Indice

Introduzione

La *Swarm Intelligence (intelligenza dello sciame)* è un termine coniato per la prima volta nel 1988 da *Gerardo Beni*, *Susan Hackwood* e *Jing Wang*, in seguito a un progetto ispirato ai sistemi robotici. [1]

Esso prende in considerazione lo studio dei sistemi auto-organizzati, nei quali un'azione complessa deriva da un'intelligenza collettiva, come accade in natura nel caso di colonie di insetti o stormi di uccelli, oppure banchi di pesci, o mandrie di mammiferi.

All'interno di uno sciame, ciascuna decisione è presa in modo decentralizzato, ossia non dettata da un individuo più importante degli altri, ma attuata sulla base di informazioni locali ottenute dall'interazione di ciascun individuo sia con l'ambiente sia con altri individui.

Un'altra caratteristica distintiva di questi sistemi è costituita dai limiti delle capacità e delle conoscenze dei singoli. Non è infatti necessario che ogni singola unità conosca la struttura globale entro cui è collocata, e per lo stesso motivo nessuna di esse svolge mansioni di supervisione sulle altre unità. Ciascuna si limita ad adottare una serie di comportamenti, che a loro volta influenzano le azioni delle altre, risultando infine nell'auto-organizzazione dell'intero sistema. [3]

Questi fenomeni sono d'ispirazione nella creazione di diverse soluzioni metaeuristiche volte a risolvere problemi di ottimizzazione, le quali risultano molto efficaci quando si ha a che fare con funzioni particolarmente irregolari.

Una di queste soluzioni è proprio *l'Algoritmo Firefly*, il quale si ispira al comportamento di uno sciame di lucciole, in particolar modo al loro modo di comunicare basato sulla *bioluminescenza*.

In questo lavoro di tesi è stato implementato e valutato un Algoritmo Firefly, confrontando le sue prestazioni con un Algoritmo Genetico.

Tutti gli Algoritmi sono stati implementati in Python, sfruttando principalmente la libreria *DEAP*, e testati su diverse funzioni di benchmark.

Nel Capitolo 1 viene discusso il significato di problema di ottimizzazione, insieme ad alcuni esempi.

Nel Capitolo 2 viene approfondito il concetto di Swarm Intelligence, con particolare attenzione all'Algoritmo Firefly.

Nel Capitolo 3 si discute del framework DEAP, della struttura generale e dei vantaggi.

Nel Capitolo 4 infine vengono presentate le funzioni di benchmark utilizzate e poi illustrati i risultati ottenuti dai vari test.

Nell'Appendice infine viene discussa la struttura degli Algoritmi Genetici.

1 Problema di ottimizzazione

Un problema di ottimizzazione è un problema che richiede di identificare una o più soluzioni che risultino ottimali rispetto ad un qualche criterio di valutazione. Un problema così fatto equivale, fissato un opportuno dominio, a massimizzare o minimizzare una funzione, detta *funzione obiettivo*.

Dunque il problema consiste nella ricerca del punto di ottimo x^*, ossia di minimo o massimo, della funzione obiettivo $f : D \subseteq R^n \to R$, dove D è il dominio. Questo genere di problemi sono molto più comuni di quanto si pensi.

Di seguito alcuni esempi di problemi di ottimizzazione:

1.1 Problema del commesso viaggiatore (TSP)

Questo problema può essere formulato nel modo seguente: *"Dato un insieme di città, e note le distanze tra ciascuna coppia di esse, trovare il tragitto di minima percorrenza che un commesso viaggiatore deve seguire per visitare tutte le città una ed una sola volta e ritornare alla città di partenza".*

Al problema del *TSP* è associabile un grafo $G = (V, A)$, in cui V è l'insieme degli n nodi (o città), mentre A è l'insieme degli archi (o strade). Detta x_{ij} la generica variabile binaria tale che $x_{ij} = 1$ se l'arco *(i,j)* appartiene al percorso seguito, e $x_{ij} = 0$ altrimenti, una possibile formulazione matematica del problema è:

$$z = min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{1.1}$$

Dove la relazione 1.1 è la funzione obiettivo che rappresenta la minimizzazione del costo del cammino.[11]

1.2 Problema della brachistocrona

Bisogna far giungere un grave di massa M da un punto iniziale (ad esempio l'origine come nell'immagine 1) a un punto finale (il punto di coordinate (a, h)) nel minor tempo possibile, ricordando che il grave è soggetto a forza peso. Qual è il percorso migliore che minimizzi il tempo impiegato?

Il tempo che M impiega per andare dal punto inziale a quello finale, con velocità iniziale nulla, dipende dalla traiettoria, ed è dato dal funzionale:

$$T = \int_0^a \frac{\sqrt{1 + (\frac{dz}{dx})^2}}{\sqrt{2gz(x)}} dx \tag{1.2}$$

La curva che permette di percorrere la traiettoria nel minor tempo possibile, è quella che minimizza tale funzionale. Tale curva (segnata in rosso nella figura 1) è una cicloide ed è chiamata *brachistocrona*. [16]

1.3 Il problema di Plateau

Inizialmente proposto da *Lagrange* nel 1760, prende il nome da *Joseph Plateau* che fece esperimenti su di esso utilizzando bolle di sapone. Il problema può essere formulato in questo modo: *data una curva $C \subset R^3$, regolare chiusa e non annodata, cercare (se esiste) una superficie S di area minima tra quelle che hanno C come bordo.* Tutte le superfici di area minima sono una soluzione di un qualche problema di Plateau. [5]

Plateau nei suoi studi fece largo uso delle lamine saponate che si possono ottenere immergendo un reticolo di ferro in una soluzione di acqua e sapone, per poi estrarlo delicatamente (vedi figura 2). Se l'esperimento è ben eseguito, si ottiene una lamina saponata che ha come bordo lo stesso reticolo. Questa lamina rappresenta la superficie di area minima per quel reticolo, questo perché la tensione superficiale della lamina saponata tende a ridurne il più possibile l'estensione, finché essa non si trova allo stato di energia minima. Il nome *superficie minima* fu dato da *Lagrange* nel 1760 a quelle superfici che sono soluzioni di un problema variazionale, più precisamente punti critici del funzionale d'area. Grazie al suo lavoro, egli diede il primo esempio di superfici minime: il *catenoide*, mostrato in figura 3, la cui equazione è:

$$f(x, y) = cosh^{-1}(\sqrt{x^2 + y^2}) \tag{1.3}$$

Il catenoide si ottiene facendo ruotare una particolare curva, detta *catenaria*, intorno ad un asse che non la interseca e che sia perpendicolare all'asse di simmetria della catenaria stessa.

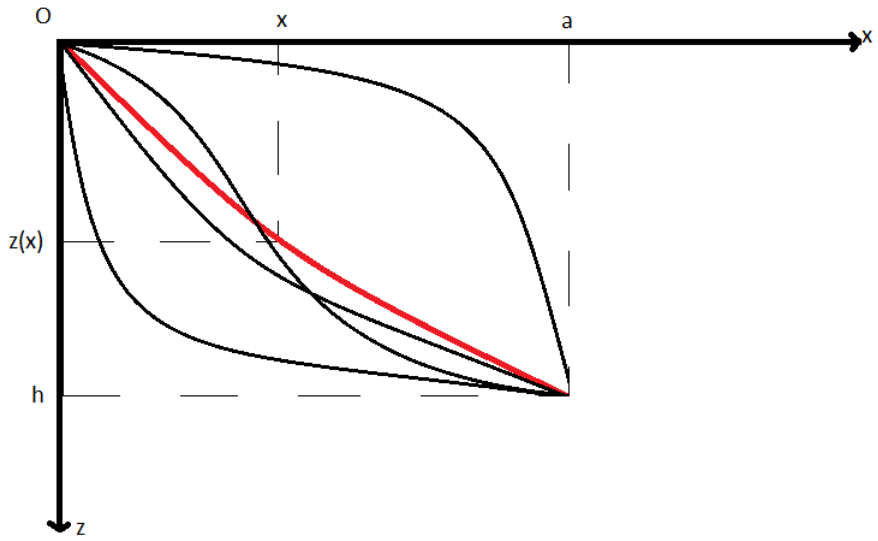

Figura 1: Rappresentazione della brachistocrona [20]

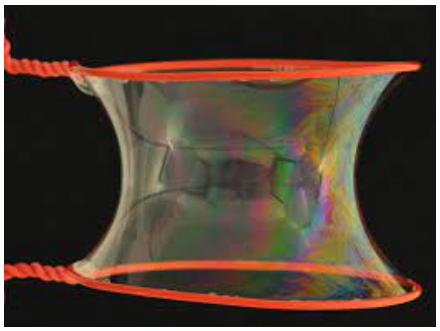

Figura 2: Esempio lamina saponata

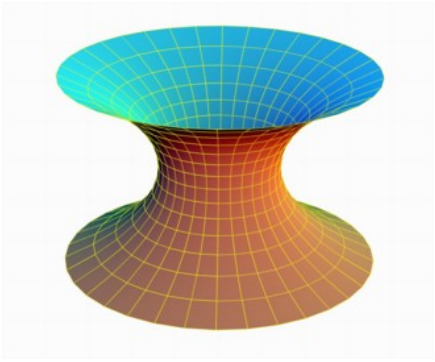

Figura 3: Catenoide

2 Swarm Intelligence

Come introdotto precedentemnte, una delle principali caratteristiche della Swarm Intelligence consiste nel concetto di *auto-organizzazione*, ossia una forma di sviluppo del sistema, attraverso influenze provenienti dagli elementi che costituiscono il sistema stesso, i quali permettono di raggiungere un maggior livello di complessità.

L'auto-organizzazione può essere considerata come un insieme di meccanismi dinamici, dove le strutture appaiono a livello globale, senza essere esplicitamente codificate a livello individuale.

Un *comportamento emergente* si ha quando un numero di individui opera in un ambiente, originando comportamenti più complessi in quanto collettivi. Tale proprietà non è predicibile, e rappresenta un'evoluzione per il sistema stesso. [2]

Le principali proprietà dell'auto-organizzazione sono:

- *Feedback positivi:* derivano dall'esecuzione di semplici regole di comportamento, che promuovono la creazione di strutture. Ad esempio la selezione del percorso verso una sorgente di cibo, che crea le condizioni per la nascita di una rete di percorsi a livello globale. In modo analogo esistono anche *Feedback negativi* che invece inibiscono la creazione di strutture.

- *Dinamicità:* la creazione, e la persistenza di strutture, richiede interazioni continue tra i membri del gruppo e l'ambiente.

- *Proprietà emergenti:* i sistemi, come già descritto, mostrano proprietà molto più complesse rispetto ai contributi di ciascun agente. Tali proprietà derivano dalla combinazione non lineare delle interazioni tra i membri della colonia.

- *Biforcazioni:* insieme alle proprietà emergenti, le interazioni non lineari portano i sistemi auto-organizzati a biforcazioni. Una biforcazione è la comparsa di nuove soluzioni stabili quando alcuni parametri del sistema cambiano. Ciò corrisponde a un cambiamento qualitativo nel comportamento collettivo.

- *Multistabilità:* i sistemi auto-organizzati possono essere multistabili. Ciò significa che, per un dato insieme di parametri, il sistema può raggiungere diversi stati stabili a seconda delle condizioni iniziali e delle fluttuazioni casuali.

Dalle precedenti descrizioni dei processi auto-organizzati, emerge un'ampia varietà di comportamenti collettivi atti a risolvere un problema.

Questa diversità può dare l'impressione che non esista un punto comune a livello collettivo, tuttavia è possibile suddividere questi comportamenti in un numero limitato di componenti comportamentali.

Ad esempio *C. Anderson* e *N. Franks* nel 2001 [9] hanno proposto di separare i comportamenti collettivi in quattro tipi di attività: *individuale*, di *gruppo*, di *squadra*, e a *compiti suddivisi*.

In seguito, *C. Anderson* ha proposto che ogni attività globale potesse essere suddivisa in una struttura gerarchica di sottoattività dei tipi precedenti.

Tale metodo può essere visto come la scomposizione di un problema nelle attività base necessarie per risolverlo.

Un modo per suddividere i comportamenti collettivi , si basa sulla scomposizione delle attività organizzative degli individui.

Sono state identificate quattro funzioni di questo tipo:

- *Coordinamento:* è l'organizzazione adeguata nello spazio e nel tempo delle attività necessarie per risolvere un problema specifico. Questa funzione porta a specifiche distribuzioni spazio-temporali degli individui, delle loro attività e/o dei risultati delle loro attività, al fine di raggiungere un determinato obiettivo. Un esempio di coordinamento si ha nell'organizzazione dello spostamento di sciami di api. In questo caso le interazioni tra gli individui generano movimenti sincronizzati (organizzazione temporale) e orientati (organizzazione spaziale) degli individui, verso un obiettivo specifico. Il coordinamento è anche coinvolto nello sfruttamento delle fonti di cibo dalle formiche che posano scie di feromoni. Esse costruiscono reti di sentieri, che organizzano spazialmente il loro comportamento di foraggiamento tra il nido e una o più fonti di cibo. Il coordinamento lavora anche nella maggior parte delle attività di costruzione nelle colonie di insetti. Durante la costruzione del nido in alcune specie di vespe o termiti, il processo stigmergico favorisce l'estensione di strutture (organizzazione spaziale) in precedenza (organizzazione temporale) ottenute da altri individui.

- *Cooperazione:* si ha quando gli individui realizzano assieme un compito, che non sarebbe potuto essere svolto da un singolo individuo. Questo comportamento è evidente nel recupero di grosse prede, quando un singolo individuo è troppo debole per spostare una certa quantità di cibo. Nella specie *pheidologeton diversus*, ad esempio, è stato riportato che le formiche partecipanti al trasporto cooperativo di una preda, possono sopportare un peso almeno dieci volte superiore rispetto ai trasportatori individuali. La cooperazione può anche essere coinvolta in compiti diversi da quelli di recupero delle prede, come ad esempio il lavoro in catena nella formica tessitrice *oecophylla longinoda*, la quale, durante la costruzione del nido, forma catene di individui "appendendosi" uno all'altro, per superare spazi vuoti tra due rami, o per legare foglie durante la costruzione del nido.

- *Deliberazione:* La deliberazione si riferisce ai meccanismi che si verificano quando uno sciame si trova di fronte a diverse possibilità. Questi meccanismi comportano una scelta collettiva per almeno una delle opportunità, come ad esempio la scelta del tragitto più breve per arrivare alla fonte di cibo.

- *Collaborazione:* significa che diverse attività sono svolte contemporaneamente da gruppi di individui specializzati. Questa specializzazione può contare su una pura differenza comportamentale, nonchè morfologica, ed

essere influenzata dall'età degli individui. Una delle espressioni più evidenti di tale divisione del lavoro è l'esistenza di caste: per esempio nel taglio di foglie, le formiche possono appartenere a quattro diverse caste, e la loro dimensione è strettamente legata ai compiti che stanno svolgendo.

2.1 Algoritmi di swarm intelligence

Tra i principali algoritmi inspirati alla Swarm Intelligence ci sono:

- **Particle Swarm Optimization (PSO):** l'algoritmo PSO, ideato da *R. Eberhart*, *J. Kennedy* e *Y. Shi* [10] deriva dallo studio del comportamento degli stormi di uccelli nella ricerca del cibo: si suppone che ogni individuo non conosca l'effettiva posizione da raggiungere, ma solo la sua distanza da esso, e che modifichi istante per istante la propria posizione e la sua velocità per minimizzare tale distanza. L'ottimizzazione avviene in funzione sia della propria esperienza passata (migliore posizione raggiunta) sia di quelle degli altri individui del gruppo. Ciascuna soluzione viene valutata attraverso una funzione di costo da minimizzare, e dall'opportuno bilancio di ciascun parametro si può garantire una completa esplorazione dello spazio di ricerca, e il superamento di ottimi locali.

  La velocità e la posizione di ciascuna particella, sono aggiornate secondo le relazioni:

$$v_{t+1} = w_t v_t + c_1 r_1 (p_t^b - x_t) + + c_2 r_2 (p^g - x_t) \tag{2.1}$$

$$x_{t+1} = x_t + v_{t+1} \tag{2.2}$$

  dove t tiene conto dell'iterazione; r_1 e r_2 sono numeri casuali compresi tra 0 e 1; p_t^b, v_t e x_t sono rispettivamente la miglior posizione, la velocità e la posizone all'istante t; w_t è detto *fattore d'inerzia*; c_1 e c_2 sono fattori di correzione, e infine, p^g è la migliore posizone di sempre

- **Ant Colony Optimization (ACO):** Questo tipo di algoritmo è stato inizialmente proposto da *M. Dorigo* nel 1990 [4], per la ricerca dei percorsi ottimali in un grafo, basandosi sul comportamento delle formiche che cercano un percorso tra la prorpia colonia e una fonte di cibo. Infatti, quest'ultime, anche se limitatamente alle capacità cognitive di una singola formica, sono in grado, collettivamente, di trovare il percorso più breve tra una fonte di cibo ed il formicaio. Generalmente usato in problemi di ottimizzazione, come il *problema del commesso viaggiatore* di cui si è parlato nel paragrafo 1.1 , l'algoritmo è basato su un insieme di individui (formiche per l'appunto), ciascuno dei quali attraversa un percorso tra quelli possibili. In ogni fase, ciascuna formica sceglie di spostarsi da una posizione all'altra secondo alcune regole:

1. Più una città è distante, meno possibilità ha di essere scelta (questa informazione è chiamata visibilità);
2. Una volta completato il proprio percorso, la formica deposita, su tutti i bordi attraversati, una quantità di feromone inversamente proporzionale alla lunghezza del percorso completato;
3. Più l'intensità del percorso di feromone è maggiore, più ha possibilità di essere scelto;
4. Le tracce di feromone depositate evaporano ad ogni iterazione;

La regola di spostamento, chiamata *"regola casuale di transizione proporzionale"* è matematicamente scritta come segue:

$$p_{ij}^k(t) = \frac{\tau_{ij}(t)^\alpha \eta_{ij}^\alpha}{\sum_{l \in J_i^k} \tau_{il}(t)^\alpha \eta_{il}^\beta} \qquad j \in J_i^k \tag{2.3}$$

dove $p_{ij}^k(t)$ è la probabilità che una formica k, all'itante di tempo t, si sposti dalla posizone i a quella j.

J_i^k è l'insieme dei possinili spostamenti di una formica k in posizone i.

η_{ij} è la *visibilità*, pari all'inverso della distanza d_{ij} tra due citta i e j.

τ_{ij} è l'intensità di feromone rialsciata lungo il percorso.

I due parametri principali che regolano l'algoritmo sono α e β che controllano l'importanza relativa dell'intensità e la visibilità di un bordo.

Una volta completato un percorso dalla posizone i alla posizone j, una formica k, deposita una quantità di feromone pari a

$$\Delta\tau_{ij}^k = \frac{Q}{L^k(t)} \qquad (i,j) \in T^k(t) \tag{2.4}$$

dove $T^k(t)$ è il percorso compiuto da una formica k all'iterazione t.

$L^k(t)$ è la lunghezza del percorso seguito, mentre Q è un parametro di regolazione.

Al termine di ogni iterazione la quantità di feromone che evapora è pari a $\rho\tau_{ij}$, con ρ parametro di regolazione

All'iterazione successiva, la quantità di feromone totale presente su ciascun percorso, tra quello effettivamente depositato e quello evaporato, è

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij}^k \tag{2.5}$$

dove m è il numero di formiche che è transitata lungo il percorso.

- **Artificial Bee Colony Optimization (ABC):** questo tipo di algoritmo è stato proposto da *D. Karaboga* nel 2005 [19], L'algoritmo utilizza una popolazione di agenti, ossia le api, per esplorare tutto lo spazio delle possibili soluzioni. Una parte della popolazione, *le api esploratrici*, cerca a caso in giro per le regioni ad alta idoneità di risultati (*ricerca globale*). Gli agenti di maggior successo (le esploratrici che trovano zone con abbondante disponibilità di polline, nettare o secrezioni zuccherine) memorizzano la propria posizione e reclutano un numero variabile di agenti inattivi, le *api spettatrici*) per cercare in prossimità delle soluzioni più idonee (*ricerca locale*). I cicli di ricerca globale e locale vengono ripetuti finché non viene scoperta una soluzione accettabile (una zona in piena fioritura), o sono trascorsi un certo numero di iterazioni (in natura il limite delle interazioni, ad esempio, potrebbe essere il tramonto del sole).

  Indichiamo con $\boldsymbol{X_i} = (x_{i,1},\ x_{i,2}, ..., x_{i,n})$ l' i-esima posizione di un ape esploratrice dello sciame, dove n indica la dimensione del problema. Ciascuna ape esploratrice $\boldsymbol{X_i}$, genera, in un intorno della propria posizione corrente, una nuova soluzione $\boldsymbol{V_i}$, secondo la relazione

$$v_{i,k} = x_{i,k} + \phi_{i,k}(x_{i,k} - x_{j,k}) \tag{2.6}$$

  Dove $\boldsymbol{X_j}$ è una delle soluzioni candidate, scelta in modo casuale, mentre $\phi_{i,k}$ è un numero caasuale scelto nell'intervallo $[-1, 1]$.

  Una volta dunque generata una nuova soluzione, $\boldsymbol{V_i}$, se il corrispondente valore di fitness è migliore rispetto a quello relativo alla soluzione $\boldsymbol{X_i}$, allora quest'ultima viene aggiornata a $\boldsymbol{V_i}$, altrimenti resta invariata.

  Le api esploratrici, una volta completato il loro processo di ricerca, condividono, attraverso una particolare danza, le informazioni sulle fonti di cibo con le altre api spettatrici. Quest'ultime scelgono, con una probabilità proporzionale al valore di fitness, in quali posizoni avviare una nuova ricerca. La probabilità di scelta è data da:

$$P_i = \frac{fit_i}{\sum_j fit_j} \tag{2.7}$$

  dove fit_i è il valore di fitness dell'i-esima soluzione.

2.2 Algoritmo FireFly

Le lucciole sono dei coleotteri appartenenti alla famiglia delle *Lampyridae*.

Nonostante differenze somatiche, entrambi i sessi sono dotati di organi luminescenti, la cui bioluminescenza è causata dall'azione di due composti chimici: la *luciferina*, un composto organico che emette luce, e la *luciferasi*, un enzima catalizzatore. In presenza di *ATP*, di magnesio e dell'enzima luciferasi, gli elettroni della luciferina si diseccitano, rilasciando l'energia in eccesso sotto forma di luce dal tipico colore verde-giallastro.

La bioluminescenza gioca un ruolo fondamentale sia nell'accoppiamento, sia nell'attirare potenziali prede. In particolar modo l'intensità luminosa oltre a scalare col quadrato della distanza, viene parzialmente assorbita dall'aria. Questi due fattori rendono le lucciole visibili solo ad una distanza limitata, sufficiente per comunicare o cacciare. La funzione di ottimizzazione dell'algoritmo è resa possibile proprio associando la funzione obbiettivo alla luminosità, e quindi all'attrattività di un esemplare.

Definiamo di seguito alcune regole, seppure idealizzate, per modellizzare ed implementare l'algoritmo Firefly [22] [7]:

- Tutte le lucciole sono asessuate, in modo che ciascun esemplare possa essere attratto o a sua volta attrarre, indipendentemente dal sesso delle altre lucciole;
- L'attrattività è proporzionale alla luminosità, quindi una lucciola meno luminosa si sposta verso una lucciola più luminosa, da essa attratta. Se due esemplari hanno lo stesso livello di luminosità, si muovono in modo casuale;
- La luminosità di ciascuna lucciola è influenzata o determinata dallo spazio della funzione obbiettivo. Inoltre la luminosità è considerata proporzionale al valore della funzione obiettivo;

Riassumendo nel FA l'attrattività è legata alla funzione obiettivo, e così come la luminosità, è inversamente proporzionale al quadrato della distanza tra due lucciole. La presenza di altri fattori, come una visibilità variabile, una funzione di attrazione versatile, e un'alta mobilità, rendono più efficiente la ricerca nello spazio della funzione da ottimizzare.

Nella figura 4 è mostrato uno tipico workflow dell'algoritmo.

Come accennato, nel FA, ci sono due questioni da prendere in considerazione: la **variazione della luminosità** e la **formulazione dell'attrattività**.

Per semplicità si può assumere che l'attrattività di una lucciola sia determinata dalla sua luminosità, la quale è a sua volta associata alla funzione di fitness da ottimizzare.

Nella maggior parte dei casi si può assumere che la luminosità I di una lucciola in una posizione $\mathbf{x}$ può essere scelta proporzionala al valore della funzione di fitness in quel punto: $I(\boldsymbol{x}) \propto f(\boldsymbol{x})$

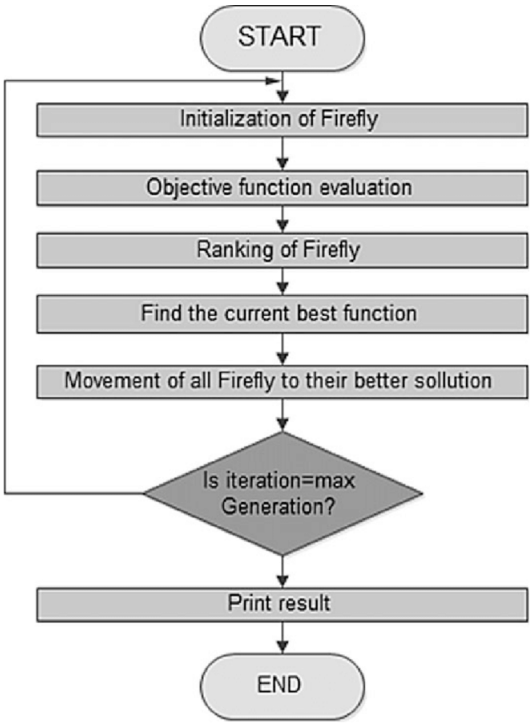

Figura 4: Workflow algoritmo Firefly [6]

L'attrattività è relativa a due singoli individui, pertanto varierà in funzione di r_{ij}, ossia alla distanza tra lucciola i e la lucciola j .

L'intensità luminosa decresce all'aumentare della distanza, oltre ad essere assorbita dal mezzo, per questo motivo è necessario introdurre un *coefficiente di assorbimento* che tenga conto di ciò.

Considerando che $I(r)$ è inversamente proporzionale al quadrato della distanza, possiamo assumere che $I(r) = I_s/r^2$, dove I_s è l'intensità in corrispondenza della posizione occupata dalla lucciola.

Per un dato mezzo con un coefficiente di assorbimento γ, possiamo esprimere l'intensità luminosa in funzione della distanza come:

$$I(r) = I_0 e^{-\gamma r^2} \tag{2.8}$$

in modo anche da evitare la divergenza in $r = 0$ nell'espressione precedente.

Alcune volte si preferisce usare una funzione che descresce monotona in modo più lento. In questo caso si può usare la seguente approssimazione:

$$I(r) = \frac{I_0}{1 + \gamma r^2} \tag{2.9}$$

In ogni caso, per brevi distanze, le due espressioni sono equivalenti fino a $O(r^3)$, come mostra l'espansione in serie di Taylor:

$$e^{-\gamma r^2} \approx 1 - \gamma r^2 + \frac{1}{2}\gamma^2 r^4 + ... \qquad \frac{1}{1+\gamma r^2} \approx 1 - \gamma r^2 + \frac{1}{2}\gamma^2 r^4 + ... \tag{2.10}$$

Come già detto, l'attrattività di una lucciola è proporzionale all'intensità luminosità percepita dalle lucciole adiacenti. Possiamo definire l'attrattività di un individuo come:

$$\beta(r) = \beta_0 e^{-\gamma r^2} \tag{2.11}$$

dove β_0 è l'attrattività a $r = 0$. Analogamente a quanto descritto in precedenza, è possibile adottare una anche rappresentazione del tipo:

$$\beta(r) = \frac{\beta_0}{1+\gamma r^2} \tag{2.12}$$

L'equazione 2.12 definisce una distanza caratteristica $\Gamma = \frac{1}{\sqrt{\gamma}}$, tale per cui β_0 si riduce di un fattore pari a $\frac{1}{e}$

In alcune implementazioni, la forma effettiva della funzione di attrattività $\beta(r)$ può essere una qualsiasi funzione monotona decrescente, da cui la seguente forma generalizzata:

$$\beta(r) = \beta_0 e^{-\gamma r^m}, \qquad (m \leq 1). \tag{2.13}$$

A γ fissato, la lunghezzza caratteristica $\Gamma = \gamma^{-1/m} \to 1$ per $m \to \infty$.

Solitamente, si fissa $\gamma = \frac{1}{\Gamma^m}$

La distanza cartesiana (anche se, nello spazio n-dimensionale, possono usarsi altre forme di distanze convenienti alla tipologia del problema in esame) tra due lucciole i e j, nelle rispettive posizioni $\boldsymbol{x_i}$ e $\boldsymbol{x_j}$ è pari a:

$$r_{ij} = \|\boldsymbol{x_i} - \boldsymbol{x_j}\| = \sqrt{\sum_{k=1}^{d}(x_{i,k} - x_{j,k})^2} \tag{2.14}$$

dove $x_{i,k}$ è la k-esima componente della coordinata spaziale $\boldsymbol{x_i}$ dell'i-esima lucciola.

In 2 dimensioni abbiamo semplicemente $r_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$.

Il movimento di una lucciola i, attratta da una lucciola j più luminosa, e quindi più attraente, è determinato da:

$$\boldsymbol{x_i} = \boldsymbol{x_i} + \beta_0 e^{-\gamma r_{ij}^2}(\boldsymbol{x_j} - \boldsymbol{x_i}) + \alpha(rand - \frac{1}{2}) \tag{2.15}$$

Nell'ultimo termine, α è il *parametro di randomizzazione* , mentre *rand* è un numero generato casualmente e uniformemente distribuito nell'intervallo $[0, 1]$, oppure esteso ad una distribuzione normale *N(0,1)*. Nella maggior parte dei casi si fissa $\beta_0 = 1$, mentre $\alpha \in [0, 1]$.

Il paramtero γ, come già accennato, caratterizza l'attrattività di un esemplare, per cui la sua variazione è di cruciale importanza nel determinare sia il comportamento dell'algoritmo, sia la velocità con la quale esso arriva a aconvergenza. In teoria, $\gamma \in [0, \infty]$, ma solitamente nella pratica si sceglie un valore tale per cui $\gamma = O(1)$, tipicamente nell'intervallo $[0.01, 100]$.

E' interessante notare come nel caso limite in cui $\gamma \to 0$ l'attrattività risulti costante al valore $\beta = \beta_0$, mentre $\Gamma \to \infty$: ciò equivale ad affermare che l'intensità luminosa, e quindi l'attrattività, non diminuiscano all'aumentare della distanza. In questo caso un ottimo globale risulterebbe facilmente individuabile.

Nel caso in cui $\gamma \to \infty$, $\Gamma \to 0$ e $\beta(r) \to \delta(r)$ (funzione *delta di Dirac*): in questo caso la visibilità di ciascuna lucciola è fortemente ridotta, inibendo così la mutua interazione tra le lucciole. Ciò corrisponde dunque ad una ricerca casuale.

Un altro fattore che rende più efficiente l'algoritmo Firefly, rispetto agli algoritmi genetici, o all'algoritmo PSO, è la *capacità di aggregazione* degli individui: la popolazione infatti, nel corso delle varie iterazioni, si suddivide, in modo autonomo, in sottogruppi, i quali sono in grado di individuare gli eventuali ottimi in modo simultaneo.

3 Strumenti utilizzati

3.1 Python e DEAP

L'implementazione dell'Algoritmo Firefly, è stata realizzata in Python: un linguaggio di programmazione ad alto livello, orientato agli oggetti e caratterizzato da dinamicità, semplicità e flessibilità.

Python possiede anche dei framework, ossia dei file che raggruppano costanti, funzioni e classi, con lo scopo di rendere più agevole la programmazione.

Uno di questi, usati nel lavoro di tesi, è **DEAP** (*Distributed Evolutionary Algorithms in Python*), sviluppato al *Computer Vision and System Laboratory (CVSL)* dell'Università Laval di Quebec City [8]

Il framework del modulo, la cui documentazione è consultabile alla pagina `https://deap.readthedocs.io/en/master/index.html` è caratterizzato dai seguenti aspetti:

- la chiarezza delle strutture dati, le quali rendono semplice l'implementazione degli algoritmi;
- la natura esplicita degli operatori di selezione ed esplorazione, nonchè la facilità con la quale essi possono essere parametrizzati. In tal senso infatti, maggiori sono le possibilità di parametrizzazione da parte dell'utente, maggiori sono le opportunità di sfrutttare le potenzialità dell'algoritmo;
- la presenza di meccanismi in grado di sviluppare facilmente paradigmi di distribuzione;

3.2 Struttura e tools di DEAP

La struttura di *DEAP* è caratterizzata da alcuni oggetti in grado di definire parti specifiche dell'algoritmo evolutivo, tutti messi a disposizione dal modulo **base**.

Il modulo **creator** ad esempio, permette di creare nuove classi partendo da quelle standard, alle quali è possibile aggiungere nuovi attributi (*composition*). Ciò è possibile sfruttando il metodo **create()**, la cui sintassi è mostrata di seguito:

```
deap.creator.create(name, base[attribute, ...])
```

Questo metodo ha come primo argomento il nome della classe che si vuole creare, e come secondo una classe i cui attributi si vogliono ereditare. Agli attributi "ereditati" se ne possono aggiungere di nuovi semplicemente specificandoli come altri argomenti.

Come già accennato, la fitness è una misura della qualità di una soluzione. Utilizzando proprio il metodo *create()*, possiamo creare una nuova classe che eredita le proprietà della classe *Fitness* messa a disposizione dal modulo *base*:

```
creator.create("Fitness", base.Fitness, weights)
```

a cui aggiungiamo l'attributo *weights*: una tupla di pesi che definisce la strategia dell'algoritmo rispetto ai parametri da ottimizzare.

Col modulo *creator* è possibile istanziare anche gli individui della popolazione:

```
creator.create("Individual",list, fitness=creator.Fitness)
```

La classe *Individual* estende la classe *list* di Python. Ciò significa che gli individui saranno del tipo lista. Ciascun individuo inoltre, è caratterizzato da un attributo chiamato *fitness*, della classe *Fitness* che abbiamo precedentemente descritto. Per poter estrarre il valore di fitness un individuo basta semplicemente richiamare l'attributo *fitness*:

```
individual.fitness.values
```

Un altro modulo messo a disposizione da *DEAP* è il **toolbox**, un vero e proprio contenitore di operatori.

I principali metodi messi a disposizione dal Toolbox sono:

- **clone()**: duplica ogni elemento che viene passato come argomento. Questo metodo eredita le sue proprietà dalla funzione *copy.deepcopy*;
- **map()**: applica la funzione che viene passata come primo argomento a tutti gli elementi di un oggetto iterabile che viene passato come secondo argomento. Queste proprietà sono ereditate dalla funzione *map*;

- **register()**: permette di "registrare" nel Toolbox qualsiasi funzione si voglia. L'implementazione di questo metodo è la seguente:

  ```
  deap.base.Toolbox.register(alias,function,argument)
  ```

  Il primo argomento è il nome dell'operatore che si vuole registrare nel toolbox. Il secondo invece è l'effettiva funzione su cui si basa l'alias, mentre gli argomenti successivi sono semplici attributi riferiti alla funzione chiamata.

- **unregister()**: è l'opposto del metodo register. Come argomento accetta l'alias della funzione che si vuole eliminare dal toolbox.

Il modulo **tools** contiene tutti gli operatori utili per mutare, ricombinare e selezionare gli individui.

Per poter registrare al toolbox l'operatore che si desidera, basta utilizzare il metodo *register()* di cui si è accennato prima. Di seguito un esempio della definizione degli operatori di crossover, mutazione e selezione.

```
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate,tools.mutGaussian, mu=0, sigma=1, indpb=0.1)
toolbox.register("select",tools.selTournament, tournsize=3)
```

Il metodo **initRepeat()**, sempre messo a disposizione dal modulo *tools*, permette anche di inizializzare sia gli individui, sia la popolazione, usando la seguente sintassi:

```
deap.tools.initRepeat(container,func,n)
```

dove *container* è l'oggetto in cui vanno inseriti i dati generati dall'attributo *func*, il quale deve essere ripetuto un numero n di volte.

Il modulo *tools* contiene anche diversi strumenti utili a raccogliere informazioni nel processo evolutivo, come ad esempio le seguenti classi:

- **Statistics**:

  ```
  deap.tools.Statistics([key])
  ```

  l'argomento *key* che viene passato all'oggetto creato è una funzione che sarà applicata ai dati su cui l'analisi statistica è stata eseguita.

- **Logbook**:

  ```
  deap.tools.Logbook()
  ```

  permette di collezionare tutte le statistiche eseguite dall'algoritmo. Usando il metodo **select()** è possibile estrarre le statistiche desiderate, come ad esempio il valore maggior (o minor) valore di fitness.

- **Hall of Fame**:

  ```
  deap.tools.HallOfFame(max_size)
  ```

  può essere usata per memorizzare i migliori individui di sempre della popolazione, anche se quest'ultimi sono andati persi durante l'evoluzione, a causa ad esempio della selezione, del crossover o della mutazione. I valori di fitness memorizzati sono continuamente ordinati in modo che il primo inidividuo della lista sia quello con la miglior fitness valutata. L'attributo *maxsize* è una costante che detrmina il nummero massimo di individui da meomorizzare.

- **History**:

  ```
  deap.tools.History()
  ```

  aiuta a costruire una genealogia di tutti gli individui generati nel corso dell'evoluzione. Contiene due attributi, il *genealogy tree* che è un dizionario di elenchi indicizzati per individuo, dove ciascun elenco contiene gli indici dei genitori. Il secondo attributo *genealogy history* contiene ogni individuo indicizzato dal proprio numero individuale. Ciò permette di realizzare un vero è proprio albero genealogico come quello mostrato in figura 5

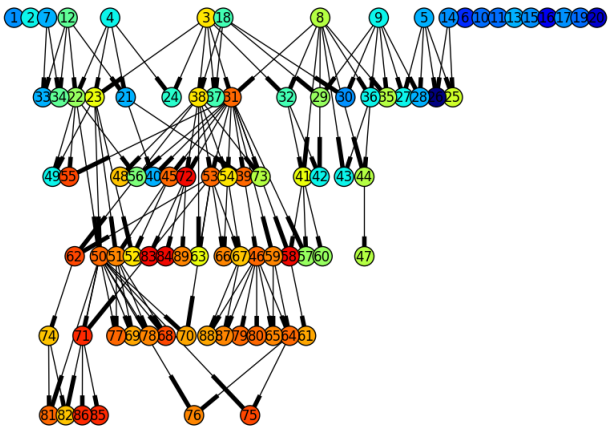

Figura 5: Albero genealogico individui della popolazione

4 Esperimenti e Risultati

4.1 Funzioni di benchmark

Le funzioni di benchmark sono funzioni aventi lo scopo di determinare la qualità di un algoritmo di ottimizzazione. In base alla "natura" di queste funzioni, può essere più o meno difficoltoso per un algoritmo trovare la soluzione ottima al problema.

Fissata la dimensione n, le funzioni di benchmark utilizzate sono riportate di seguito [18].

4.1.1 Funzione della sfera

$$f(x) = \sum_{i=1}^{n} x_i^2 \tag{4.1}$$

Come si può notare si tratta di una funzione continua, convessa ed unimodale. Il punto di minimo globale è $x^* = (0, ..., 0)$ mentre il minimo globale è $f(x^*) = 0$. In questo caso, la funzione è stata valutata, come spesso avviene, nell'ipercubo per cui $x_i \in [-5.12, 5.12]$

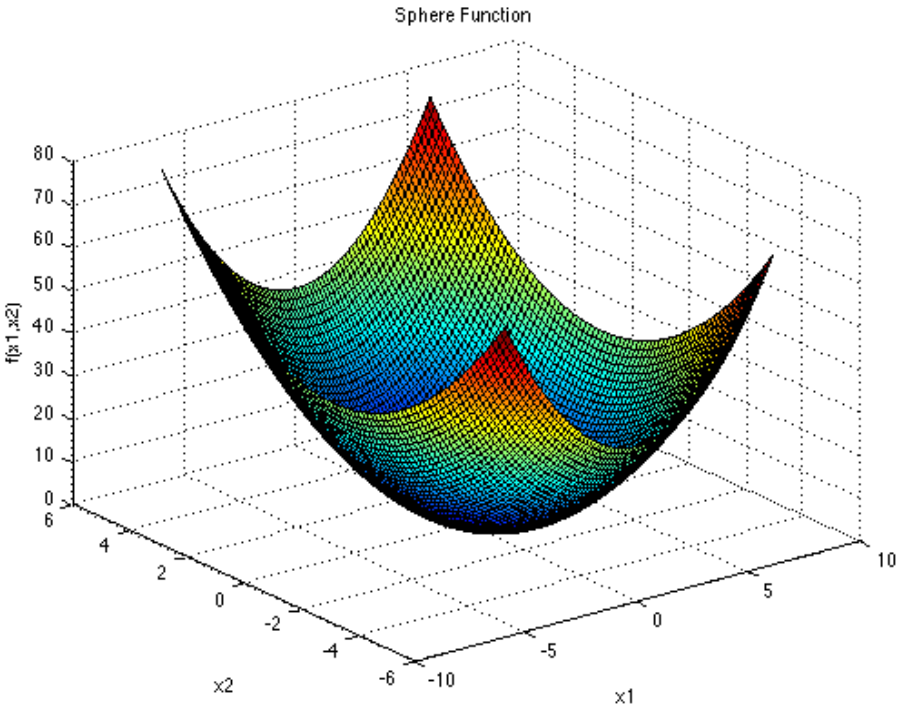

Figura 6: Funzione della sfera

4.1.2 Funzione Dropwave

$$f(x) = -\frac{1 + cos(12 + \sqrt{x_1^2 + x_2^2})}{0.5(x_1^2 + x_2^2) + 2} \tag{4.2}$$

La funzione è valutata per i valori di $x_i \in [-5.12, 5.12]$ per $i = 1, 2$. Il punto di minimo globale è $x^* = (0, 0)$, mentre il minimo globale è $f(x^*) = -1$. Come si può notare, la funzione è multimodale, in particalor modo ha numerosi minimi locali che stressano l'algoritmo.

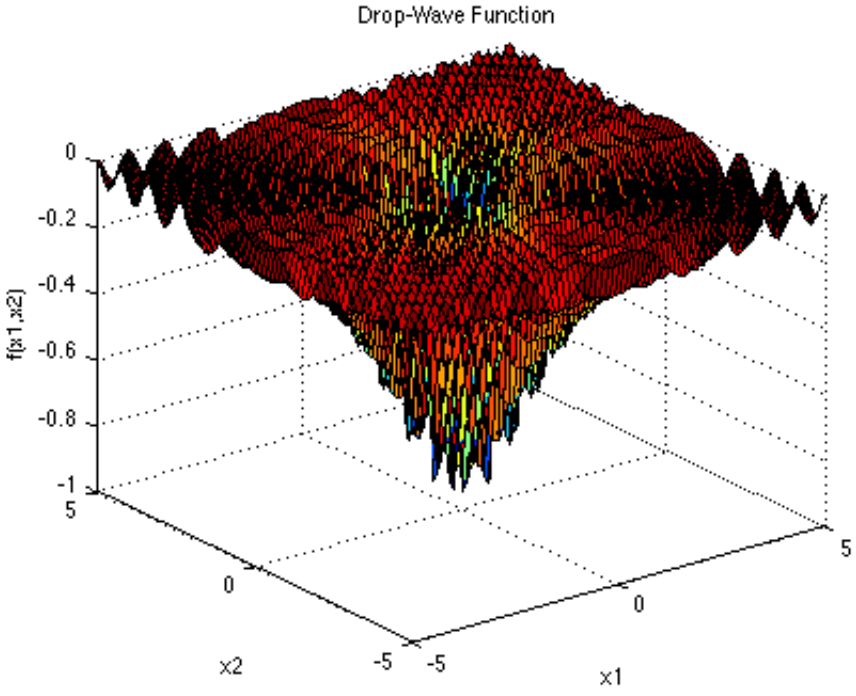

Figura 7: Funzione Dropwave

4.1.3 Funzione di Easom

$$f(x) = -cos(x_1)cos(x_2)exp(-(x_1-\pi)^2-(x_2-\pi)^2) \tag{4.3}$$

Valutata nel quadrato $x_i \in [-100, 100]$ per $i = 1, 2$. Il punto di minimo globale della funzione è $x^* = (\pi, \pi)$, tale per cui $f(x^*) = -1$. Come è possibile osservare dalla figura 8, si tratta di una funzione unimodale, il cui minimo globale è localizzato in una piccola area dello spazio di ricerca.

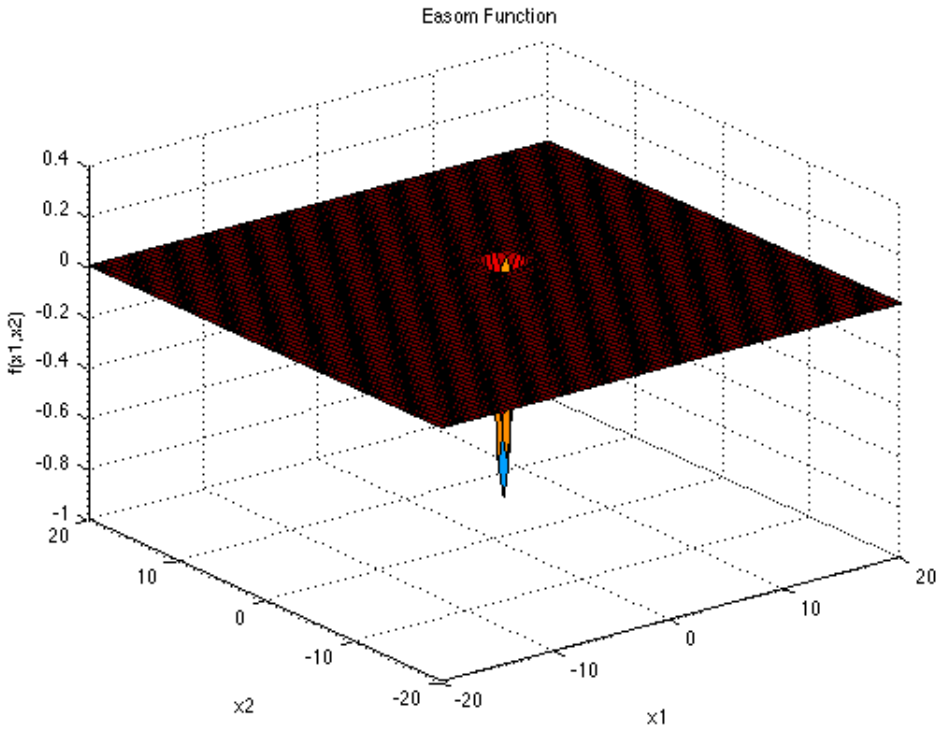

Figura 8: Funzione di Easom

4.1.4 Funzione di Shubert

$$f(x) = \left(\sum_{i=1}^{5} i\, cos((i+1)\, x_1 + i)\right)\left(\sum_{i=1}^{5} i\, cos((i+1)\, x_2 + i)\right) \tag{4.4}$$

E' valutata in $x_i \in [-10, 10]$ per $i = 1, 2$, mentre il valore di minimo globale è $f(x^*) = -186.7309$. Questa funzione presenta non solo diversi minimi globali, ma anche svariati minimi locali, i quali ostacolano il processo di ricerca dell'algoritmo.

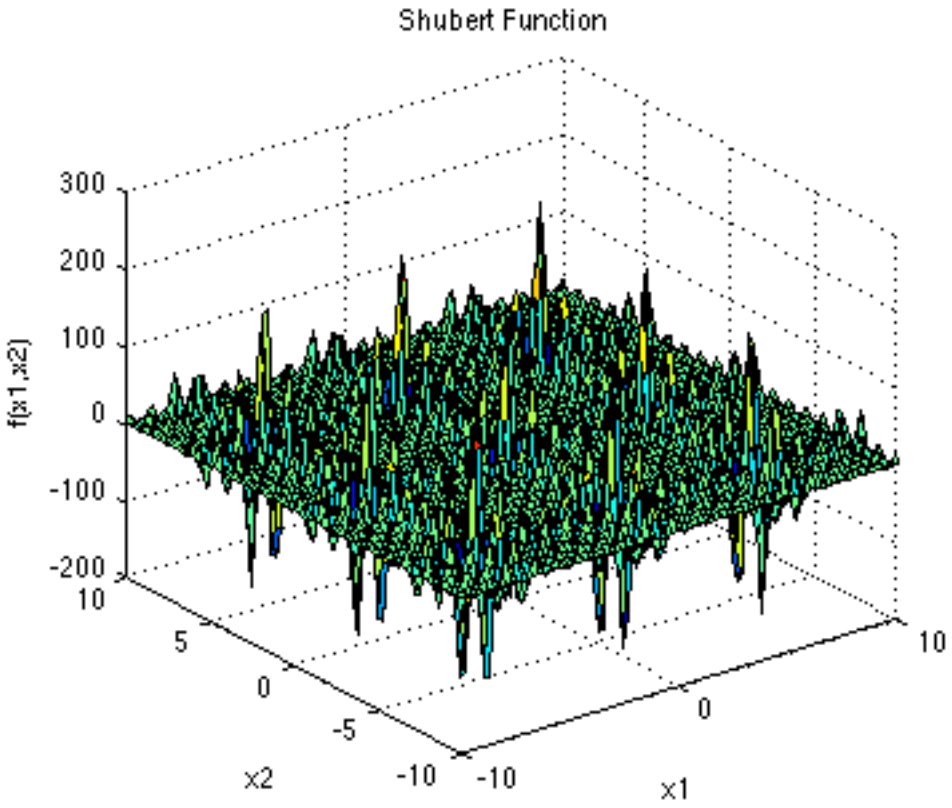

Figura 9: Funzione di Shubert

4.1.5 Funzione di Rosenbrock

$$f(x) = \sum_{i=1}^{n-1} (100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2) \tag{4.5}$$

Valutata rispetto ai valori di $x_i \in [-5, 10]$. In questo caso il punto di minimo globale è $x^* = (1, ..., 1)$ in cui la funzione vale $f(x^*) = 0$. E' una funzione unimodale con il minimo globale situato in una "valle" parabolica, il che può rendere difficile la ricerca del suddetto minimo.

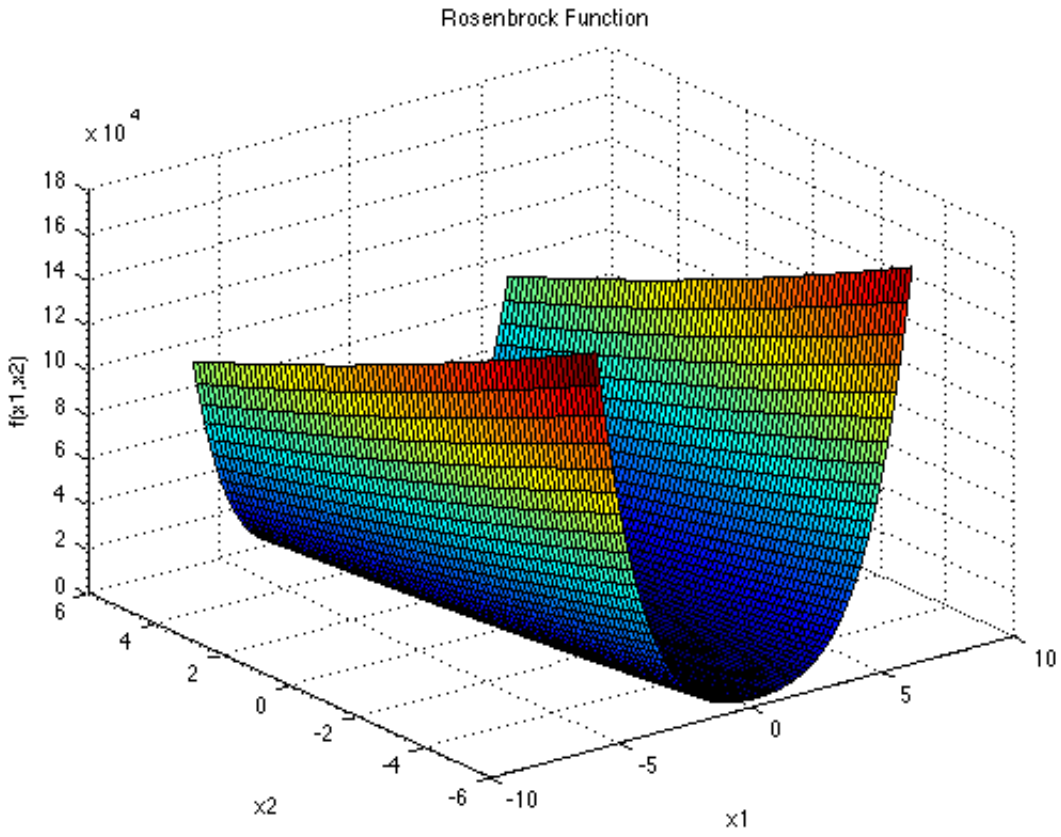

Figura 10: Funzione di Rosenbrock

4.2 Risultati esperimenti

La sperimentazione svolta in questa tesi rappresenta uno studio delle prestazioni dell'Algoritmo Firefly, le cui principali caratteristiche sono state descritte nel Capitolo 2. Queste prestazioni sono state confrontate con quelle di un Algoritmo Genetico, la cui struttura è stata descritta nell'Appendice.

In totale sono state realizzate 9 varianti sia per l'Algoritmo Genetico, sia per l'Algoritmo Firefly, e per ognuna di essa si sono seguite le istruzioni indicate per la competizione *CEC"2005 Special Session* [17]:

- per ogni funzione di test, ciascun algoritmo è eseguito per 25 volte, quindi ciascun campione di dati è composto da 25 valutazioni di fitness;
- ciascun algoritmo esegue 100000 valutazioni della funzione di fitness, quindi ogni run si ferma al raggiungimento delle 100000 valutazioni.

L'Algoritmo Genetico genera in maniera casuale una popolazione individui, il cui numero è indicato con *POPULATION SIZE*, i quali vengono subito dopo valutati attraverso l'operatore di valutazione che applica loro la funzione di fitness assegnandone così un valore di fitness. L'operatore di selezione adoperato è di tipo *"Torneo" (Tournament)*, caratterizato dall'iperparametro k uguale a 3.

L'operatore di crossover è il *Simulated Binary Crossover*. Il processo di crossover riguarda solo una parte degli individui, in base alla probabilità di crossover, denominata CXPB.

Dopo di ciò, ogni individuo viene modificato usando l'operatore *mutPolynomialBounded*, con probabilità indipendente di mutazione pari a MUTPB.

Anche nell'Algoritmo Firefly viene generata casualmente una popolazione di individui pari a *POPULATION SIZE*, valutati immediatamente con l'operatore di valutazione che funziona in maniera analoga all'Algoritmo Genetico. Le nuove popolazioni vengono generate con la formula generalizzata 2.15, dove i vari parametri possono assumere differenti valori, determinati seguendo le istruzioni esposte in [14].

Per la comparazione tra l'algoritmo Firefly e l'algoritmo Genetico, sono stati utilizzati i cosiddetti *boxplot*: ciascun "boxplot", orientato verticalmente, è rappresentato tramite un rettangolo diviso in due parti, da cui fuoriescono due segmenti. Il rettangolo è delimitato dal primo e dal terzo quartile, $q_{1/4}$ e $q_{3/4}$, e diviso al suo interno dalla mediana, $q_{1/2}$. I segmenti sono delimitati dal minimo e dal massimo dei valori. In questo modo vengono rappresentati graficamente i quattro intervalli ugualmente popolati delimitati dai quartili.

Per arricchire la comparazione tra l'Algorltimo Firefly e l'Algortimo Genetico, ed evidenziare le loro differenze, si è provveduto ad implementare anche dei *grafici scatterplot*.

In ciascun grafico si sono riportate le posizioni degli individui ottenuti come miglior soluzione al termine di ciascuno dei 25 run eseguiti per ciascun algoritmo coinvolto nella comparazione.

4.2.1 Risultati funzione della sfera

Per entrambi gli algoritmi la valutazione della funzione della sfera è stata implementata considerando individui di dimensioni $D = 2, 3, 10$

- **Dimensione = 2**

POPULATION SIZE	α	β_0	γ	θ
50	0.0800	0.0500	0.0005	0.0040

Tabella 1: Iperparametri FA

POPULATION SIZE	CXPB	MUTPB	η
300	0.4	0.1	1

Tabella 2: Iperparametri GA

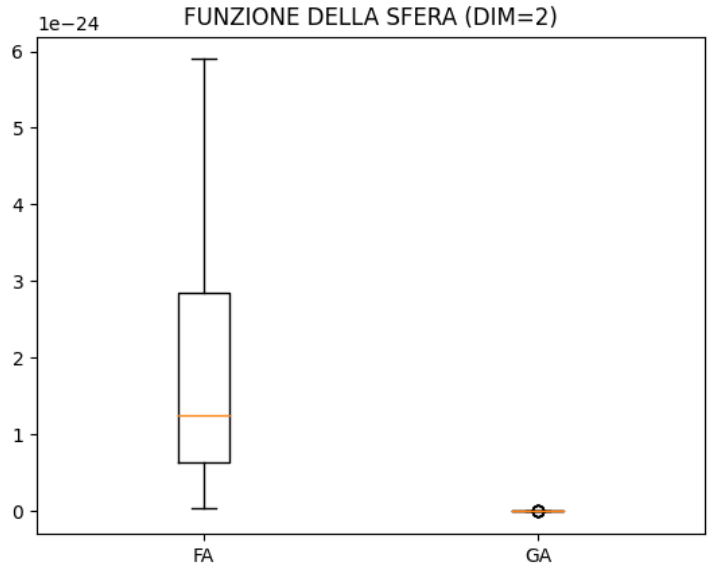

Figura 11: Box plot valori di fitness

Come si può osservare dalla figura 6, la funzione della sfera non è particolarmente complessa, ragion per cui l'Algoritmo Firefly non è la migliore soluzione per problemi di ottimizzazione relativamente semplici. Il teorema del *No Free Lunch* dimostra infatti proprio l'impossibilità di trovare un

algoritmo che si comporti meglio di tutti gli altri per un qualsiasi problema. Dal box plot è possibile osservare come nonostante anche l'Algoritmo Firefly sia in grado di predire correttamente il valore del minimo globale, l'algoritmo Genetico è molto più preciso.

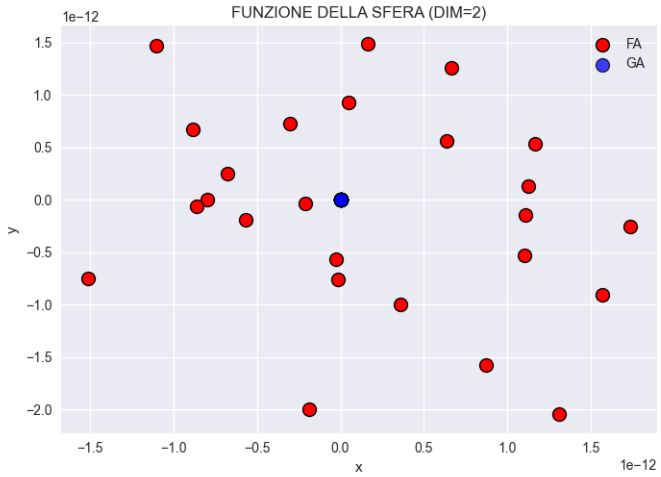

Figura 12: Scatter plot posizioni individui

Anche dallo scatter plot è possibile osservare la dispersione delle posizioni predette dall'Algoritmo Firefly, rispetto a quella corretta, individuata dall'Algoritmo Genetico.

- **Dimensione = 3**

POPULATION SIZE	α	β_0	γ	θ
40	0.0800	0.0500	0.0005	0.0040

Tabella 3: Iperparametri FA

POPULATION SIZE	CXPB	MUTPB	η
300	0.4	0.1	1

Tabella 4: Iperparametri GA

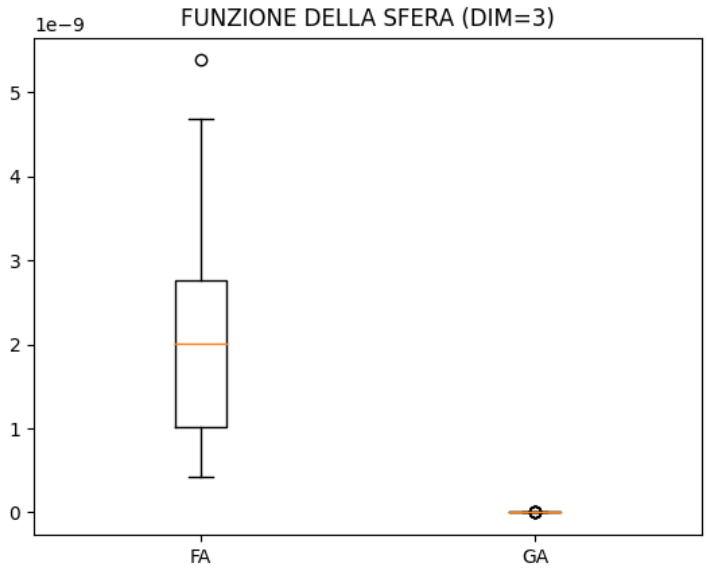

Figura 13: Box plot valori di fitness

Nonostante l'Algoritmo Genetico continui a predire correttamente il minimo globale della funzione, l'accuratezza rispetto al caso bidimensionale è sensibilmente diminuita, come è possibile notare dall'ordine di grandezza dei valori di fitness. Ciò è dovuto al fatto che aumentando la dimensione, aumenta anche la complessità del problema di ottimizzazione. Nonostante i valori predetti continuino ad essere corretti, anche l'algoritmo Firefly risente di questa variazione di dimensionalità, ma questa volta la discrepanza con i valori di fitness dell'Algoritmo Genetico è meno accentuata.

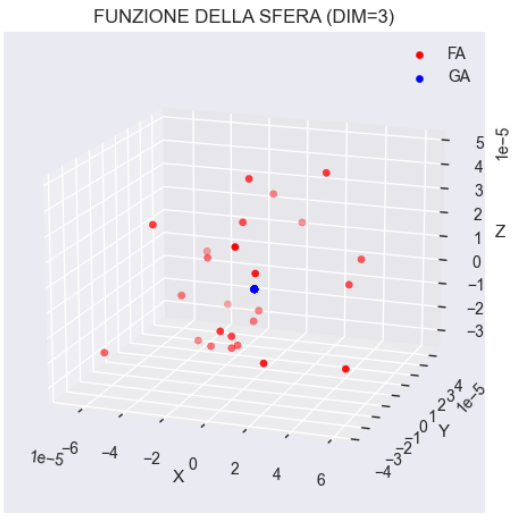

Figura 14: Scatter plot posizioni individui

Analogamente a quanto visto nel caso bidimensionale, anche in questo caso le posizioni degli indivui predetti dall'Algoritmo Firefly si disperdono in un intorno del punto di minimo ricavato dall'Algoritmo Genetico.

- **Dimensione = 10**

POPULATION SIZE	α	β_0	γ	θ
50	0.0800	0.0500	0.0005	0.0040

Tabella 5: Iperparametri FA

POPULATION SIZE	CXPB	MUTPB	η
300	0.4	0.1	1

Tabella 6: Iperparametri GA

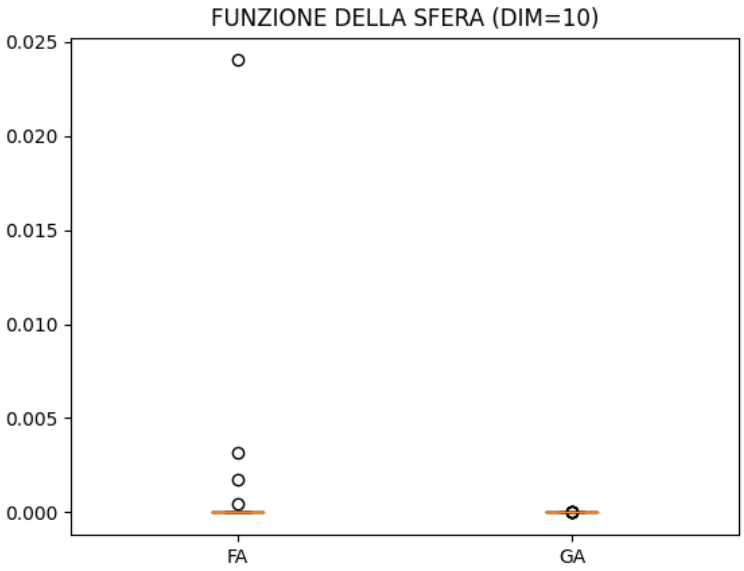

Figura 15: Box plot valori di fitness

Aumentando la dimensione si può notare come, nonostante l'Algoritmo Genetico continui ad essere ancora l'algoritmo più efficiente, l'Algoritmo Firefly inizi ad avere una distribuzione di valori più centrata intorno al valore corretto. Ciò rimarca la tendenza dell'Algoritmo Firefly ad essere una buona soluzione per problemi di ottimizzazione più complessi.

4.2.2 Risultati funzione Dropwave

POPULATION SIZE	α	β_0	γ	θ
20	0.09900	0.00100	0.00001	0.70000

Tabella 7: Iperparametri FA

POPULATION SIZE	CXPB	MUTPB	η
300	0.25	0.70	1

Tabella 8: Iperparametri GA

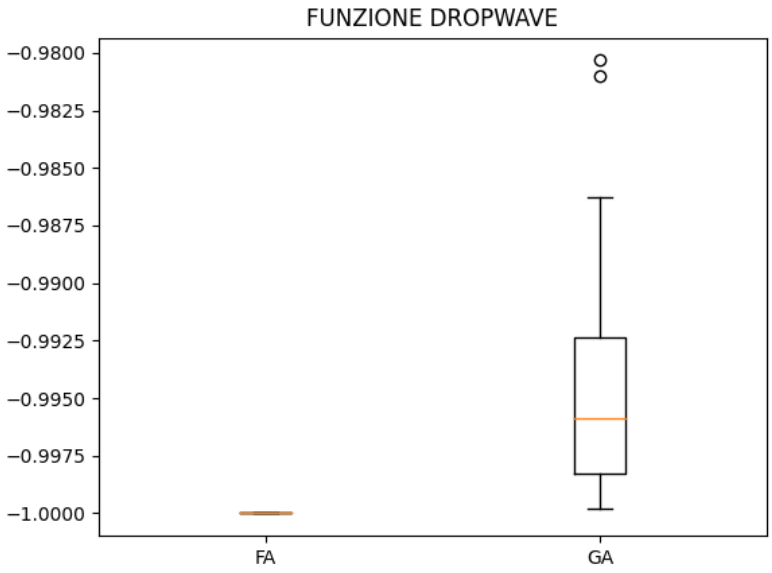

Figura 16: Box plot valori di fitness

Dal boxplot è possibile notare che in ciascuno dei 25 run, l'Algoritmo Firefly riesce sempre ad individuare perfettamente il minimo della funzione, e nessun valore predetto si discosta da esso. Ciò invece non è vero nel caso dell'Algoritmo Genetico, in cui si nota una più ampia dispersione dei valori di fitness, conseguenza del fatto che la funzione presenta molti minimi locali in cui l'Algoritmo Genetico resta facilmente "intrappolato".

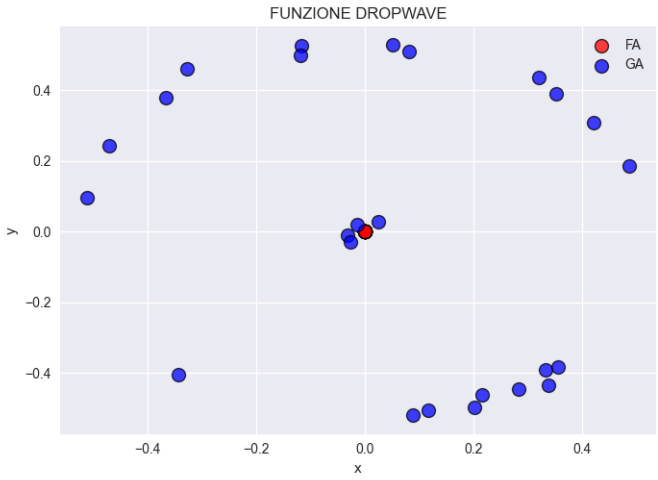

Figura 17: Scatter plot posizioni individui

Quanto detto in precedenza viene confermato anche nel caso dello scatter-plot. L'Algoritmo Firefly, in ciascuna esecuzione, converge al punto di minimo della funzione, mentre i migliori individui dell'Algoritmo Genetico tendono a disporsi in un intorno di tale punto, senza effettivamente mai raggiungerlo.

4.2.3 Risultati funzione di Easom

POPULATION SIZE	α	β_0	γ	θ
30	0.09900	0.00100	0.00001	0.70000

Tabella 9: Iperparametri FA

POPULATION SIZE	CXPB	MUTPB	η
300	0.25	0.10	1

Tabella 10: Iperparametri GA

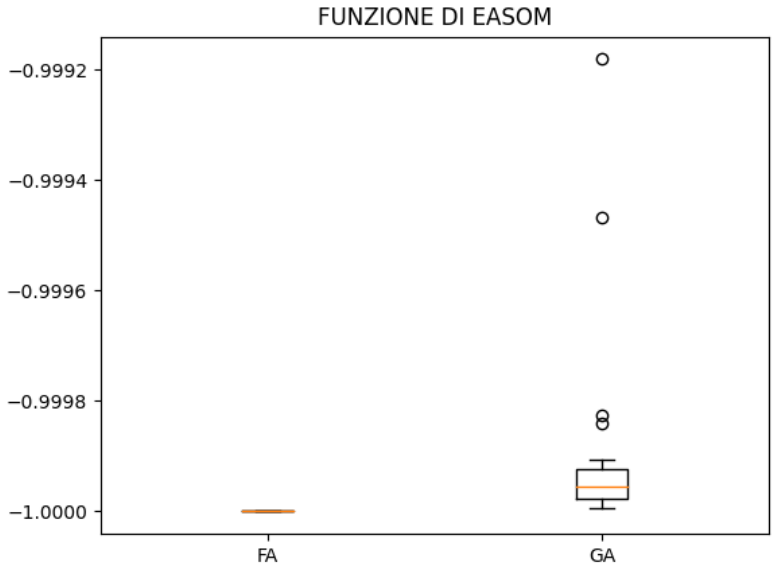

Figura 18: Box plot valori di fitness

Anche in questo caso è evidente come l'Algoritmo Firefly, che in ciascuna delle 25 esecuzioni individua sempre il valore di minimo, risulti molto più efficiente rispetto all'Algoritmo Genetico, il quale invece ha una distribuzione di valori che si discosta dal valore effettivo del minimo. Nonostante non siano presenti minimi locali, in questo caso la difficoltà della ricerca risiede nel fatto che il minimo è localizzato in una regione molto ristretta del dominio, come mostrato in figura 8. Da ciò si evince come l'Algoritmo Firefly, rispetto a quello Genetico, esplori in maniera più efficiente lo spazio di ricerca.

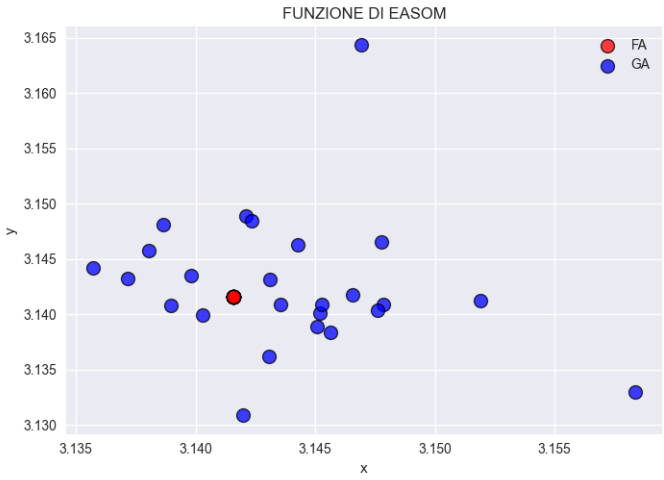

Figura 19: Scatter plot posizioni individui

Lo scatter plot rimarca quanto appena detto, poichè nessuno dei migliori individui predetti dall'Algoritmo Genetico centra il punto di minimo, al contrario dell'Algoritmo Firefly che arriva sempre a convergenza.

4.2.4 Risultati funzione di Shubert

POPULATION SIZE	α	β_0	γ	θ
50	2.000	0.001	0.700	0.095

Tabella 11: Iperparametri FA

POPULATION SIZE	CXPB	MUTPB	η
300	0.40	0.10	1

Tabella 12: Iperparametri GA

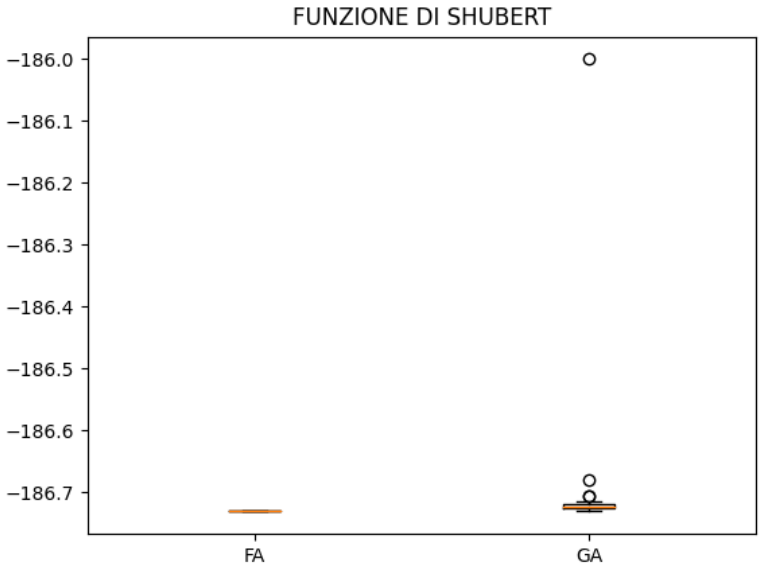

Figura 20: Box plot valori di fitness

Nonostante questa volta la distribuzione dei valori di fitness dell'Algoritmo Genetico sia abbastanza concentrata intorno al minimo della funzione, alcuni valori si discostano di molto da esso. Nel caso dell'Algoritmo Firefly invece, il minimo viene individuato in ognuno dei 25 run.

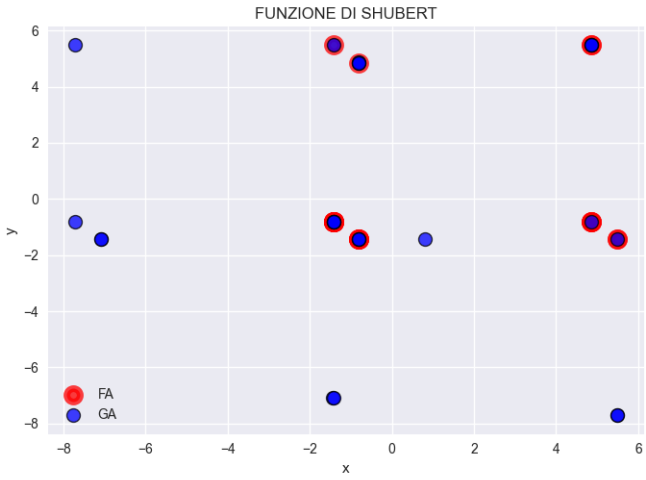

Figura 21: Scatter plot posizioni individui

Essendo la funzione di Shubert multimodale, lo scatterplot mostra come in alcuni casi l'Algoritmo Genetico confonda i minimi locali con quelli globali, a dfferenza dell'Algoritmo Firefly, che invece non riscontra questo problema.

4.2.5 Risultati Funzioni di Rosenbrock

Così come nel caso della funzione della sfera, per entrambi gli algoritmi la valutazione della funzione di Rosenbrock è stata implementata considerando individui di dimensioni $D = 2, 3, 10$

- **Dimensione = 2**

POPULATION SIZE	α	β_0	γ	θ
10	0.1000	0.0250	0.0009	0.0100

Tabella 13: Iperparametri FA

POPULATION SIZE	CXPB	MUTPB	η
300	0.950	0.001	1

Tabella 14: Iperparametri GA

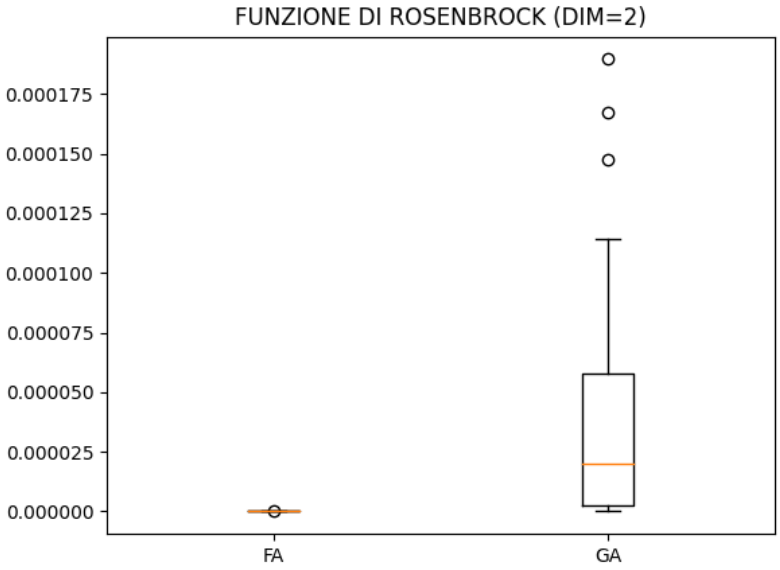

Figura 22: Box plot valori di fitness

Come mostrato in figura 10, la funzione di Rosenbrock è una funzione unimodale con il minimo globale situato in una "valle" parabolica, il che rendere difficile la ricerca del suddetto minimo. Questo problema però

è efficacemente superato dall'Algoritmo Farfly, che è sempre in grado di determinare il minimo globale.

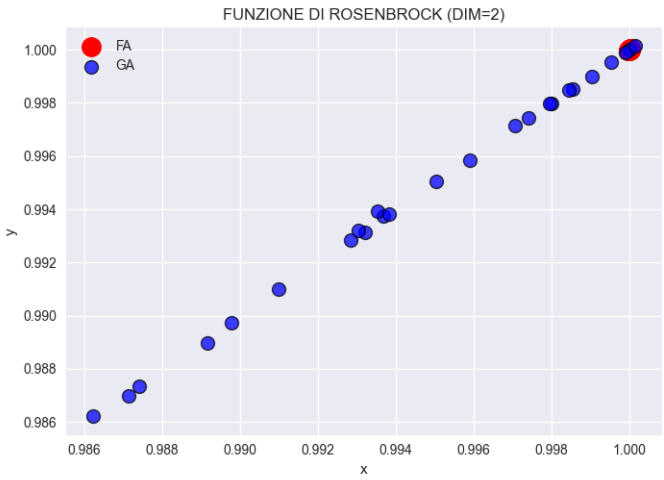

Figura 23: Scatter plot posizioni individui

Lo scatter plot mette in evidenza come, nella maggior parte delle volte, l'Algoritmo Genetico "venga ingannato" dalla valle parobalica, che impedisce di convergere al minimo globale, cosa che invece non accade per l'Algoritmo Firefly.

- **Dimensione = 3**

POPULATION SIZE	α	β_0	γ	θ
20	0.1000	1.0000	0.0090	0.0999

Tabella 15: Iperparametri FA

POPULATION SIZE	CXPB	MUTPB	η
300	0.75	0.90	1

Tabella 16: Iperparametri GA

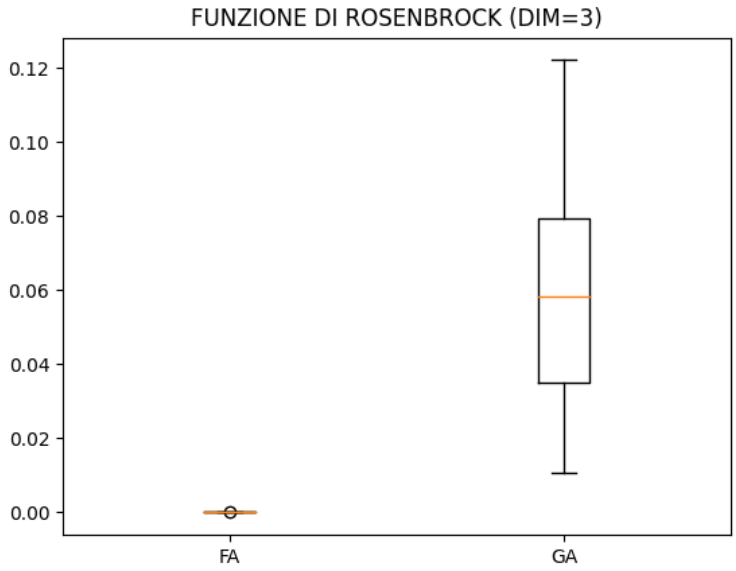

Figura 24: Box plot valori di fitness

Aumentando la dimensione si può notare come le predizioni dell'Algoritmo Genetico peggiorino, mentre l'Algoritmo Firefly continua a ad individuare correttamente il minimo globale.

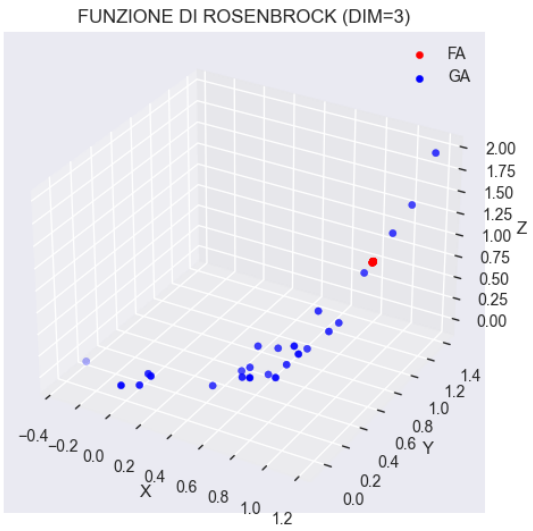

Figura 25: Scatter plot posizioni individui

Analizzando lo scatter plot in 3 dimensioni notiamo come il comportamento dei due algoritmi sia molto simile a quello osservato in 2 dimensioni. Anche in questo caso è dunque possibile osseravare come la "valle parabolica" ostacoli la ricerca dell'Algoritmo Genetico, mentre il comportamento dell'Algoritmo Firefly resta sostanzialmente invariato.

- **Dimensione = 10**

POPULATION SIZE	α	β_0	γ	θ
10	0.001	1.400	0.004	0.970

Tabella 17: Iperparametri FA

POPULATION SIZE	CXPB	MUTPB	η
300	0.55	0.04	1

Tabella 18: Iperparametri GA

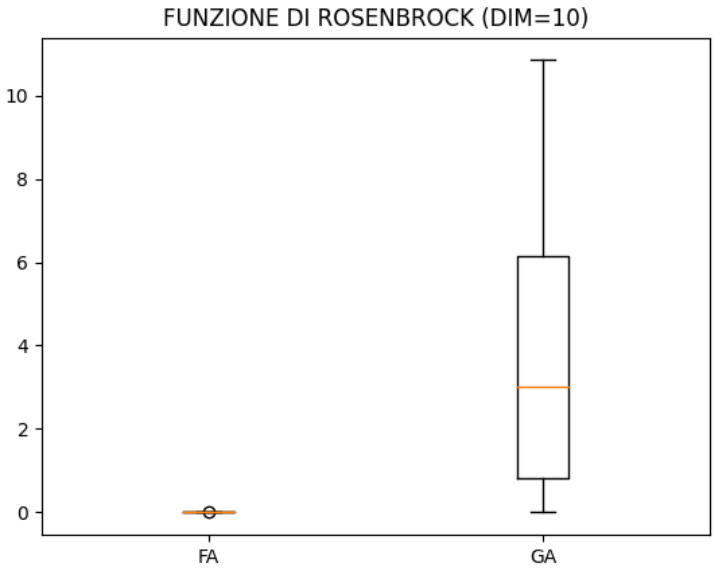

Figura 26: Box plot valori di fitness

Aumentando la dimensione dello spazio di ricerca si può osservare come l'Algoritmo Genetico non sia più in grado di determinare il minimo della funzione, mentre l'Algoritmo Firefly continua a predirlo correttamente. Uno dei vantaggi dell'Algoritmo Firefly rispetto agli Algoritmi Genetici è proprio l'elevata efficienza anche in spazi di ricerca con dimensioni elevate.

Conclusioni

In questo elaborato di tesi è stato approfondito l'Algoritmo Firefly, le cui prestazioni sono state confrontate con un'altra metaeuristica: l'Algoritmo Genetico. Entrambi gli algoritmi sono stati sviluppati in Python, con il supporto del framework *DEAP*. Dal confronto tra i due algoritmi è emerso come l'Algoritmo Firefly sia molto più performante nel caso di funzioni estramente complesse, grazie ad un metodo di ricerca basato sulla cooperazione e collaborazione degli individui, caratteristica peculiare degli algoritmi di Swarm Intelligence. Si è anche provato come l'Allgoritmo Firefly sia molto meno incline a restare intrappolato in minimi locali. Ciò è dovuto, in particolar modo, alla capacità di aggregazione degli individui, la cui autonoma suddivisione in sottogruppi permette una più ampia ed efficiente esplorazione dello spazio di ricerca. Risultano chiare le potenzialità di tale algoritmo, che seppur di recente implementazione, viene continuamente aggiornato da alcune varianti che ne migliorano l'efficienza.

Tra queste vogliamo ricordare una versione "quantistica", in cui il comportamento delle lucciole è reso probabilstico, secondo le leggi della meccanica quantistica, [13], oppure la versione "adattiva" dell'Algoritmo Firefly, basata sul cosiddetto *"Stochastic Inertia Weight"*, il quale varia in funzione del valore di fitness e delle posizioni delle lucciole. In questa versione la velocità di convergenza migliora sensibilmente [12]. Tra le tante altre, è stata introdotta anche una versione "ibrida", in cui si rilascia l'ipotesi secondo la quale le lucciole siano asessuate, introducendo la possibilità di accoppiamento [15].

Riferimenti bibliografici

[1] Gerardo Beni and Jing Wang. Swarm intelligence in cellular robotic systems. In *Robots and biological systems: towards a new bionics?*, pages 703–712. Springer, 1993.

[2] Eric Bonabeau, Marco Dorigo, and Guy Théraulaz. From natural to artificial swarm intelligence, 1999.

[3] Eric Bonabeau, Guy Theraulaz, and Marco Dorigo. *Swarm intelligence*. Springer, 1999.

[4] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28–39, 2006.

[5] Jesse Douglas. Solution of the problem of plateau. *Transactions of the American Mathematical Society*, 33(1):263–321, 1931.

[6] Gopi Krishna Durbhaka, Barani Selvaraj, and Anand Nayyar. Firefly swarm: metaheuristic swarm intelligence technique for mathematical optimization. In *Data management, analytics and innovation*, pages 457–466. Springer, 2019.

[7] Iztok Fister, Iztok Fister Jr, Xin-She Yang, and Janez Brest. Acomprehensivereviewoffirefly algorithms. *Swarm and Evolutionary Computation*, 13:34–46, 2013.

[8] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner Gardner, Marc Parizeau, and Christian Gagné. Deap: Evolutionary algorithms made easy. *The Journal of Machine Learning Research*, 13(1):2171–2175, 2012.

[9] Simon Garnier, Jacques Gautrais, and Guy Theraulaz. The biological principles of swarm intelligence. *Swarm intelligence*, 1(1):3–31, 2007.

[10] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of ICNN'95-international conference on neural networks*, volume 4, pages 1942–1948. IEEE, 1995.

[11] Bernhard Korte and Jens Vygen. Il problema del commesso viaggiatore. In *Ottimizzazione Combinatoria*, pages 561–597. Springer, 2011.

[12] Changnian Liu, Yafei Tian, Qiang Zhang, Jie Yuan, and Binbin Xue. Adaptive firefly optimization algorithm based on stochastic inertia weight. In *2013 Sixth International Symposium on Computational Intelligence and Design*, volume 1, pages 334–337. IEEE, 2013.

[13] A Manju and MJ Nigam. Firefly algorithm with fireflies having quantum behavior. In *2012 International Conference on Radar, Communication and Computing (ICRCC)*, pages 117–119. IEEE, 2012.

[14] Yuan-bin Mo, Yan-zhui Ma, and Qiao-yan Zheng. Optimal choice of parameters for firefly algorithm. In *2013 Fourth International Conference on Digital Manufacturing & Automation*, pages 887–892. IEEE, 2013.

[15] Xiangbo Qi, Sihan Zhu, and Hao Zhang. A hybrid firefly algorithm. In *2017 IEEE 2nd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, pages 287–291. IEEE, 2017.

[16] Anna Rita Sambucini. Un problema proposto da bernoulli: la brachistocrona. *Periodico di matematiche, Serie VIII*, 2:2, 2002.

[17] Ponnuthurai N Suganthan, Nikolaus Hansen, Jing J Liang, Kalyanmoy Deb, Ying-Ping Chen, Anne Auger, and Santosh Tiwari. Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization. *KanGAL report*, 2005005(2005):2005, 2005.

[18] S. Surjanovic and D. Bingham. Virtual library of simulation experiments: Test functions and datasets. Retrieved September 22, 2021, from `http://www.sfu.ca/~ssurjano`.

[19] Pei-Wei TSai, Jeng-Shyang Pan, Bin-Yih Liao, Shu-Chuan Chu, et al. Enhanced artificial bee colony optimization. *International Journal of Innovative Computing, Information and Control*, 5(12):5081–5092, 2009.

[20] Wikipedia. Brachistocrona — wikipedia, l'enciclopedia libera, 2020. [Online; in data 19-ottobre-2021].

[21] Eyal Wirsansky. Hands-on genetic algorithms with python: applying genetic algorithms to solve real-world deep learning and artificial intelligence problems, 2020.

[22] Xin-She Yang. Firefly algorithms for multimodal optimization. In *International symposium on stochastic algorithms*, pages 169–178. Springer, 2009.

A Algoritmi Genetici

Gli *Algoritmi Genetici* (noti come GA) sono una classe dei cosidetti *Algoritmi Evolutivi* (noti come EA), i quali sono metodi stocastici di ottimizzazione ispirati ai fenomeni biologici dell'evoluzione naturale.

Un algoritmo di questo tipo simula l'evoluzione di una popolazione di soluzioni candidate per il problema in esame, applicando iterativamente un insieme di operatori stocastici, come ad esempio la *selezione* , il *crossover*, e la *mutazione*.

Il processo che ne risulta tende a trovare soluzioni globalmente ottime per il problema oggetto in modo del tutto simile a come in Natura popolazioni di organismi si adattano all'ambiente che le circonda.

Il tipico workflow di un algoritmo genetico è del tipo:

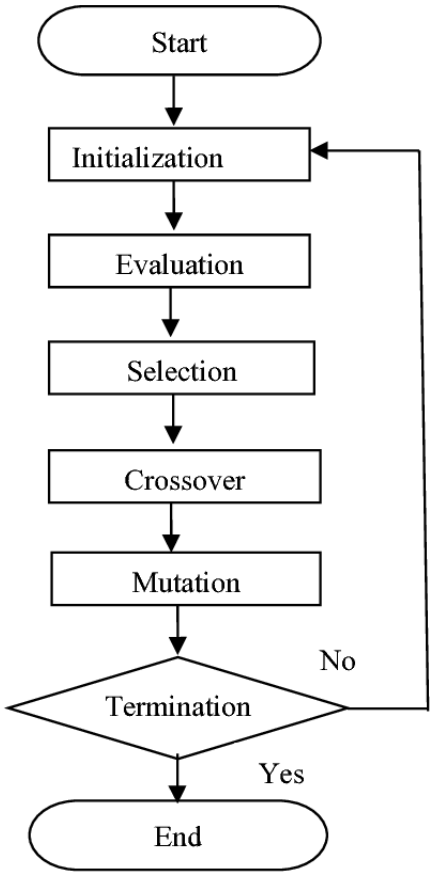

Figura 27: Workflow di un algoritmo genetico [21]

- **Inizializzazione della popolazione:** la *popolazione* è una collezione di possibili individui generati in modo casuale. Ciascun individuo è rappresentato da un *cromosoma*, ossia una collezione di geni. Ad esempio un cromosoma può essere espresso come una stringa di caratteri binari, oppure numeri decimali (a seconda del problema in esame)

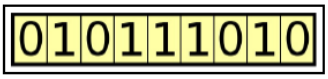

Figura 28: Esempio cromosoma binary-coded [21]

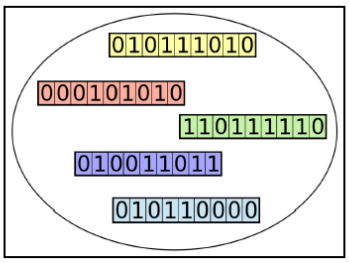

Figura 29: Esempio popolazione [21]

- **La funzione di fitness:** essa viene valutata per ciascun individuo della popolazione, e ciò viene fatto per ogni generazione ottenuta applicando gli operatori di selezione, mutazione e crossover. Gli individui caratterizzati da una migliore funzione di fitness rappresentano le possibili migliori soluzioni, ed hanno una maggiore possibilità di "riprodursi" ed essere ripresentate alle prossime generazioni Ne segue che è quindi la funzione di ftness a "guidare" in qualche modo l'ottimizzazione della popolazione al fne di ottenere soluzioni sempre migliori.

- **Selezione:** dopo aver valutato la funzione di fitness di ciascun individuo della popolazione, la selezione permette di determinare quali individui della popolazione si "riprodurranno" per originare la prossima generazione. Tra i possibili metodi di selezione ci sono:

  - *Roulette wheel selection:* con questo metodo la probabilità di selezionare un individuo è direttamente proporzionale al suo valore di fitness.
  - *Stochastic universal sampling:* Funziona in modo analogo alla Roulette wheel, solo che invece di applicare una selezione singola, si applica una selezione multipla.
  - *Rank based selection:* è simile alla roulette wheel selection, ma invece di usare la funzione di fitness per calcolare le probabilità di selezione per ciascun individuo, la funzione di fitness viene usata per ordinare gli individui, ai quali viene attribuito un punteggio in base alla classifica. Sulla base di questi punteggi sono calcolate le probabilità di selezione della roulette.

- *Tournament selection:* in ciascun turno di selezione, k individui sono scelti in modo casuale dalla popolazione. Tra questi, viene selezionato l'individuo col maggior valore di fitness.

- **Crossover:** per dare origine a nuovi individui, due "genitori" sono scelti dalla generazione corrente, e parte dei loro cromosomi viene interscambiata, per originare due nuovi cromosomi (e dunque due nuovi individui). Le varie modalità di crossover sono:

  - *single point crossover:* una posizione del cromosoma di entrambi i genitori viene scelta casualmente. In corrispondenza di essa ciascun cromosoma viene tagliato, e riassemblato con la parte rimossa dall'altro genitore.

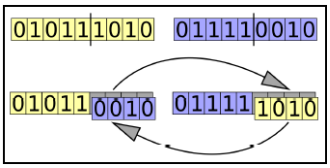

Figura 30: Single Point Crossover [21]

  - *Two-point* e *k-point crossover:* nel primo caso i punti di crossover sono due, mentre nell'altro caso sono k, con k intero. Analogamente al single point, le varie sezioni di un individuo vengono riassemblate con quelle dell'altro.

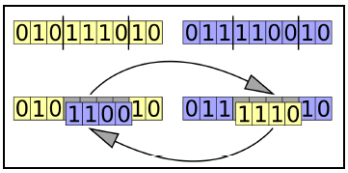

Figura 31: Two Point Crossover [21]

– *Uniform crossover:* ciascun gene dell'individuo figlio è casualmente assegnato scegliendo tra uno dei due corrispondenti geni dei genitori.

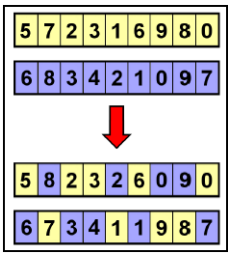

Figura 32: Uniform Crossover [21]

Gli esempi visti in precedenza sono usati specialmente nel caso di cromosomi a valori binari o interi, o nei casi in cui lo spazio delle soluzioni sia discreto. Quando invece lo spazio delle soluzioni è continuo, si utilizzano i seguenti operatori di crossover:

– *Blend Crossover(BLX):* ciascun gene figlio è generato casualmente in un intervallo calcolato a partire dai geni genitori secondo la formula:

$$[\ parent_1 - (parent_2 - parent_1),\ parent_2 + (parent_2 - parent_1)\ ] \tag{A.1}$$

Solitamente $\alpha = 0,5$, in modo che l'intervallo di selzione abbia un'ampiezza doppia rispetto all'intervallo tra i genitori

– *Simulated binary crossover(SBX):* i due individui figli sono generati dagli individui genitori a partire dalla seguenti formule

$$offspring_1 = \frac{1}{2}[(1+\beta)parent_1 + (1-\beta)parent_2] \tag{A.2}$$

$$offspring_2 = \frac{1}{2}[(1-\beta)parent_1 + (1+\beta)parent_2] \tag{A.3}$$

In questo modo la media dei valori di fitness dei figli è uguale a quella dei genitori. Il valore di β è anche chiamato *spread factor*, ed è calcolato usando una combinazione di un numero scelto casualmente e un parametro predeterminato η, noto come *crowding factor*. Valori comuni di β sono compresi tra 10 e 20

- **Mutazione:** essa è applicata agli individui della prole con lo scopo di aggiornare periodicamente e casualmente la popolazione, introdurre nuovi modelli nei cromosomi e incoraggiare la ricerca in aree inesplorate dello spazio delle soluzioni. La probabilità con la quale l'operatore è applicato è molto bassa, onde evitare il rischio di convertire l'algoritmo genetico in un algoritmo di ricerca causale. Tra i metodi di mutazioni più comuni troviamo:

– *Flip bit mutation:* è specialmente utilizzata quando si ha a che fare con cromosomi binary-coded come quelli della figura 28. Questa tecnica consiste nel selezionare casualmente uno o più geni del cromosoma, e complementarlo.

Figura 33: Flip bit mutation [21]

– *Swap mutation:* si applica specialmente a cromosomi del tipo binary o integer-based. In questi casi due geni sono selezionati in maniera casuale, e i loro valori vengono scambiati, come mostrato di seguito.

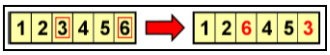

Figura 34: Swap mutation [21]

– *Inversion mutation* e *Scramble mutation:* quando la prima mutazione è applicata ad un cromosoma, una sequenza di geni è scelta in modo casuale, e il loro ordine è invertito. Nel secondo caso invece la sequenza di geni casualmente selezionata, non è invertita, bensì mischiata. Anche in questo caso, queste mutazioni sono applicate nel caso di cromosomi binary o integer-based.

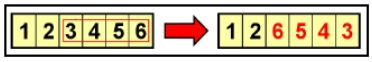

Figura 35: Inversion mutation [21]

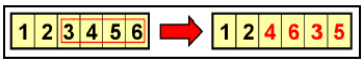

Figura 36: Scramble mutation [21]

– *Gaussian muatation:* un'opzione per applicare la mutazione nel caso di algoritmi real-coded potrebbe essere sostituire uno o più valori reali con altri generati casualmente. Una conseguenza di questo approccio porterebbe però portare a nuovi individui che non abbiano relazione con quelli precedenti. La soluzione consiste nel generare casualmente un numero reale, usando una distribuzione normale, con valor medio nullo.

Ringraziamenti

Un doveroso ringraziamento va alla professoressa Autilia Vitiello.

La ringrazio per la sua infinita disponibilità e pazienza, e per il suo prezioso contributo, senza il quale non sarebbe stato possibile realizzare questo lavoro di tesi.

A miei genitori.

A papà per avermi sempre incoraggiato con il suo *"tutto passa"*,

ma soprattutto a mamma, per essere sempre stata il mio più grande supporto. Questo traguardo è anche il tuo.

Ad Alessandro, il più caro amico e compagno di studi che potessi mai incontrare. Non aggiungo altro, ma sappi che se sono arrivato qui, è anche grazie a te.

Ad Edoardo, fedele compagno di laboratorio, e non solo.

A Rodolfo, Giovanni, Rocco, Carlo, Davide, Leonardo, Michele, Daniele, Giorgio, Martina, Anja, Viola, Carla, Alessia. Grazie per aver condiviso con me questo percorso.