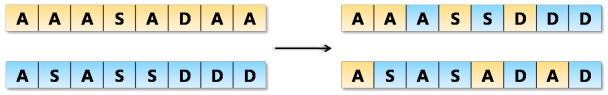

Figura 5.4: Crossover uniforme (2)

Considerazioni

Il crossover è uno degli operatori principali che distingue gli algoritmi genetici dagli altri metodi stocastici, ma il suo ruolo deve ancora essere compreso pie-

namente. Sono in corso numerosi studi che cercano di trovare sotto quali condizioni il crossover consente di ricombinare sequenze di cromosoma per generare individui migliori e sotto quali altre condizioni il crossover rappresenta semplicemente una macro-mutazione, utile per spostarsi maggiormente all'interno dello spazio delle soluzioni, rispetto ad una mutazione semplice.

Negli anni sono state suggerite numerose altre tecniche di crossover, ottenute variando il numero dei tagli o assegnando distribuzioni di probabilità non uniformi per decidere in quale porzione del cromosoma effettuare il crossover.

Il dibattito su quale sia la tecnica migliore è ancora in corso. Sono stati effettuati studi sull'efficienza del crossover multipoint, arrivando a preferire in generale il crossover a due punti: infatti l'aggiunta di molti punti di taglio distrugge le sottosequenze utili, però allo stesso tempo avere molti punti su cui fare crossover consente una ricerca più accurata nello spazio delle soluzioni. Sotto questo punto di vista il crossover uniforme sembra essere la tecnica più robusta.

Il successo o il fallimento di un particolare tipo di crossover dipende comunque da vari fattori, come il particolare campo di utilizzo, la funzione di fitness, la codifica degli individui. Rimandiamo questa discussione al paragrafo 8.

6 Mutazioni

In genetica le mutazioni sono una variazione casuale del DNA di un individuo. Assieme alla ricombinazione genica, tramite ad esempio il crossover, operano un ruolo molto importante per conservare la variabilità genetica: la condizione per cui all'interno della stessa popolazione vi sono individui simili con patrimonio genetico differente.

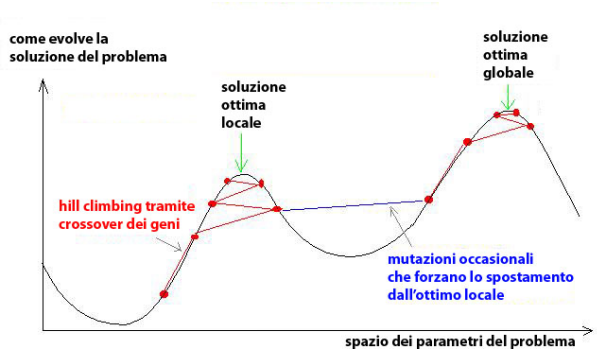

Figura 6.1: Ruolo del crossover e delle mutazioni

Per la casualità intrinseca nelle mutazioni, non si può conoscere a priori quale sia l'individuo della popolazione su cui avvengono e su quale gene dello stesso agiscono. In questo modo si modifica il patrimonio genetico che a fronte della selezione naturale può portare ad un peggioramento o miglioramento delle caratteristiche, nel nostro caso una diminuzione o un aumento del valore della *fitness function*. Questo ci permette di affermare che le mutazioni consentono di uscire dai punti di ottimo locale in cui la soluzione potrebbe tendere a seguito del crossover ma, per contro, ci potrebbero far allontanare dalla stessa.

Cenni implementativi Nel G.A.M.P. è possibile impostare la probabilità con cui avvengano le mutazioni che tipicamente variano nell'intervallo 0,1-1%.

Nei test effettuati è stata utilizzata una percentuale del 10% perchè una probabilità di mutazione più bassa tendeva ad appiattire la variabilità dei vari individui che si uniformavano e il processo si bloccava in un ottimo locale.

Per ogni individuo si calcola un numero casuale e lo si confronta con il valore di probabilità impostata dall'utente. Se il valore è inferiore o uguale si esegue una delle due possibili mutazioni che sono state implementate e che è possibile scegliere dall'apposito menù:

- *simple mutation* in cui all'individuo selezionato si modifica un gene scelto a caso nel suo patrimonio, ad esempio se il gene fosse 'A' potrebbe essere sostituito casualmente con 'D' o 'S'.

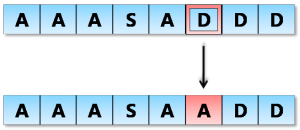

Figura 6.2: Simple mutation

- *uniform mutation* in cui si inverte l'ordine di due geni scelti casualmente nello stesso individuo.

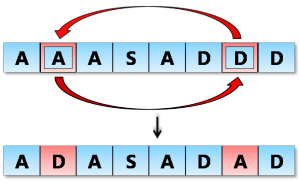

Figura 6.3: Uniform mutation

7 Software

Il software è stato realizzato in Java, scelta dettata dalla rodata conoscenza del linguaggio e dalla presenza di librerie di comunicazione con il robot sviluppate con la versione 1.4 del sopracitato linguaggio [10]. L'applicativo possiede un'interfaccia grafica grazie alla quale è possibile creare l'ambiente su cui fare muovere il robot. Il tutto è configurabile in tre modi.

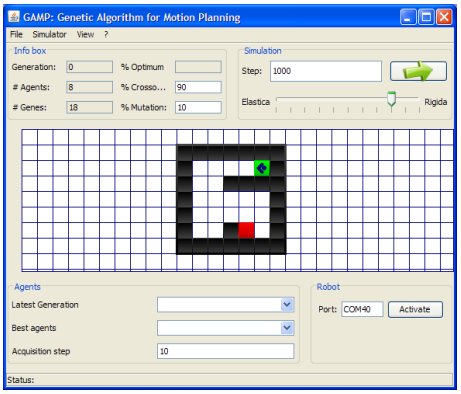

Figura 7.1: Screenshot di GAMP, il software sviluppato

Il primo consente di specificare nel dettaglio l'ambiente attraverso semplici click del mouse per posizionare lo start, il goal e gli ostacoli, il secondo consiste in una creazione automatica casuale, mentre il terzo consiste nella creazione di ambienti con uno specifico pattern selezionabile da un menu a tendina. Il primo pattern è una spirale che si avviluppa dall'esterno della mappa verso il goal posto al centro della spirale stessa, mentre il secondo è un corridoio a serpentina che collega start e goal posti in posizioni diametralmente opposte rispetto alla mappa.

Dalla stessa schermata è possibile anche inserire il numero di individui della popolazione e il numero di geni di un singolo individuo. È possibile definire sia un *upper bound* che un *lower bound* al numero di geni di un individuo adatto ad uno specifico ambiente (implementato come una matrice bidimensionale $n \cdot m$):

$$UB = n \cdot m$$

$$LB = |x_s - x_g| + |y_s - y_g|$$

dove x_s e y_s sono ascissa e ordinata dello *start* mentre x_g e y_g sono ascissa e ordinata del *goal*. Una volta creato l'ambiente si ha accesso alla schermata di simulazione in cui viene lanciato il processo genetico vero e proprio.

La simulazione avviene per *step* successivi, ovvero è data all'utente la possibilità di simulare un certo numero di generazioni consecutive a sua scelta, visualizzarne il risultato e procedere successivamente con la simulazione di una nuova serie di generazioni a partire dall'ultima trovata, fino al raggiungimento della soluzione. Gli individui ottenuti possono essere visualizzati attraverso i due menu a tendina posti sul fondo della schermata. Il primo consente di visualizzare tutti gli individui creati nell'ultima generazione eseguita, mentre l'altro consente di visualizzare i migliori individui ogni Δ generazioni, dove Δ può essere inserito dall'utente nel campo di testo sottostante.

È possibile eseguire simulazioni variando diversi parametri, come i tipi di crossover, di mutazione e di selezione; nonché le probabilità di *mutazione* e di *crossover*

Un'interessante feature è la possibilità di visualizzare il percorso ottimo (utilizzando in questo caso l'informazione a priori sulla posizione degli ostacoli) e vedere, in termini percentuali, di quanto la soluzione trovata con il GA è peggiore. Infine è possibile connettere il robot per testare dal vivo un certo individuo.

8 Prestazioni ottenute

La bontà di un algoritmo genetico è influenzata da una serie di parametri che caratterizzano le varie componenti di cui è composto. Sono stati da noi eseguiti numerosi test di misurazione delle performance dell'algoritmo di MP al variare delle seguenti feature: tipo di crossover, tipo di selezione, tipo di mutazione, tipo di fitness function, numero di geni e di individui.

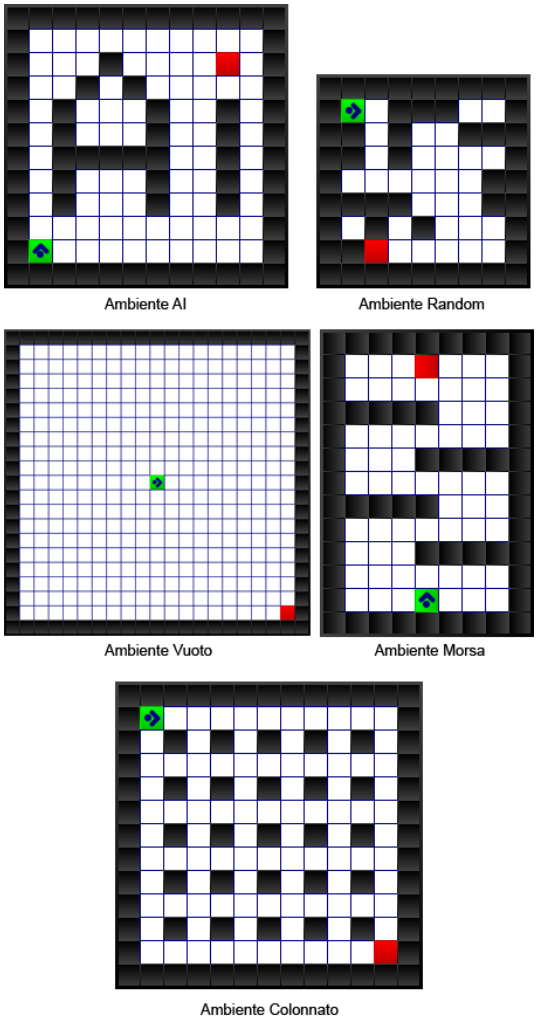

Figura 8.1: Ambienti di test delle performance.

Dopo aver deciso che la misura di prestazione più significativa per l'algoritmo sarebbe stato il numero di generazioni calcolate prima di ottenere un percorso va-

lido dallo start al goal, sono stati disegnati 5 ambienti di test:

- **Vuoto**: ambiente vuoto di grandi dimensioni (21×21) privo di ostacoli,
- **Random**: ambiente generato casualmente di dimensioni 9×9 con una quantità di ostacoli che ricopre il 35% dell'area disponibile,
- **Morsa**: ambiente di dimensioni 9×13 che costringe il robot a muoversi a zig-zag attraverso uno stretto corridoio,
- **Colonnato**: ambiente di dimensioni 13×13 che alterna spazi liberi ad ostacoli,
- **AI**: ambiente di dimensioni 12×12 con ostacoli disposti a formare le lettere A e I dotato di ottimo locale in corrispondenza della base della lettera A.

A causa della natura stocastica dell'algoritmo si è rivelato necessario, per ogni configurazione dei parametri di test, effettuare 5 prove e tenere la media delle stesse, in modo tale da avere una misura più accurata delle prestazioni. Per ogni mappa sono state eseguite in batch 960 prove che hanno condotto ai risultati esposti nel seguito.

L'algoritmo si comporta bene nell'ambiente Vuoto in cui i parametri che maggiormente fanno variare le prestazioni sono la tipologia di crossover e il numero di individui della popolazione. I migliori risultati per questa mappa, mediati sulle 5 prove per ogni configurazione, si ottengono con il metodo di crossover OneUniform e con una popolazione di 50 individui in cui il numero di generazioni richieste si attesta a 2,8. Tuttavia ottimi risultati si ottengono anche con gli altri tipi di crossover mantenendo fisso il numero di individui: tali valori variano tra 3 e 4,3 generazioni. La media calcolata tenendo fisso il numero di individui a 10 e facendo variare tutti gli altri parametri risulta pari a 144 generazioni ed è la peggiore performance media riscontrata su questo ambiente.

La mappa Random ha la peculiarità di avere una elevata quantità di ostacoli ed alcuni ottimi locali. Anche in questo caso le performance sono determinate dal numero di individui e dalla tipologia di mutazione, a prova del fatto che le mutazioni influiscono molto sul comportamento in presenza di ottimi locali. La configurazione vincente in questo ambiente si ottiene con 50 individui e mutazioni semplici, per contro invece la peggiore è con 10 individui e mutazione uniforme in cui si ottiene una misurazione di 4412 generazioni. Tutte le altre configurazioni attestano le performance in un range che va da circa 1650 generazioni a circa 2550.

Nella mappa Morsa e nella mappa Colonnato si ottiene, nella media delle 5 prove, la soluzione con un minor numero di generazioni utilizzando la selezione basata sul rank, con il crossover OneUniform e la mutazione semplice. Per la prima di queste due mappe, il numero di generazioni si attesta in media a 4355 per 10 individui e per 50 individui si dimezza a 2224; nella seconda con 10 individui si raggiungono 1457 generazioni e 300 generazioni con 50 individui. La peggior configurazione per la Morsa è l'utilizzo della selezione a torneo con il crossover ad un punto e la mutazione uniforme che ottiene soluzione alla 7020 generazione. Per la mappa Colonnato la peggior configurazione avviene usando la selezione proporzionale con il crossover a due punti e mutazione uniforme il cui numero di generazioni è di 2408.

Per la mappa AI il miglior modello si ottiene imponendo la selezione basata sul rank con crossover ad un punto e mutazione semplice. Rispetto alle altre mappe il tipo di Fitness Function non è un fattore caratterizzante per la determinazione della migliore soluzione dato che il numero di generazioni, mantenendo fissa la modalità di Fitness Function, sono di 437 per quella elastica e 485 per quella rigida. Il range di valori selezionando 10 individui varia tra 382 e 1623; per 50 individui l'intervallo è tra 31 e 310 generazioni.

È significativo osservare che l'inserimento di un numero elevato di individui consente di avere a disposizione, nelle varie generazioni, maggior materiale genetico e quindi poter raggiungere la soluzione in modo più rapido. La stessa cosa però non è valida per quanto riguarda il numero di geni il cui elevato numero nei vari individui potrebbe ritardare, sempre rispetto al numero di generazioni, il raggiungimento della soluzione.

È tuttavia importante sottolineare che, data la natura stocastica dell'algoritmo, le diverse prove fatte con uguale configurazione dei parametri possono restituire risultati molto diversi tra loro, nei casi estremi addirittura di un ordine di grandezza.

9 Conclusioni

9.1 Soluzione ibrida

La navigazione in ambiente sconosciuto è un campo molto complesso a causa della possibilità di presenza massiccia di minimi locali dai quali bisogna allontanarsi molto per raggiungere la soluzione. In questi casi un algoritmo privo di memoria come i GA difficilmente porterà ad una soluzione. Per questo motivo è stata introdotta una soluzione alternativa che prende spunto dai Virus Evolutionary Genetic Algorithm (VEGA). Ogni volta che un individuo incontra un ostacolo segnala quell'ostacolo alle generazioni sucessive; gli individui destinati ad andarci contro a causa del loro patrimonio genico verranno quindi infettati e il gene che li porterebbe alla collisione viene sostituito con uno casuale. In questo modo la posizione degli ostacoli viene tramandata nelle generazioni, facendo convergere l'algoritmo alla soluzione in tempi molto brevi. Dal punto di vista realizzativo sarebbe come se un robot principale avesse a disposizione uno sciame di robot esploratori che me-

diante GA esplorano l'ambiente e comunicano al robot principale la posizione degli ostacoli.

Riferimenti bibliografici

[1] D.E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.

[2] J.D. Lohn, G.S. Hornby, D.S. Linden, *An Evolved Antenna for Deployment on NASA's Space Technology 5 Mission*, NASA.

[3] M. De Berg, M. Van Krefeld, M. Overmars, O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, Springer.

[4] M. Mitchell, *An Introduction to Genetic Algorithms*, MIT Press, Cambridge, 1998.

[5] R. Siegwart, I.R. Nourbakhsh, *Introduction to Autonomous Mobile Robots*, MIT Press, Boston, 2004.

[6] Y. Hu, S.X. Yang, L.-Z. Xu and Max Q.-H. Meng, *A Knowledge Based Genetic Algorithm for Path Planning in Unstructured Mobile Robot Environments*, IEEE International Conference on Robotics and Biomimetics, 2004.

[7] H.P. Moravec, A. Elfes, *High Resolution Maps from Wide Angle Sonar*, The Robotics Institute Carnegie-Mellon University.

[8] J.C. Latombe, *Robot Motion Planning*, Springer.

[9] S.M. LaValle, *Planning Algorithms*, Cambridge University Press, 2006.

[10] DeAgostini, *I-Droid*, http://www.i-d01.deagostini.it

[11] S. Cagnoni, R. Poli, *Genetic and Evolutionary Computation*, Intelligenza Artificiale (periodico trimestrale dell'Associazione Italiana per l'Intelligenza Artificiale), Anno III, N°1/2, Marzo-Giugno 2006.