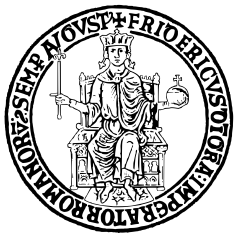
UNIVERSITÀ DEGLI STUDI DI NAPOLI
"FEDERICO II"



Scuola Politecnica e delle Scienze di Base

Area Didattica di Scienze Matematiche Fisiche e Naturali

Dipartimento di Fisica "Ettore Pancini"

Laurea Triennale in Fisica

Implementazione e valutazione dell'Algoritmo Big Bang-Big Crunch

Relatori:
Dott.ssa Autilia Vitiello

Candidato:
Sannino Raffaele
Matr. *N*85001127

Anno Accademico 2020/2021

Indice

Introduzione

I fenomeni naturali sono le principali fonti d'ispirazione nella creazione di metaeuristiche volte a risolvere problemi di ottimizzazione. Molte di queste traggono spunto da fenomeni della fisica. Queste metaeuristiche risultano molto efficaci quando si ha a che fare con funzioni particolarmente irregolari.

Oggetto di studio principale di questo elaborato di tesi è l'Algoritmo Big Bang-Big Crunch.

L'Algoritmo Big Bang-Big Crunch (BB-BC) si ispira ad una delle teorie attualmente riconosciute sull'evoluzione dell'universo. Il Big Bang è la fase in cui la materia si è espansa, a partire da un punto a densità e temperatura altissime, permettendo così successivamente la formazione degli elementi a noi noti. La fase di Big Crunch invece, è quella in cui i corpi per mezzo dell'attrazione gravitazionale convergono in un unico punto. Questo modello cosmologico può in qualche modo essere tradotto in termini di algoritmo di ottimizzazione se si pensa che la fase del Big Bang aumenta la casualità, e quindi la capacità di esplorazione dello spazio di ricerca, mentre la fase di Big Crunch aumenta l'ottimizzazione delle possibili soluzioni.

L'Algoritmo BB-BC ha oggi diverse occasioni di essere applicato: dalla risoluzione di problemi NP classici come il *graph partitioning* e il *Traveling Salesman Problem*, alle applicazioni nel campo scientifico ed industriale, che comprendono il software testing, l'analisi modale nel campo dell'ingegneria civile e l'analisi costi-benefici sulla costruzione di un impianto ad alto impatto ambientale.

In questo lavoro di tesi è stato implementato un Algoritmo BB-BC ed è stato confrontato con un Algoritmo Genetico (GA). Per realizzare questo confronto, si sono applicati questi due algoritmi su alcune funzioni di benchmark: la funzione della sfera, di Ackley, di Levy, di Rastrigin e di Rosenbrock. Nel fare ciò si sono prima determinati i migliori iperparametri per i due algoritmi funzione per funzione, attraverso i test statistici di Friedman e di Holm, e in seguito si è eseguita la comparazione tra i due algoritmi sfruttando il test statistico di Wilcoxon.

Nel Capitolo 1 vengono discussi il concetto di problema di ottimizzazione e gli Algoritmi Genetici. Nel Capitolo 2 si presentano diversi esempi di algoritmi physics-inspired e ci si sofferma particolarmente sull'Algoritmo BB-BC. Nel Capitolo 3 si discute prima di Python e del framework DEAP, illustrandone la struttura generale e i vantaggi che derivano dal suo utilizzo, e poi si descrivono i test statistici utilizzati nell'analisi delle performance dei due algoritmi, cioè i te-

st di Friedman, di Holm e di Wilcoxon. Per concludere, nel Capitolo 4 vengono presentate le funzioni di benchmark utilizzate e poi vengono illustrati i risultati ottenuti dai vari test, potendo così giungere a delle conclusioni riguardanti l'efficienza dell'Algoritmo Big Bang-Big Crunch rispetto all'Algoritmo Genetico.

Capitolo 1

Problemi di ottimizzazione e algoritmi evolutivi

1.1 Cos'è un 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 a massimizzare o minimizzare una funzione, detta "funzione obiettivo", fissato un opportuno dominio contenente tutte le soluzioni possibili. Formalmente diremo quindi che il problema consiste nella ricerca del punto di ottimo x^*, cioè di minimo o massimo, della funzione obiettivo $f : D \subseteq \mathbb{R}^n \longrightarrow \mathbb{R}$, dove D è il dominio [2].

Prenderemo in considerazione solo problemi di minimizzazione. Si ricorda che per punto di minimo globale si intende un punto $x^* \in D$ tale che $f(x^*) \leq f(x) \quad \forall x \in D$. Si definisce invece punto di minimo locale un punto $\overline{x} \in D$ se esiste un intorno $I(\overline{x})$ tale che $f(\overline{x}) \leq f(x) \quad \forall x \in I \cap D$

Ad un problema di ottimizzazione quindi ne è associato uno di carattere matematico per cui bisogna sviluppare una strategia di risoluzione.

1.2 Algoritmi evolutivi

Gli Algoritmi Evolutivi (anche noti come EA) sono metodi metaeuristici, cioè tecniche di risoluzione basate su processi stocastici in grado di trovare soluzioni approssimate a problemi computazionali. Il loro funzionamento, come suggerisce il nome, si ispira ai principi evolutivi naturali descritti da Charles Darwin.

In un EA le soluzioni del problema sono modellizzate come individui che formano una popolazione. A seconda del contesto in cui vengono considerati, gli individui vengono chiamati in diverso modo: fenotipi nel contesto del problema originale che si vuole ottimizzare e genotipi o cromosomi quando sono codificati nell'algoritmo. Le variabili che compongono il cromosoma sono detti geni mentre

un'istanza possibile di un gene è detto allele. L'EA agisce nello spazio dei genotipi che può essere molto diverso da quello dei fenotipi.

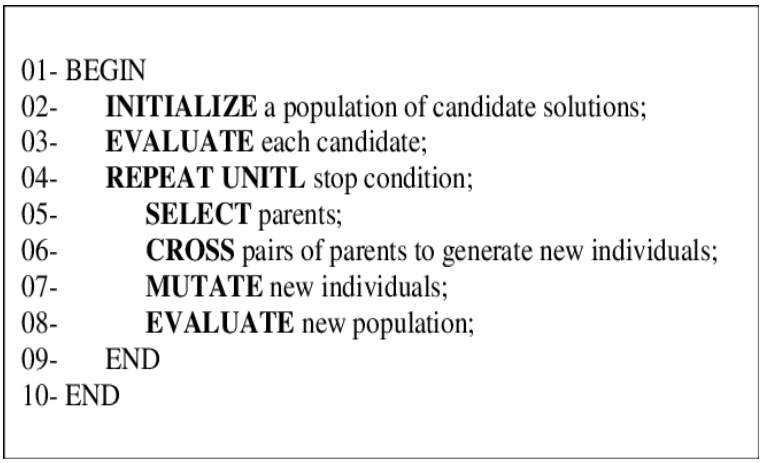


Figura 1.1: Pseudo-codice di un algoritmo evolutivo [4]

Per inizializzare la popolazione si generano casualmente i suoi individui, generalmente fatti rientrare in un certo range di valori possibili. In seguito si valutano gli individui attraverso una funzione di fitness (fitness function) che attribuisce ad ognuno di loro uno o più valori a seconda della loro bontà come soluzioni. Ne segue che è quindi la funzione di fitness a "guidare" in qualche modo l'ottimizzazione della popolazione al fine di ottenere soluzioni sempre migliori.

Ciò che rende possibile il processo di evoluzione della popolazione sono gli operatori di variazione (riproduzione e mutazione) e quelli di selezione. I primi servono ad accrescere la diversità della popolazione, i secondi ad aumentare la qualità. Essi agiscono in questo modo: dopo la valutazione, di cui abbiamo parlato prima, si applica un meccanismo di selezione che, confrontando i valori di fitness dei vari individui, estrae dalla popolazione gli individui destinati a generare la nuova popolazione. Questo operatore si chiama *parent selection operator*. È fondamentale specificare che si tratta di un'operazione stocastica: agli individui è associata una **probabilità** di successo nella selezione della parent selection.

Dopo di ciò viene applicato l'operatore di crossover che agisce su una coppia di individui selezionati dalla parent selection, creando così uno o più nuovi individui frutto della ricombinazione dei geni della coppia. L'insieme degli individui creati in questo modo è detto *offspring*.

Terminata questa fase, giunge il momento di applicare l'operatore di mutazione, che attua stocasticamente (e quindi con una certa probabilità) una variazione dei geni degli individui.

Fatto questo, avviene una nuova selezione, detta *survivor selection*, che secondo criteri deterministici impostati a priori sceglie gli individui che formeranno

la nuova popolazione e che seguirà di nuovo il ciclo illustrato in questo paragrafo fino al raggiungimento di un criterio di terminazione che varierà a seconda delle esigenze e del tipo di problema con cui si ha a che fare. Il criterio può essere ad esempio, il numero massimo di valutazioni di fitness della popolazione o il tempo massimo di elaborazione della CPU o il raggiungimento di una soglia minima di errore del valore di fitness rispetto ad un certo valore.

Gli algoritmi evolutivi vengono spesso sfruttati per affrontare problemi di ottimizzazione. Il loro utilizzo, anche grazie agli strumenti messi a disposizione dal linguaggio di programmazione Python e dal framework DEAP che saranno trattati in seguito, è semplice e immediato, ed ha una leggibilità molto chiara. Paragonati agli algoritmi classici, dimostrano di essere molto efficienti in problemi aventi a che fare con funzioni discontinue, non differenziabili e multimodali, e di essere meno efficaci rispetto a problemi lineari, quadratici, convessi, unimodali e separabili.

Al netto delle considerazioni appena fatte, rimane da evidenziare un limite che caratterizza fortemente gli algoritmi evolutivi, ovvero il problema della convergenza prematura: spesso, quando si ha a che fare con problemi con un gran numero di punti di ottimo locale, si corre il rischio che i cromosomi convergano verso uno di essi piuttosto che verso l'ottimo globale. La questione a cui quindi l'utente deve prestare particolare attenzione è quella del giusto equilibrio tra "ottimizzazione" e "diversità" della popolazione. Con quest'ultima espressione si intende la varietà tra le soluzione del problema all'interno della popolazione.

Il funzionamento generale di un algoritmo evolutivo è stato appena illustrato, ma ne esistono diversi tipi che si distinguono tra loro per vari aspetti, che possono riguardare le istanze dei vari operatori di selezione e di variazione oppure i modi di rappresentazione degli individui. Alcune versioni, tra le più note, di EA sono:

- Algoritmi Genetici (GA)
- Evoluzione Differenziale (EA)
- Particle Swarm Optimization (PSO)

1.2.1 Algoritmi Genetici

L'Algoritmo Genetico (GA) è forse il tipo più popolare di EA. Nella Tabella 1.1 sono rappresentate le caratteristiche di un tipico GA.

Tabella 1.1: Caratteristiche dell'Algoritmo Genetico

Rappresentazione	Bit-strings
Ricombinazione	1-Point Crossover
Mutazione	Bit flip
Parent Selection	Fitness proportional
Survival Selection	Generational

Come si può notare, i genotipi sono rappresentati come stringhe di bit. Si tratta di una delle prime rappresentazioni adottate per molti GA.

L'operatore di crossover scelto prende una coppia di cromosomi e, posta L la lunghezza dell'individuo, sceglie un valore random compreso tra 1 e $L-1$: questo valore rappresenta la posizione del gene lungo cui avviene il taglio che divide in due i due individui. Fatto questo, l'operatore accoppia la prima parte del primo individuo con la seconda parte del secondo individuo e analogamente con gli altri 2 "pezzi" di codice genetico.

Figura 1.2: 1-Point Crossover [4]

Il 1-Point Crossover appena descritto può chiaramente essere generalizzato con il n-Points Crossover che effettua n tagli.

Figura 1.3: 2-Point Crossover [4]

Un altro tipo di crossover molto popolare è lo *uniform crossover* che consiste nel prendere una coppia di individui, generare in maniera random una stringa di L bit che prende il nome di *marcatore*, scegliere un parametro p (usualmente pari a 0,5) compreso tra 0 e 1 e confrontare ogni bit del marcatore con p: se $p > bit_i$ ($p < bit_i$) si seleziona il bit i-esimo del primo individuo (secondo individuo). Con i bit selezionati per ogni i si crea il "primo figlio" mentre quelli non selezionati formeranno il "secondo figlio".

Per quanto riguarda l'operatore di mutazione, si tratta di un Bit-Flip: ad ogni gene di ogni individuo per cui si applica la mutazione è associata la probabilità (preimpostata) p_m che esso sia invertito (sarà quindi uguale a 1 se era 0 e viceversa).

Per quanto riguarda la parent selection, l'algoritmo genetico fa uso di un metodo "fitness-proportional" (FPS) cioè, come suggerisce l'espressione, rende maggiormente probabile per gli individui con fitness alto di essere selezionati

Figura 1.4: Esempio di Bit Flip [4]

per il crossover. La probabilità di selezione di un individuo i attraverso il FPS è:

$$P_{FPS}(i) = \frac{f_i}{\sum_{j=1}^{N} f_j} \tag{1.1}$$

In particolare, la parent selection si esplica col metodo Roulette Wheel. Quest'ultimo si può immaginare considerando un disco diviso in n parti, dove n è il numero di individui, in cui ogni parte i-esima è tanto ampia quanto alta è la fitness dell'individuo i-esimo. Va da sé che la probabilità di estrarre un valore di una parte ampia è maggiore rispetto alle altre.

Un altro tipo di parent selection è il *Tournament* che seleziona il migliore tra un numero fissato di individui (il *tournsize*) per un numero k di volte.

La survivor selection invece è Generational, ovvero consiste nel sostituire tutti gli individui della vecchia popolazione con quelli della nuova generazione. Un diverso approccio, spesso utilizzato, è quello del *Fitness-Based Replacement* che fa sì che vengano selezionati gli individui con fitness più alta, indifferentemente dalla generazione a cui appartengono. In alcuni casi può dimostrarsi molto utile fare un ibrido tra i due approcci.

Un altro modo di rappresentare gli individui in un GA è con valori reali. In questo modo i valori che costituiscono i geni posso essere estratti da una distribuzione continua piuttosto che discreta. Il genotipo di una soluzione candidata con k geni in questo caso è un vettore $\mathbf{v}$:

$$\mathbf{v} = [x_1, \ldots, x_k] \quad , \quad x_i \in \mathbb{R} \tag{1.2}$$

Ciò permette di adottare operatori diversi rispetto a quelli prima illustrati. Si mostra a titolo di esempio due possibili operatori di crossover e mutazione:

Per il crossover si può utilizzare il *Blend Crossover*. Esso crea un nuovo individuo $\overline{z}$ a partire da due genitori $\overline{x}$ e $\overline{y}$ come segue:

$$z_i = u \tag{1.3}$$

dove u è un numero random estratto dall'intervallo $[x_i - \alpha(y_i - x_i), y_i + \alpha(y_i - x_i)]$, assumendo che $x_i \leq y_i$. Il valore migliore di α riportato in letteratura è $\alpha = 0.5$.

Un possibile operatore di mutazione, molto semplice, è lo *Uniform Mutation* che cambia il valore dell'allele di ogni gene con un numero random estratto da un opportuno intervallo limitato.

Esistono diversi tipi di GA, il quale varia a seconda degli operatori e della rappresentazione adottati. L'istanza presentata in questo caso è detta *Simple Genetic Algorithm* (SGA).

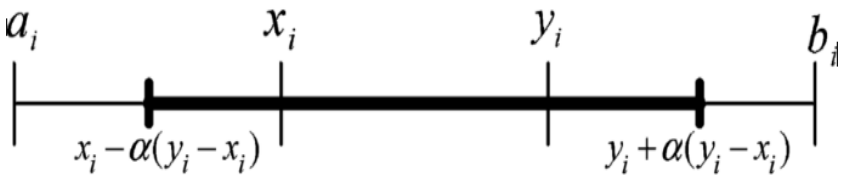


Figura 1.5: Blend Crossover

Capitolo 2

Algoritmi di ottimizzazione "Physics-Inspired" e Algoritmo Big Bang-Big Crunch

2.1 Algoritmi "Physics-Inspired"

I tradizionali algoritmi evolutivi sono metodi metaeuristici che si ispirano a processi naturali quali quelli illustrati da Charles Darwin nella sua celebre teoria o a modelli sociali di adattamento di diverse popolazioni animali (Particle Swarm Optimization). In questo capitolo si vuole accennare ad altri tipi di metodi metaeuristici che traggono spunto da alcuni fenomeni o modelli della fisica [1]. Alcuni di questi sono:

1. *Gravitational Search Algorithms*
2. *Quantum Genetic Algorithms*
3. *Simulated annealing*
4. *Big Bang-Big Crunch Algorithms*

Nelle pagine seguenti si illustra in maniera generale il funzionamento dei suddetti algoritmi, soffermandocisi maggiormente sull'Algoritmo Big Bang-Big Crunch che è oggetto principale di questo lavoro di tesi.

1. Il Gravitational Search Algorithm (GSA) trae ispirazione dalla meccanica newtoniana e, più precisamente, dalla legge di gravitazione universale: ogni particella nell'universo esercita verso ogni altra particella una forza attrattiva direttamente proporzionale al prodotto delle loro masse e

inversamente proporzionale al quadrato della loro distanza. Cioè:

$$\mathbf{F} = G\frac{m_1 m_2}{r^3}\mathbf{r} \tag{2.1}$$

Nel GSA l'insieme delle soluzioni candidate è rappresentato da un sistema isolato di particelle. Sono le masse di quest'ultime ad essere effettivamente possibili soluzioni del problema. L'idea è che le forze che le particelle

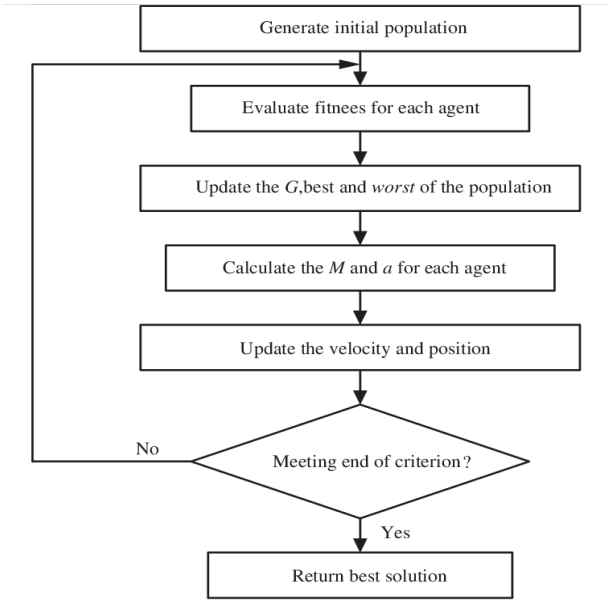


Figura 2.1: Work-flow del GSA [10]

esercitano tra loro permettano l'esplorazione dello spazio di ricerca e quindi che la forza gravitazionale diventi uno strumento di "trasferimento di informazione" all'interno del sistema. In quest'ottica le particelle aventi le masse più grandi sono anche le migliori soluzioni candidate.

La Figura 2.1 mette in luce il *work-flow* dell'algoritmo.

2. Il *Quantum Genetic Algorithm* (QGA) è un algoritmo che combina il GA e il *quantum computing*. Il funzionamento generale del QGA è lo stesso del GA, tuttavia, mentre in quest'ultimo i cromosomi sono rappresentati come numeri binari, nel QGA sono rappresentati come vettori di qubits (*registri quantistici*) (Figura 2.2)[7][8]. In questo modo, un cromosoma può davvero essere una sovrapposizione dei possibili stati. La struttura di un QGA è illustrata nella Figura 2.3.

3. Il Simulated Annealing (SA) non è una metaeuristica *population-based* come le altre presentate in questa tesi: in altri termini, non è la popolazione

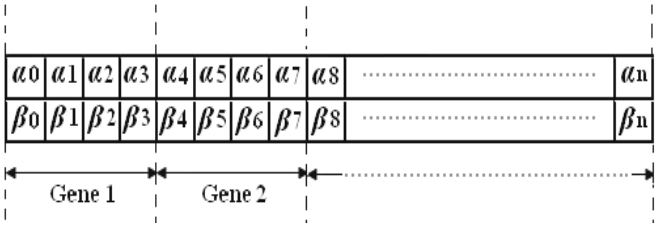


Figura 2.2: Registro quantistico [7]

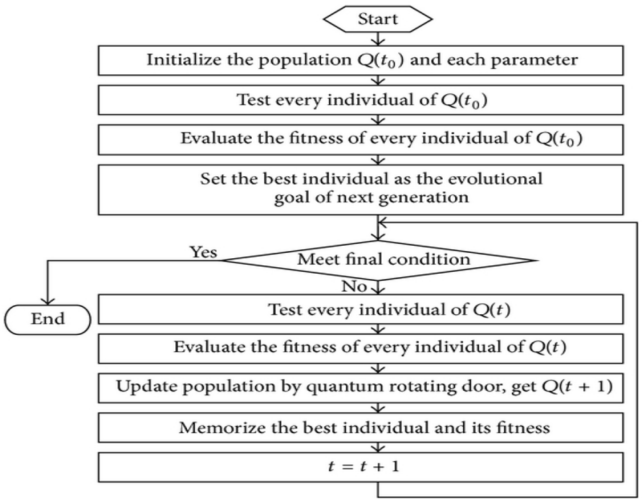


Figura 2.3: Work-flow del QGA [18]

intera che evolve ma un singolo individuo. Ad ogni modo, rientra nella tipologia degli algoritmi *physics-inspired*.

Esso si basa sul fenomeno del lento raffreddamento dei metalli, che a livello microscopico si traduce ad una progressiva riduzione dei movimenti atomici fino al raggiungimento di uno stato energetico più basso. In analogia col fenomeno fisico, nell'algoritmo esiste un parametro dipendente dal tempo chiamato Temperatura e le soluzioni candidate variano significativamente quando questa è alta, viceversa, se la Temperatura è bassa, tendono a rimanere fisse [16].

Il Simulated Annealing ha la virtù di essere molto efficace nell'evitare convergenze premature verso ottimi locali, in quanto permette movidenti delle soluzioni verso punti con valori di fitness inferiori a quelli della soluzione considerata.

2.2 Algoritmo Big Bang-Big Crunch

L'Algoritmo Big Bang-Big Crunch (BB-BC Algorithm) è una metaeuristica che trae ispirazione dai modelli cosmologici del Big Bang e del Big Crunch [5][19]. Secondo questi modelli l'universo iniziò a espandersi a partire da uno stato iniziale ad altissima temperatura e densità (Big Bang). Questa espansione continua ancora oggi e continuerà fino a quando, raggiunto un certo limite, comincerà il processo inverso (Big Crunch): l'universo si contrarrà fino al punto di collassare su se stesso. All'interno dell'algoritmo quindi, questi due processi saranno sviluppati con due opportuni operazioni: il Big Bang rappresenterà l'operazione di esplorazione, in cui la variabilità tra le soluzioni candidate aumenta, mentre quella di ottimizzazione sarà rappresentata dal Big Crunch. Nella prima fase la casualità è predominante in maniera analoga a come lo è la dissipazione energetica nel processo fisico del Big Bang, diversamente nella seconda fase la convergenza delle soluzioni candidate ad un punto di ottimo può essere vista equivalente all'attrazione gravitazionale.

Per quanto riguarda la popolazione, essa è composta da punti materiali (gli individui) le cui posizioni all'interno dello spazio di ricerca sono le soluzioni candidate.

2.2.1 Tipico workflow di un Algoritmo BB-BC

Innanzitutto si genera casualmente una popolazione composta da N posizioni[1], le quali sono rappresentate da vettori $\mathbf{x}^i$ di dimensione D le cui componenti sono valori reali compresi in un dominio preimpostato. Dopo, si calcolano i valori della funzione di fitness per ciascun punto. Fatto ciò, parte la fase Big Crunch, che consiste nel calcolare il centro di massa del sistema di particelle creato.

[1] in questo elaborato si useranno i termini "posizioni", "particelle" e "punti" interscambiabilmente

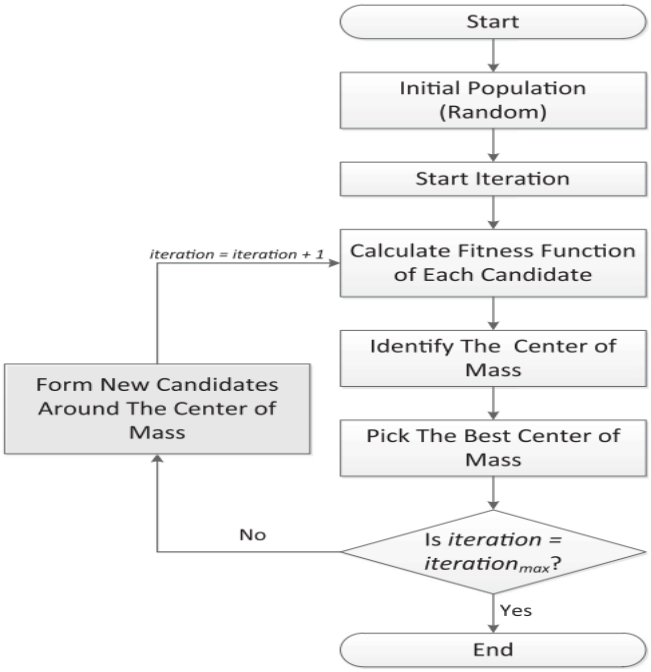


Figura 2.4: Work-flow del BB-BC [17]

Nel contesto di un algoritmo BB-BC volto a trovare i punti di minimo di una funzione, le masse dei punti materiali non sono altro che l'inverso dei valori che la funzione di fitness assume per le posizioni dei suddetti punti. Indicando con $\mathbf{x}^c$ il centro di massa, sarà:

$$\mathbf{x}^c = \frac{\sum_{i=1}^{N} \frac{\mathbf{x}^i}{f^i}}{\sum_{i=1}^{N} \frac{1}{f^i}} \tag{2.2}$$

dove f^i è il valore della funzione di fitness dell'i-esimo punto. Come si vede, un'operazione siffatta coinvolge tutti gli elementi della popolazione, a differenza del canonico GA che nella fase di crossover combina gli individui due a due.

Infine si generano i nuovi punti che all'interno dello spazio di ricerca si troveranno attorno al centro di massa e che si formeranno sommando algebricamente valori random r ,compresi in un dominio opportunamente scelto, che diminuiranno all'aumentare delle iterazioni del ciclo. Formalmente:

$$\mathbf{x}^{new} = \mathbf{x}^c + \frac{lr}{k} \tag{2.3}$$

dove l è l'ampiezza del dominio opportunamente scelto e k è l'indice del ciclo che si sta svolgendo.

È possibile creare delle varianti dell'Algoritmo BB-BC modificando il calcolo del centro di massa attraverso l'utilizzo della migliore soluzione candidata trovata fino a quel momento $\mathbf{x}^{best}$ e di un parametro β che ne gestisce "l'influenza" nel risultato finale:

$$\mathbf{x}^{new} = \beta\mathbf{x}^c + (1-\beta)\mathbf{x}^{best} + \frac{lr}{k} \tag{2.4}$$

L'Algoritmo si conclude quando si raggiunge il criterio di terminazione, che usualmente è il raggiungimento di un certo numero di valutazioni di fitness.

2.2.2 Esempi di impiego dell'Algoritmo BB-BC

L'algoritmo BB-BC si è dimostrato utile ed efficiente in diversi campi.

Esistono studi che testimoniano l'utilizzo dell'Algoritmo BB-BC per determinare e localizzare danni strutturali a edifici, infrastrutture o anche a macchinari dell'industria meccanica, per correggere così eventuali difetti di progettazione [15]. In questi studi le strutture vengono modellizzate attraverso il metodo degli elementi finiti e, siccome danni strutturali ne modificano le proprietà dinamiche, studiando questi ultimi, e in particolare i modi vibrazionali e le frequenze di risonanza, si può risalire alle posizioni dei danni e alla loro entità.

Un altro impiego interessante è quello relativo a problemi di ottimizzazione aventi lo scopo di rendere più efficienti gli edifici nel consumo energetico, cercando quindi di eliminare o ridurre gli sprechi [9]. È infatti noto che la realizzazione di edifici energeticamente efficienti richiede quasi sempre l'impiego di materiali isolanti e di un sistema di riscaldamento, ventilazione e condizionamento dell'aria particolarmente costosi, che possono impattare pesantemente sul costo dell'edificio. Si tratta dunque di risolvere un problema di ottimizzazione

multi-obiettivo. L'Algoritmo BB-BC si è dimostrato efficace nel minimizzare la spesa totale, cioè la somma tra la spesa in termini economici e quella in termini ambientali.

È documentato l'utilizzo dell'Algoritmo BB-BC anche nel campo del software testing [12].

Capitolo 3

Strumenti utilizzati

3.1 Python e DEAP

L'implementazione degli algoritmi BB-BC e GA, il cui confronto è oggetto principale di questo elaborato di tesi, è stata realizzata su Python.

Esistono diversi motivi per cui è risultato utile sfruttare Python: è un linguaggio di programmazione ad alto livello, orientato agli oggetti e caratterizzato da dinamicità, semplicità e flessibilità. Il fatto di essere così ad alto livello lo rende vicino ad una sorta di pseudo-codice e, di conseguenza, molto chiaro. Inoltre la sua enorme popolarità ha fatto sì che si creasse una vastissima comunità di utenti in grado di, attraverso forum e siti web dedicati, condividere informazioni utili alla risoluzione di eventuali problemi di programmazione.

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 è DEAP (**Distributed Evolutionary Algorithms in Python**) sviluppato al **Computer Vision and System Laboratory (CVSL)** nell'Università Laval in Quebec City.

DEAP è un recente framework nato per lo sviluppo di Algoritmi Evolutivi, per la prototipazione rapida e il test delle idee.

Gli Algoritmi Evolutivi possono assumere delle forme molto articolate e poco leggibili, e per questo è facile che un framework, per quanto ben progettato, possa rivelarsi complicato da comprendere. A causa di ciò, per facilitare la programmazione, questi framework nascondono quanto più possibile i dettagli dell'implementazione e per questo motivo sono definiti "black-box" framework.

Se da un lato questa forma a "scatola nera" rende più "puliti" i codici, dall'altro, all'aumentare della loro complessità, diventano più ermetici, soprattutto per programmatori meno esperti. DEAP riesce a risolvere questo problema in quanto punta a rendere gli algoritmi espliciti e le strutture dati chiare, combinando la flessibilità e la potenza di Python con un core chiaro e versatile in cui sono implementate le componenti fondamentali degli EA, che facilitano la programmazione grazie alla loro semplicità. A differenza di altri framework ad esempio,

DEAP permette la creazione degli individui, permettendo così all'utente di avere maggiormente sotto controllo le caratteristiche dell'algoritmo.

In sostanza, il framework si fonda su tre aspetti principali:

- la chiarezza delle strutture dati, che devono rendere semplice l'implementazione degli algoritmi ed essere facilmente personalizzabili;
- la natura esplicita e leggibile degli operatori di selezione ed esplorazione. Questi, fanno uso di alcuni parametri caratteristici che devono essere facilmente modificabili, in quanto la loro efficacia dipende fortemente dal problema che si vuole risolvere. In tal senso, maggiori sono le possibilità di parametrizzazione da parte dell'utente, maggiori sono le opportunità di sfruttare le potenzialità dell'algoritmo;
- la presenza di meccanismi in grado di sviluppare facilmente paradigmi di distribuzione. Questa esigenza è dovuta al fatto che gli EA sono il più delle volte paralleli, richiedendo il calcolo simultaneo delle funzioni di fitness.

Per adempiere correttamente ai propositi appena illustrati, DEAP viene spesso affiancato dal modulo SCOOP.

3.2 Struttura e tools di DEAP

La struttura di DEAP pone le sue basi nella presenza di alcuni oggetti in grado di definire parti specifiche dell'algoritmo evolutivo.

Uno di questi è l'oggetto **creator**. Attraverso quest'ultimo, DEAP non vincola il programmatore a fare uso di tipi o funzioni predefinite, ma permette di crearne autonomamente e con grande semplicità. In particolar modo, sfruttando il metodo **create** è possibile creare le classi relative alla funzione di fitness e all'individuo.

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

Codice 3.1: metodo create

Nel Codice 3.1 è mostrata la sintassi del metodo create. Esso 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 altri semplicemente specificandoli come altri argomenti.

Un altro strumento fondamentale messo a disposizione da DEAP è il **toolbox**, un contenitore di operatori e altri oggetti necessari all'implementazione dell'algoritmo. Toolbox fa uso del metodo **register**, la cui sintassi è mostrata nel Codice 3.2:

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

Codice 3.2: metodo register

come primo argomento va inserito il nome dell'operatore che si vuole registrare nel toolbox. Il secondo argomento è la funzione su cui si basa l'alias. Gli argomenti successivi sono semplici attributi riferiti alla funzione chiamata.

Definire gli operatori di crossover, mutazione e selezione diventa quindi immediato con il metodo register: una volta che si è scelti gli operatori che si vuole, basta registrarli nel toolbox (Codice 3.3).

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

Codice 3.3: esempio di implementazione degli operatori

Per inizializzare la popolazione si può usare il metodo **initRepeat**:

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

Codice 3.4: metodo initRepeat

container è l'oggetto dentro cui inserire i dati generati da *func* che deve ripetersi un numero n di volte.

DEAP inoltre, contiene anche un modulo che fornisce degli esempi di algoritmi evolutivi già implementati, ovvero il modulo **algorithms**, e il modulo **dtm** (Distributed Task Manager) che mette a disposizione delle funzioni da sostituire a quelle canoniche di Python come *apply* o *map*.

3.3 Test statistici non parametrici

In anni recenti è cresciuto con particolare vigore l'interesse per le analisi sperimentali nel campo degli algoritmi evolutivi e quindi per gli strumenti statistici necessari alla loro valutazione.

I test statistici necessari alla valutazione delle performance degli algoritmi si dividono in due classi: parametrici e non parametrici. In questo elaborato di tesi si sono utilizzati questi ultimi. Questo perchè i test parametrici si applicano quando i dati da analizzare sono indipendenti, seguono un andamento gaussiano e rispettano l'omoschedasticità, condizioni non rispettate dai dati prodotti in questa tesi.

I test non parametrici hanno il vantaggio di potersi applicare ad una vasta classe di campioni e di rivelarsi spesso molto semplici da implementare.

Il teorema del *No Free Lunch* dimostra l'impossibilità di trovare un algoritmo che si comporti meglio di tutti gli altri per un qualsiasi problema. Di conseguenza, il confronto tra le performance di due diversi algoritmi deve essere svolto per ogni singolo problema.

I test statistici di cui si è fatto uso in questa tesi sono il test di Friedman, il test di Holm e il *Wilcoxon matched-pairs signed rank test* [6] [3].

3.3.1 Test di Friedman

Il test di Friedman ha lo scopo di determinare se in un insieme di k campioni ($k \geq 2$) almeno due di essi rappresentano popolazioni con mediane differenti. Nel nostro caso i campioni saranno i risultati delle varianti dei nostri due algoritmi: il BB-BC e il GA. Ne segue che con questo test si stima l'eventuale presenza ed entità di una differenza di comportamento tra le varianti di una stessa tipologia di algoritmo.

Si definisce ipotesi nulla per il test di Friedman l'uguaglianza tra le mediane di varie popolazioni (3.1).

$$H_0 : \theta_1 = \theta_2 = \cdots = \theta_k \tag{3.1}$$

In caso contrario si parla di ipotesi alternativa, che si scrive H_1: Not H_0.

Per quanto riguarda la realizzazione vera e propria del test si comincia ordinando i risultati osservati per l'algoritmo (r_j per l'algoritmo j con k algoritmi) ad ogni run, assegnando alla run migliore il rank 1, e alla peggiore il rank k. Accettando l'ipotesi nulla la statistica di Friedman è distribuita in accordo al χ^2 con $k-1$ gradi di libertà, posto $R_j = \frac{1}{N}\sum_i r_i^j$ e N il numero di run. Per rigettare l'ipotesi nulla, il valore ottenuto χ_F^2

$$\chi_F^2 = \frac{12N}{k(k+1)}\left[\sum_j R_j^2 - \frac{k(k+1)^2}{4}\right] \tag{3.2}$$

deve essere maggiore o uguale del valore critico tabulato di χ^2 al livello di significatività α predefinito.

Per implementare il test di Friedman è stata usata la libreria Python messa a disposizione dalla piattaforma STAC (**Staistical Tests for Algorithms Comparison**) [11], una recente piattaforma per l'analisi statistica e la verifica dei risultati ottenuti da algoritmi di computational intelligence.

3.3.2 Test di Holm

In un'analisi statistica, più ipotesi vengono controllate, maggiore è la probabilità di ottenere dei *falsi positivi*. Il test di Holm è un modo per risolvere questo problema. Si tratta di una procedura di comparazione multipla che utilizza un algoritmo di controllo, generalmente quello che ha ottenuto il rank minore nel test di Friedman, confrontandolo con i restanti algoritmi. In termini di p-value, si ordinano i p-value dei vari algoritmi in modo che $p_1 \leq p_2 \leq \cdots \leq p_{k-1}$, si compara ogni p_i con $\alpha/(k-i)$ partendo dal p relativo all'algoritmo che ha ottenuto il risultato migliore nel test di Friedman. Se p_1 è minore di $\alpha/(k-1)$, la corrispondente ipotesi è rifiutata. Il processo continua per tutti i p-value fino a quando l'ipotesi non può essere scartata.

La statistica per confrontare l'algoritmo i-esimo con quello j-esimo è:

$$z = \frac{(R_i - R_j)}{\sqrt{\frac{k(k+1)}{6N}}} \tag{3.3}$$

dove R_i e R_j sono le somme dei rank degli algoritmi i e j rispettivamente e N è il numero di algoritmi.

Il valore di z ottenuto è usato per trovare la probabilità corrispondente ai valori tabulati della distribuzione normale che è confrontato con un valore di significatività α predefinito.

3.3.3 Wilcoxon matched-pairs signed-ranks test

Il test di Wilcoxon ha l'obiettivo di verificare se due campioni rappresentino due diverse popolazioni, dando quindi indicazione sulla differenza di performance tra due algoritmi.

Prima di applicare il test bisogna specificarne la variante, che dipende dall'ipotesi che si vuole verificare. Esistono due varianti: il *two-sided* e il *one-sided*.

Per il two-sided l'ipotesi nulla è che la mediana delle differenze tra i risultati dei due algoritmi θ_D sia nulla ($H_0 : \theta_D = 0$), contro l'ipotesi alternativa che sia diversa da zero ($H_1 : \theta_D \neq 0$).

Il one-sided ha come ipotesi nulla che la mediana delle differenze sia positiva ($H_0 : \theta_D > 0$) mentre l'alternativa è che sia negativa ($H_1 : \theta_D < 0$) o viceversa.

Lo svolgimento vero e proprio del test avviene nel seguente modo: sia d_i la differenza tra i risultati i-esimi, su un totale di N, di due algoritmi. Tutte queste differenze vengono ordinate ed enumerate in base al loro valore assoluto in ordine crescente. Nel caso in cui si abbiano dei pareggi tra di esse si possono adottare diverse scelte: la più popolare è quella di assegnare loro la media aritmetica come rango. Sia R^+ la somma dei rank per cui il risultato del secondo algoritmo è maggiore di quello del primo e sia R^- il contrario. La relazione che lega R^+ e R^- è:

$$\sum R^+ + \sum R^- = \frac{N(N+1)}{2} \tag{3.4}$$

Il valore assoluto del più piccolo tra i valori di R^+ e R^- è chiamato **Wilcoxon T test statistic**. Se i campioni rappresentano la stessa popolazione e quindi la mediana delle differenze è nulla, equivale a dire che R^+ e R^- coincidano.

$$\sum R^+ = \sum R^- = \frac{N(N+1)}{4} \tag{3.5}$$

Il valore appena espresso viene usualmente chiamato **valore atteso del Wilcoxon T statistic**

Per vedere se c'è alta probabilità che il primo campione rappresenti una popolazione con risultati più alti della seconda bisogna verificare se e quanto R^+ sia più grande di R^-. Viceversa, se R^- è significativamente più grande di R^+ ci sarà un'elevata probabilità che il secondo campione rappresenti una popolazione con valori più alti della prima. Il primo caso corrisponde all'ipotesi alternativa direzionale $H_1 : \theta_D > 0$, il secondo all'altra $H_1 : \theta_D < 0$. Per poter giungere alla eventuale conclusione che la differenza tra i due campioni sia significativa e non frutto del caso, il valore di T ottenuto deve essere minore o uguale del valore critico di T tabulato ad un livello di significatività α predefinito (Table B.12 in [20]).

Quando la dimensione del campione è abbastanza grande (non esiste una chiara indicazione al riguardo), il Wilcoxon T statistic è ben approssimato da una distribuzione gaussiana (3.6):

$$z = \frac{T - \frac{n(n+1)}{4}}{\sqrt{\frac{n(n+1)(2n+1)}{24}}} \tag{3.6}$$

z rappresenta l'approssimazione normale per il Wilcoxon T. Nella (3.6) T indica il valore calcolato del T di Wilcoxon, n il numero dei rank per cui la differenza è non nulla, il termine $n(n+1)/4$ è il valore atteso di T e il denominatore è la deviazione standard attesa della distribuzione dei campioni del T statistico.

Quando si usa un'ipotesi alternativa non direzionale, l'ipotesi nulla può essere scartata se il valore assoluto di z è maggiore o uguale al valore critico tabulato secondo il livello di significanza α scelto. Quando invece si impiega un'ipotesi alternativa direzionale, essa sarà corroborata se il valore assoluto di z è maggiore o uguale (o minore o uguale a seconda dell'ipotesi) al valore critico tabulato. Ciò dipende da quale tra R^+ e R^- sia maggiore.

In termini di p-value si dirà che un p-value più basso del valore di significatività α indica che è necessario rifiutare l'ipotesi di uguaglianza delle popolazioni e accettare l'ipotesi alternativa.

Per implementare il test di Wilcoxon si è utilizzato il pacchetto *stats* della libreria open-source SciPy.

Capitolo 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 in questo elaborato sono [14]:

1. La funzione della sfera (Figura 4.1), la cui espressione è:

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

   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, nel dominio n-dimensionale di lato $x_i \in [-5.12, 5.12]$

2. La funzione di Ackley (Figura 4.2), avente la forma:

$$f(x) = -a \exp\left(-b\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(cx_i)\right) + a + \exp(1) \tag{4.2}$$

   Con $a = 20$, $b = 0.2$, $c = 2\pi$. La funzione è valutata nell'ipercubo di lato $x_i \in [-32.768, 32.768]$ ed assume valore $f(x^*) = 0$ nel punto di minimo $x^* = (0, ..., 0)$. Come si può notare, la funzione ha numerosi minimi locali che stressano in particolar modo l'algoritmo.

3. La funzione di Rastrigin (Figura 4.3):

$$f(x) = 10n + \sum_{i=1}^{n}[x_i^2 - 10\cos(2\pi x_i)] \tag{4.3}$$

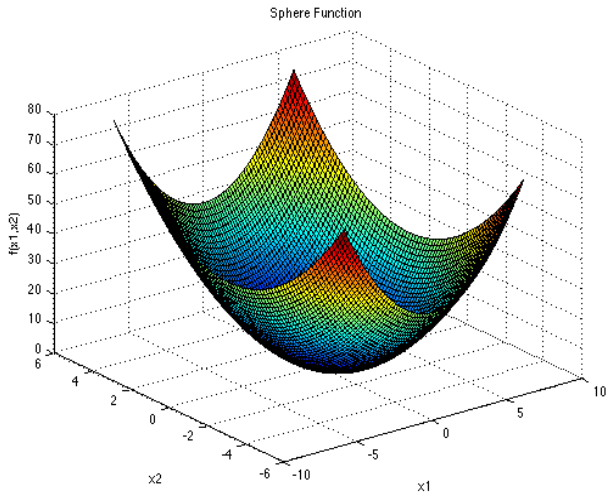


Figura 4.1: Funzione della sfera

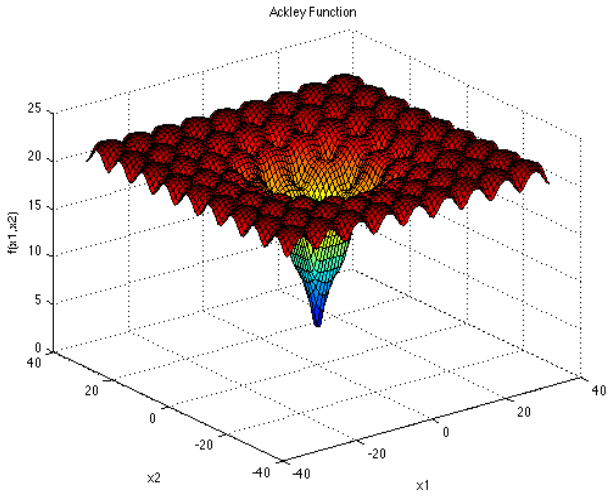


Figura 4.2: Funzione di Ackley

Valutata nell'ipercubo di lato $x_i \in [-5.12, 5.12]$. La funzione assume valore $f(x^*) = 0$ per il punto di minimo globale $x^* = (0, ..., 0)$. Si tratta di una funzione fortemente multimodale per cui i minimi sono distribuiti regolarmente.

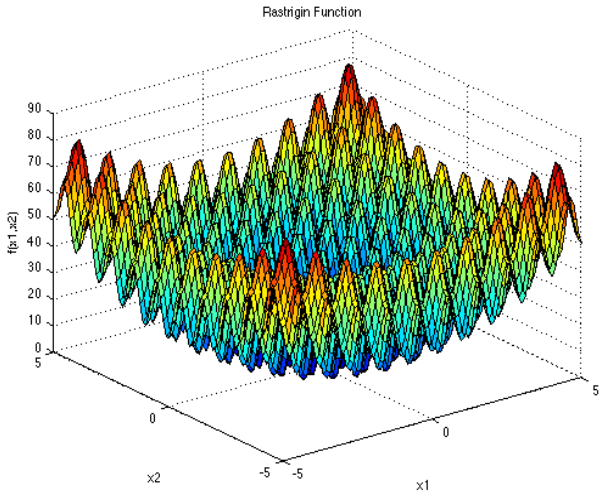


Figura 4.3: Funzione di Rastrigin

4. La funzione di Rosenbrock (Figura 4.4):

$$f(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2] \tag{4.4}$$

Valutata nell'ipercubo di lato $x_i \in [-5, 10]$, in corrispondenza del minimo globale $x^* = (1, ..., 1)$ 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.

5. La funzione di Levy (Figura 4.5):

$$f(x) = \sin^2(\pi w_1) + \sum_{i=1}^{n-1} (w_i - 1)^2 [1 + 10 \sin^2(\pi w_i + 1)] + (w_n - 1)^2 [1 + \sin^2(2\pi w_n)] \tag{4.5}$$

Valutata nell'ipercubo $x_i \in [-10, 10]$ e con $w_i = 1 + \frac{x_i - 1}{4}$. La funzione assume il valore $f(x^*) = 0$ in corrispondenza del punto di minimo $x^* = (1, ..., 1)$

4.2 Esperimenti

La sperimentazione svolta in questa tesi rappresenta uno studio delle prestazioni dell'Algoritmo Big Bang-Big Crunch implementato, le cui caratteristiche

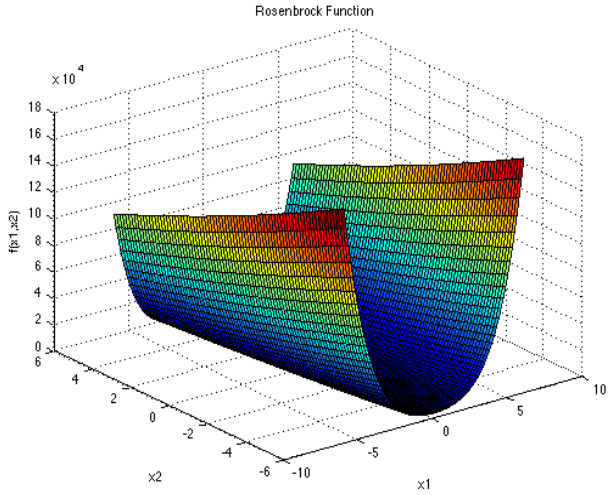


Figura 4.4: Funzione di Rosenbrock

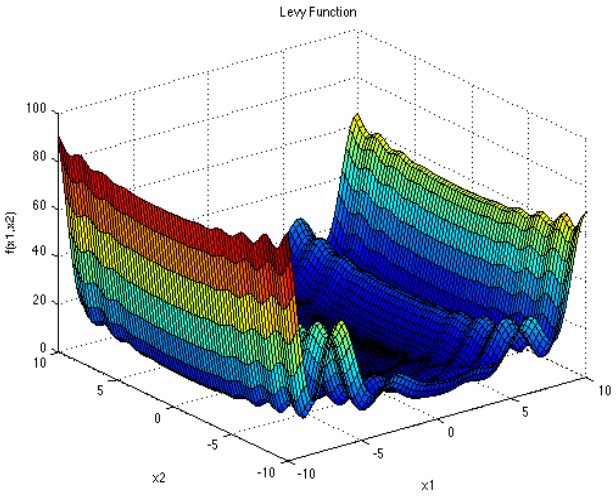


Figura 4.5: Funzione di Levy

principali sono state discusse nel Capitolo II. Queste prestazioni sono state confrontate con quelle di un Algoritmo Genetico, la cui struttura è stata descritta nel Capitolo I.

Prima di arrivare al confronto tra il BB-BC e il GA si sono eseguiti diversi esperimenti volti a determinare quali fossero i migliori iperparametri per l'uno e l'altro algoritmo. In totale si sono realizzate 6 varianti del BB-BC e 4 varianti del GA e per ognuna di essa si sono seguite le istruzioni indicate per la competizione *CEC"2005 Special Session* [13]:

- ogni algoritmo è eseguito per 25 volte per ogni funzione di test; quindi un campione di dati è composto da 25 valutazioni di fitness;
- la dimensione dell'individuo è fissata a $D = 10$;
- ogni algoritmo esegue 100000 valutazioni della funzione di fitness, quindi ogni run si ferma al raggiungimento delle 100000 valutazioni.

In aggiunta a questi esperimenti, se ne sono svolti altri in maniera analoga con la dimensionalità D aumentata a $D = 20$.

L'Algoritmo Genetico genera in maniera random una popolazione di $n = 50$ individui, 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 uno a "Torneo" (Tournament) che per $k = 50$ volte prende casualmente un campione di 3 individui e ne seleziona quello con la fitness più alta.

L'operatore di crossover è un *Two-Points Crossover*.Il processo di crossover riguarda solo una parte degli individui, in base alla *probabilità di crossover* denominata CXPB. I valori di CXPB provati sono CXPB=0.5,0.7.

Dopo di ciò ogni individuo viene "modificato" attraverso l'operatore di mutazione gaussiana di media $\mu = 0$ e deviazione standard $\sigma = 1$, con probabilità indipendente di mutazione per ogni attributo pari a MUTPB. I valori di MUTPB sperimentati sono MUTPB=0.2,0.3

Anche nell'Algoritmo BB-BC viene generata casualmente una popolazione di $n = 50$ individui, valutati immediatamente con l'operatore di valutazione che funziona in maniera analoga al GA. La struttura dell'algoritmo implementato è quella mostrata nel Capitolo II. Le nuove popolazioni vengono generate con la formula generalizzata (2.4), dove β può assumere i valori riportati nella Tabella 4.1. Per il valore random r si è scelta l'estrazione da una gaussiana di media $\mu = 0$ e deviazione standard $\sigma = 1$

4.3 Determinazione dei migliori iperparametri

Per effettuare la scelta dei migliori iperparametri dei due algoritmi per ogni funzione di benchmark si sono sfruttati i test di Friedman e di Holm. Il primo ci ha permesso di determinare l'algoritmo di controllo necessario all'esecuzione del secondo.

Algoritmo	Iperparametro	Istanza
GA	CXPB	0.5, 0.7
	MUTB	0.2, 0.3
BB-BC	β	0, 0.2, 0.3, 0.5, 0.7. 0.8, 1

Tabella 4.1: Iperparametri dei due algoritmi implementati

Configurazione	Rank				
	Ackley	Levy	Sfera	Rastrigin	Rosenbrock
BB_BC_0	7.0	7.0	7.0	7.0	7.0
BB_BC_0.2	3.72	3.84	3.52	3.6	3.56
BB_BC_0.3	3.6	3.56	3.16	3.72	3.16
BB_BC_0.5	3.0	3.88	3.76	3.56	4.24
BB_BC_0.7	3.2	3.04	3.8	3.52	3.0
BB_BC_0.8	3.36	3.36	3.48	3.24	4.0
BB_BC_1	4.12	3.32	3.28	3.36	3.04

Tabella 4.2: Rank dal Test di Friedman per l'Algoritmo BB-BC

In questi test si è posto come livello di significatività il valore $\alpha = 0.05$. Nelle Tabelle 4.2, 4.3 sono illustrati rispettivamente i rank ed i p-value ottenuti dai test di Friedman per l'Algoritmo BB-BC, mentre nelle Tabelle 4.4, 4.5 vi sono i rank e i p-value per l'Algoritmo Genetico.

Le denominazioni BB_BC_β e gen_CXPB_MUTB indicano rispettivamente le configurazioni dell'Algoritmo BB-BC con $\beta = 0, 0.2, 0.3, 0.5, 0.7, 0.8, 1$ e dell'Algoritmo Genetico con CXPB=0.5,0.7 e MUTPB=0.2,0.3.

I risultati dei test di Friedman mostrano che si può rifiutare l'ipotesi nulla del test, e che quindi ci sono differenze significative tra le istanza valutate, sia per l'Algoritmo BB-BC che per l'Algoritmo Genetico, ad eccezione di quest'ultimo applicato alla funzione di Rosenbrock. Infatti, tutti i p-value sono inferiori al livello di significatività $\alpha = 0.05$. Ciò vale soprattutto per il BB-BC, per il quale i p-value sono di almeno 10 ordini di grandezza inferiori ad α.

Le configurazioni che hanno ottenuto il rank più basso nei test di Friedman sono riportate nella Tabella 4.6 e sono state scelte come algoritmi di controllo nei test di Holm i cui risultati sono illustrati nelle Tabelle 4.7, 4.8.

Funzione	p-value
Ackley	$3.14 \cdot 10^{-14}$
Levy	$0.97 \cdot 10^{-13}$
Sfera	$2.22 \cdot 10^{-13}$
Rastrigin	$4.43 \cdot 10^{-13}$
Rosenbrock	$2.78 \cdot 10^{-15}$

Tabella 4.3: p-value dal Test di Friedman per l'Algoritmo BB-BC

Configurazione	Rank				
	Ackley	Levy	Sfera	Rastrigin	Rosenbrock
gen_0.5_0.2	2.92	3.04	3.12	2.84	2.68
gen_0.5_0.3	1.92	2.16	2.2	2.04	2.12
gen_0.7_0.2	3.24	2.84	2.92	3.36	2.64
gen_0.7_0.3	1.92	1.96	1.76	1.76	2.56

Tabella 4.4: Rank dal Test di Friedman per l'Algoritmo Genetico

Funzione	p-value
Ackley	$2.82 \cdot 10^{-5}$
Levy	$4.88 \cdot 10^{-3}$
Sfera	$1.81 \cdot 10^{-4}$
Rastrigin	$3.20 \cdot 10^{-6}$
Rosenbrock	$4.00 \cdot 10^{-1}$

Tabella 4.5: p-value dal Test di Friedman per l'Algoritmo BB-BC

	BB-BC	GA
Ackley	BB_BC_0.5	gen_0.7_0.3
Levy	BB_BC_0.7	gen_0.7_0.3
Sfera	BB_BC_0.3	gen_0.7_0.3
Rastrigin	BB_BC_0.8	gen_0.7_0.3
Rosenbrock	BB_BC_0.7	gen_0.5_0.3

Tabella 4.6: Algoritmi di controllo nei Test di Holm

Configurazioni BB-BC	p-value	$\alpha/(k-i); \alpha = 0.05$
Ackley		
BB_BC_0	$3.53 \cdot 10^{-10}$	0.008
BB_BC_1	$3.34 \cdot 10^{-1}$	0.01
BB_BC_0.2	0.95	0.0125
BB_BC_0.3	0.97	0.0166
BB_BC_0.8	1	0.025
BB_BC_0.7	1	0.05
Levy		
BB_BC_0	$5.46 \cdot 10^{-10}$	0.008
BB_BC_0.5	0.85	0.01
BB_BC_0.2	0.85	0.0125
BB_BC_0.3	1	0.0166
BB_BC_0.8	1	0.025
BB_BC_1	1	0.05
Sfera		
BB_BC_0	$1.97 \cdot 10^{-9}$	0.008
BB_BC_0.7	1	0.01
BB_BC_0.5	1	0.0125
BB_BC_0.2	1	0.0166
BB_BC_0.8	1	0.025
BB_BC_1	1	0.05
Rastrigin		
BB_BC_0	$4.54 \cdot 10^{-9}$	0.008
BB_BC_0.3	1	0.01
BB_BC_0.2	1	0.0125
BB_BC_0.5	1	0.0166
BB_BC_0.7	1	0.025
BB_BC_1	1	0.05
Rosenbrock		
BB_BC_0	$3.53 \cdot 10^{-10}$	0.008
BB_BC_0.5	0.21	0.01
BB_BC_0.8	0.41	0.0125
BB_BC_0.2	1	0.0166
BB_BC_0.3	1	0.025
BB_BC_1	1	0.05

Tabella 4.7: Risultati del Test di Holm tra le configurazioni dell'Algoritmo BB-BC

Configurazioni GA	p-value	$\alpha/(k-i); \alpha = 0.05$
Ackley		
gen_0.7_0.2	$0.90 \cdot 10^{-3}$	0.016
gen_0.5_0.2	0.012	0.025
gen_0.5_0.3	1	0.05
Levy		
gen_0.5_0.2	$0.92 \cdot 10^{-2}$	0.016
gen_0.7_0.2	0.031	0.025
gen_0.5_0.3	0.58	0.05
Sfera		
gen_0.5_0.2	$0.58 \cdot 10^{-3}$	0.016
gen_0.7_0.2	0.0029	0.025
gen_0.5_0.3	0.23	0.05
Rastrigin		
gen_0.7_0.2	$3.53 \cdot 10^{-5}$	0.016
gen_0.5_0.2	0.0062	0.025
gen_0.5_0.3	0.44	0.05
Rosenbrock		
gen_0.5_0.2	0.38	0.016
gen_0.7_0.2	0.38	0.025
gen_0.7_0.3	0.38	0.05

Tabella 4.8: Risultati del Test di Holm tra le configurazioni dell'Algoritmo Genetico

	BB-BC	GA
Ackley	BB_BC_0.3	gen_0.7_0.3
Levy	BB_BC_0.3	gen_0.7_0.3
Sfera	BB_BC_0.5	gen_0.5_0.3
Rastrigin	BB_BC_0.8	gen_0.7_0.3
Rosenbrock	BB_BC_1	gen_0.7_0.3

Tabella 4.9: Algoritmi di controllo nei Test di Holm (D=20)

Come si può notare, i test di holm mostrano che per il BB-BC le varianti non portano a risultati molto diversi ad eccezione della variante con $\beta = 0$. Per il GA invece, vi sono maggiori differenze tra le configurazioni, tranne tra le seguenti: per la funzione di Ackley la *gen_0.7_0.3* e la *gen_0.5_0.3*; per la funzione di Levy la *gen_0.7_0.3*, *gen_0.7_0.2* e la *gen_0.5_0.3*; per la funzione della Sfera la *gen_0.7_0.3* e la *gen_0.5_0.3*; per la funzione di Rastrigin la *gen_0.7_0.3* e la *gen_0.5_0.3*; infine, per la funzione di Rosenbrock le configurazioni sono tutte equivalenti.

Ad ogni modo si possono considerare le configurazioni di controllo utilizzate nei test di Holm come quelle migliori. Queste ultime quindi, sono utilizzate nei test di Wilcoxon.

Per quanto riguarda gli esperimenti effettuati con una dimensionalità pari a $D = 20$, si riportano i risultati ottenuti nei Test di Holm nelle Tabelle 4.10, 4.11.

4.4 Risultati

Le migliori varianti degli algoritmi BB-BC e GA sono state messe a confronto facendo uso del Test di Wilcoxon. Quest'ultimo è stato impostato con un'ipotesi alternativa direzionale (Tabelle 4.12, 4.13) di tipo $H_1 : \theta < 0$ con un livello di significatività $\alpha = 0.05$. Sia per dimensionalità $D = 10$ che per $D = 20$ si può dire che essendo i p-value minori di α, si può rifiutare l'ipotesi nulla a favore dell'ipotesi alternativa, cioè dell'ipotesi per cui è maggiormente probabile che l'Algoritmo BB-BC generi risultati più piccoli di quelli dell'Algoritmo Genetico.

In conclusione, si può quindi affermare che:

- l'Algoritmo BB-BC è migliore dell'Algoritmo Genetico per tutte le funzioni e per entrambe le dimensionalità provate;
- Le varianti dell'Algoritmo BB-BC statisticamente conducono a risultati equivalenti, ad esclusione della variante $\beta = 0$;
- Tra i risultati delle varianti dell'Algoritmo Genetico ci sono differenze non trascurabili che variano da funzione a funzione, ma, dai risultati osservati, si nota che sono più efficaci le varianti con coefficiente MUTPB=0.3.

Configurazioni BB-BC	p-value	$\alpha/(k-i); \alpha = 0.05$
Ackley		
BB_BC_0	$8.42 \cdot 10^{-10}$	0.008
BB_BC_1	0.95	0.01
BB_BC_0.8	1	0.0125
BB_BC_0.2	1	0.0166
BB_BC_0.7	1	0.025
BB_BC_0.5	1	0.05
Levy		
BB_BC_0	$1.29 \cdot 10^{-9}$	0.008
BB_BC_0.2	0.58	0.01
BB_BC_0.7	1	0.0125
BB_BC_0.5	1	0.0166
BB_BC_0.8	1	0.025
BB_BC_1	1	0.05
Sfera		
BB_BC_0	$5.46 \cdot 10^{-10}$	0.008
BB_BC_0.8	0.33	0.01
BB_BC_0.7	0.68	0.0125
BB_BC_1	1	0.0166
BB_BC_0.3	1	0.025
BB_BC_0.2	1	0.05
Rastrigin		
BB_BC_0	$5.92 \cdot 10^{-11}$	0.008
BB_BC_0.2	0.21	0.01
BB_BC_0.5	0.53	0.0125
BB_BC_0.7	0.57	0.0166
BB_BC_0.3	0.72	0.025
BB_BC_1	0.72	0.05
Rosenbrock		
BB_BC_0	$1.46 \cdot 10^{-10}$	0.008
BB_BC_0.5	0.58	0.01
BB_BC_0.7	0.60	0.0125
BB_BC_0.2	0.64	0.0166
BB_BC_0.3	0.64	0.025
BB_BC_0.8	0.69	0.05

Tabella 4.10: Risultati del Test di Holm tra le configurazioni dell'Algoritmo BB-BC (D=20)

Configurazioni GA	p-value	$\alpha/(k-i); \alpha = 0.05$
Ackley		
gen_0.5_0.2	$3.53 \cdot 10^{-5}$	0.016
gen_0.7_0.2	0.0043	0.025
gen_0.5_0.3	0.51	0.05
Levy		
gen_0.5_0.2	0.026	0.016
gen_0.7_0.2	0.097	0.025
gen_0.5_0.3	1	0.05
Sfera		
gen_0.5_0.2	$5.81 \cdot 10^{-5}$	0.016
gen_0.7_0.2	0.0004	0.025
gen_0.7_0.3	0.15	0.05
Rastrigin		
gen_0.7_0.2	$3.53 \cdot 10^{-5}$	0.016
gen_0.5_0.2	0.0062	0.025
gen_0.5_0.3	0.44	0.05
Rosenbrock		
gen_0.7_0.2	0.15	0.016
gen_0.5_0.3	0.20	0.025
gen_0.5_0.2	0.32	0.05

Tabella 4.11: Risultati del Test di Holm tra le configurazioni dell'Algoritmo Genetico (D=20)

	BBBC vs GA
Ackley	$3.18 \cdot 10^{-5}$
Levy	$2.98 \cdot 10^{-8}$
Sfera	$2.98 \cdot 10^{-8}$
Rastrigin	$2.98 \cdot 10^{-8}$
Rosenbrock	$2.98 \cdot 10^{-8}$

Tabella 4.12: p-value per il Test di Wilcoxon ($H_0 : \theta < 0$)

Nelle Figure 4.6, 4.7, 4.9, 4.11, 4.13, 4.15, 4.17, 4.19 sono rappresentati i confronti, su ogni funzione di benchmark, tra i valori di fitness dei due algoritmi per le dimensioni $D = 10$ e $D = 20$.

Nelle Figure 4.8, 4.10, 4.12, 4.14, 4.16, 4.18 vengono illustrati, su ogni funzione di benchmark ad eccezione di quella di Ackley (funzione per cui già nella Figura 4.6 è visibile il range entro cui si trovano i risultati dell'algoritmo più efficiente), i valori di fitness dell'algoritmo che ha ottenuto i risultati migliori (ancora per le dimensioni $D = 10$ e $D = 20$).

Il fatto che l'Algoritmo BB-BC fornisca risultati migliori dell'Algoritmo Genetico può spiegarsi notando che il primo, coinvolgendo tutta la popolazione nel calcolo del centro di massa, riesce ad ottimizzare meglio le soluzioni candidate. Ci si può anche rendere conto dei risultati molto meno efficienti della variante $\beta = 0$ osservando che quest'ultima ha una capacità di variare le soluzioni candidate molto più bassa delle altre varianti.

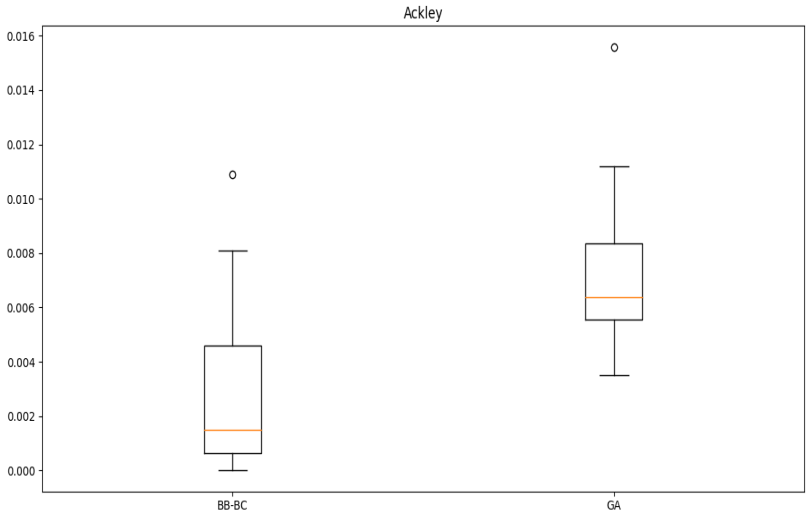


Figura 4.6: Valori di fitness per i due algoritmi sulla funzione di Ackley

	BBBC vs GA
Ackley	$2.98 \cdot 10^{-8}$
Levy	$2.98 \cdot 10^{-8}$
Sfera	$2.98 \cdot 10^{-8}$
Rastrigin	$2.98 \cdot 10^{-8}$
Rosenbrock	$2.98 \cdot 10^{-8}$

Tabella 4.13: p-value per il Test di Wilcoxon ($H_0 : \theta < 0$) (D=20)

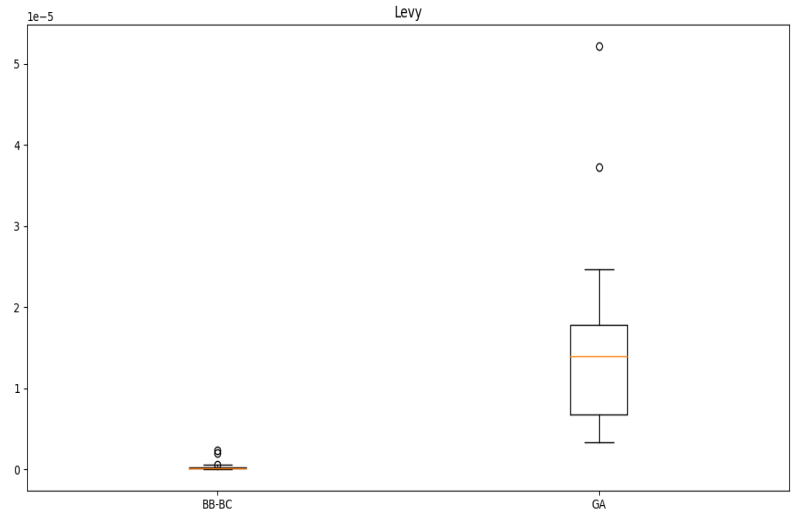


Figura 4.7: Valori di fitness per i due algoritmi sulla funzione di Levy

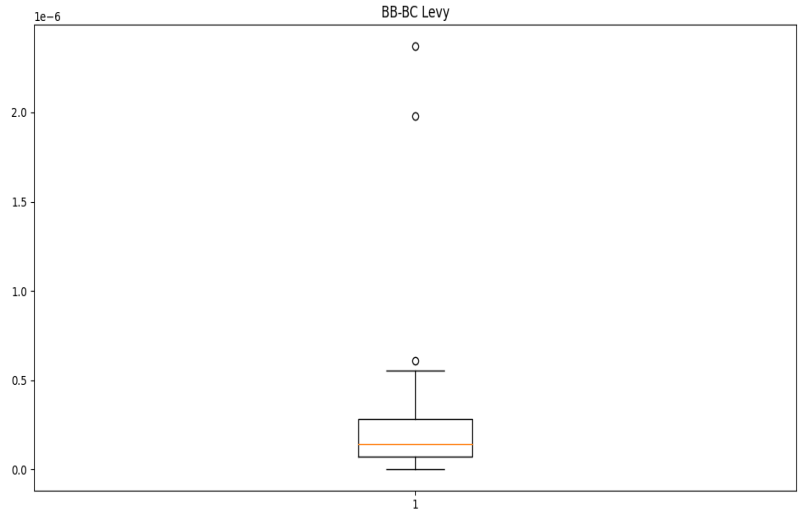


Figura 4.8: Valori di fitness per il BB-BC sulla funzione di Levy

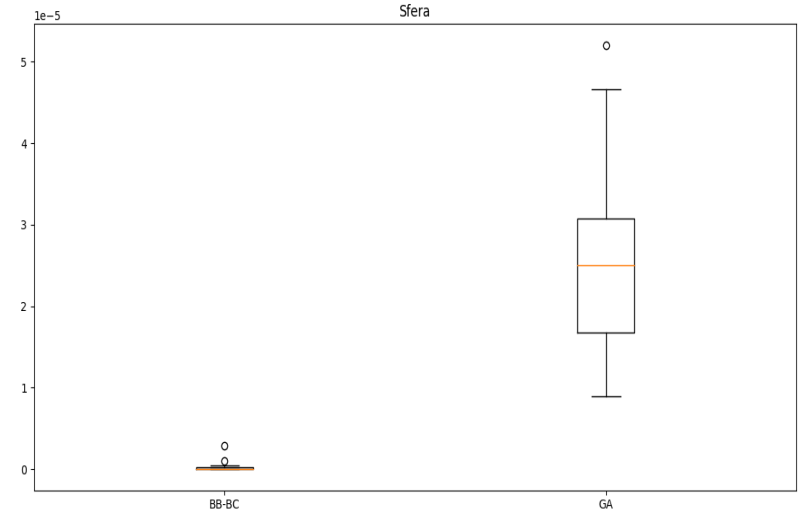


Figura 4.9: Valori di fitness per i due algoritmi sulla funzione della sfera

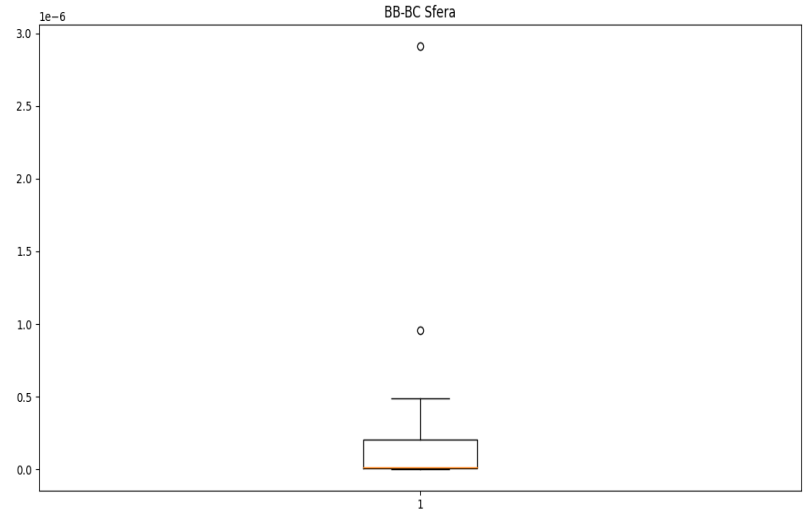


Figura 4.10: Valori di fitness per il BB-BC sulla funzione della sfera

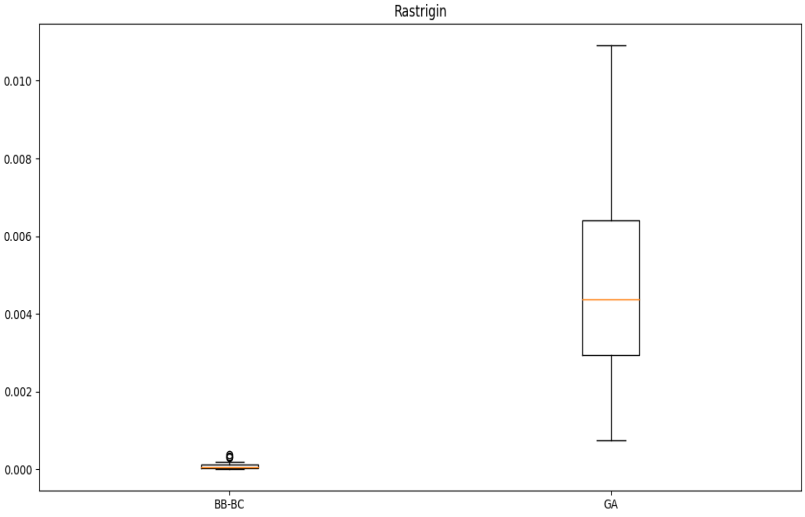


Figura 4.11: Valori di fitness per i due algoritmi sulla funzione di Rastrigin

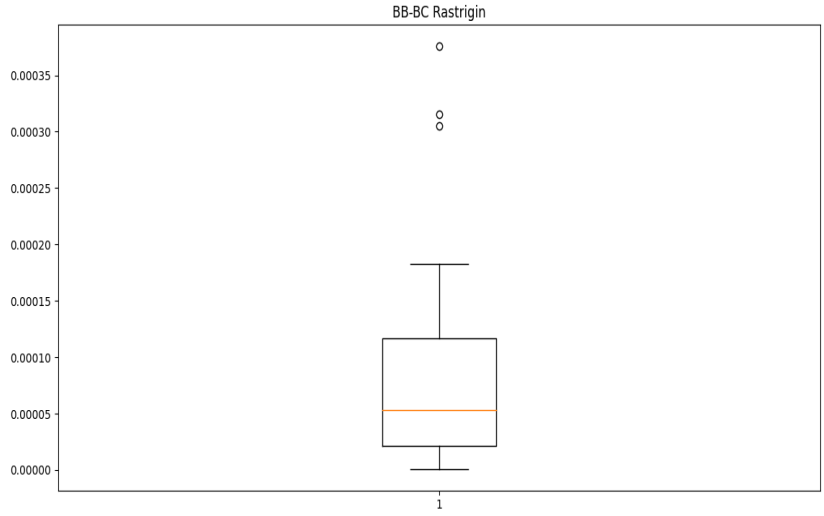


Figura 4.12: Valori di fitness per il BB-BC sulla funzione di Rastrigin

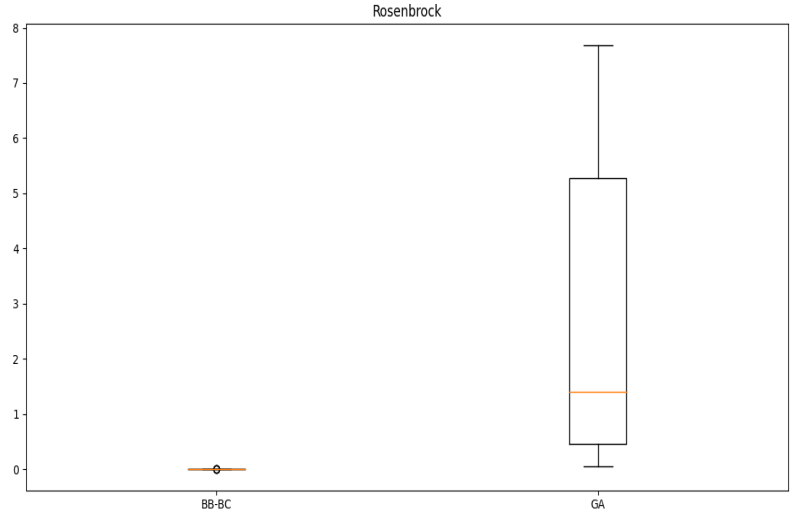


Figura 4.13: Valori di fitness per i due algoritmi sulla funzione di Rosenbrock

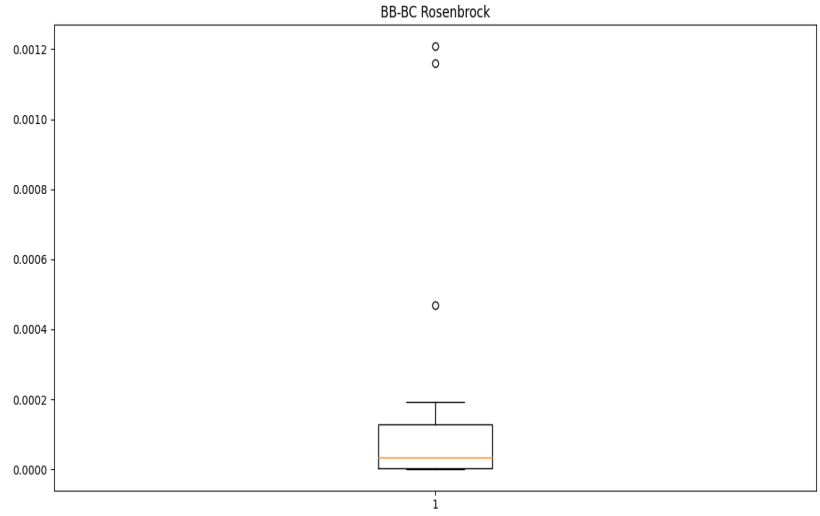


Figura 4.14: Valori di fitness per il BB-BC sulla funzione di Rosenbrock

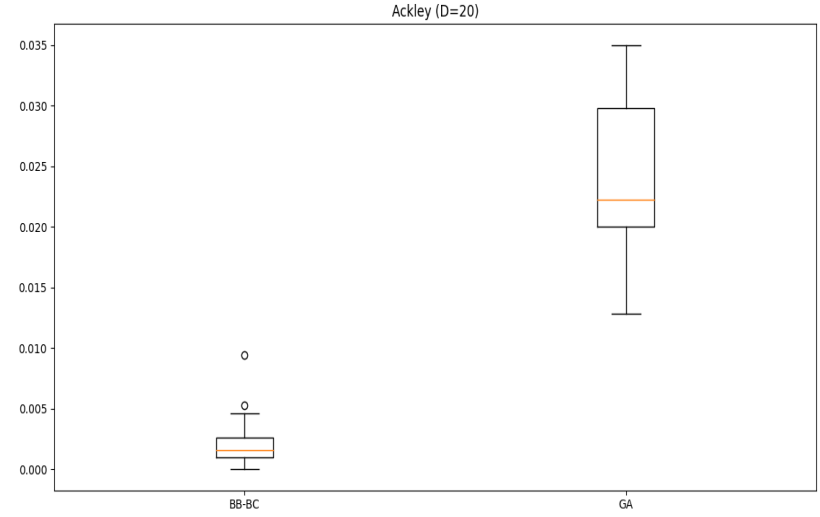


Figura 4.15: Valori di fitness per i due algoritmi sulla funzione di Ackley (D=20)

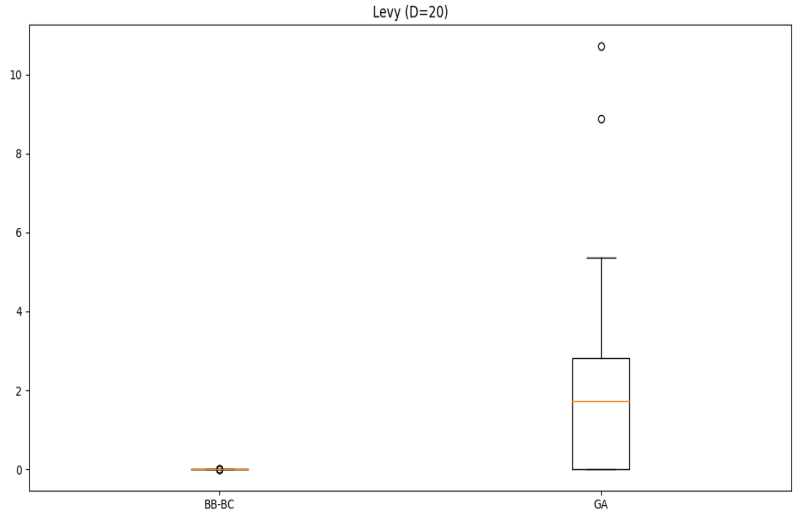


Figura 4.16: Valori di fitness per i due algoritmi sulla funzione di Levy (D=20)

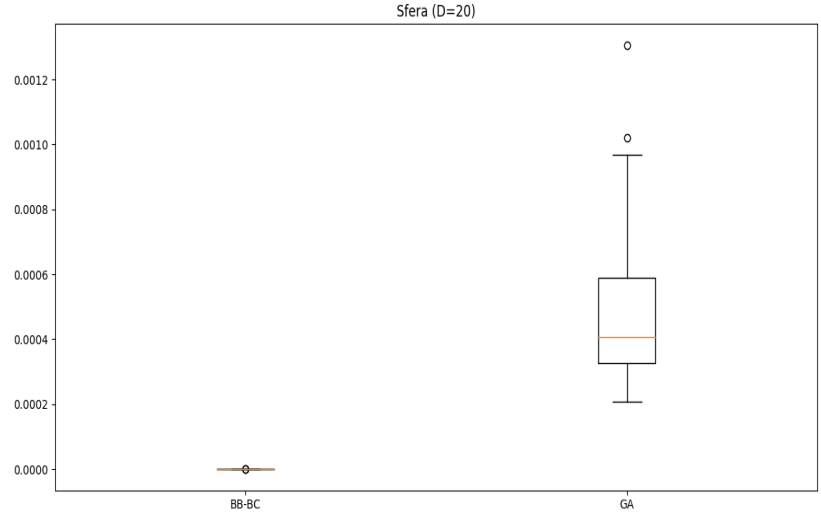


Figura 4.17: Valori di fitness per i due algoritmi sulla funzione della sfera (D=20)

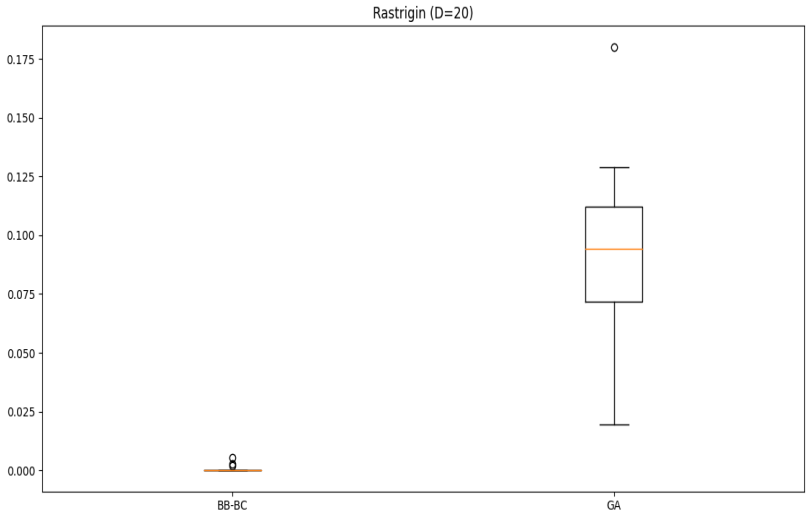


Figura 4.18: Valori di fitness per i due algoritmi sulla funzione di Rastrigin (D=20)

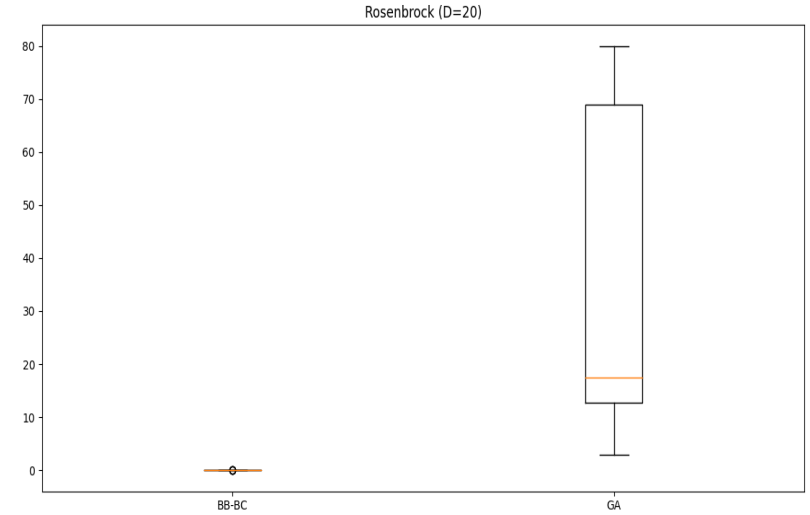


Figura 4.19: Valori di fitness per i due algoritmi sulla funzione di Rosenbrock (D=20)

Conclusioni

Oggetto di studio di questo elaborato di tesi è stato l'Algoritmo Big Bang-Big Crunch, che è stato prima implementato e poi confrontato con un'altra metaeuristica: l'Algoritmo Genetico.

Gli algoritmi sono stati sviluppati in Python, con il supporto fondamentale del framework DEAP.

Dopo l'implementazione dei due algoritmi, ci si è occupati di configurarne gli iperparametri: si è valutata, per ogni funzione di benchmark, ciascuna variante confrontandola con le altre. Questo lo si è potuto fare mediante l'utilizzo dei test statistici di Friedman e Holm.

Una volta determinate le migliori varianti dei due algoritmi per ciascuna funzione, si è potuto proseguire al confronto tra l'Algoritmo BB-BC e l'Algoritmo Genetico sfruttando il test di Wilcoxon. Da questa analisi risulta che l'Algoritmo BB-BC ottiene migliori risultati rispetto all'Algoritmo Genetico per tutte le funzioni di benchmark provate. Questi risultati, come detto in precedenza, possono essere giustificati dal fatto che l'Algoritmo BB-BC, coinvolgendo tutta la popolazione nel calcolo del centro di massa, riesce ad ottimizzare meglio le soluzioni candidate.

Ricerche future potrebbero continuare la sperimentazione sviluppata in questo elaborato utilizzando altre funzioni di benchmark o effettuando una parallelizzazione degli algoritmi, potendo quindi sfruttare i paradigmi di distribuzione implementati da DEAP non utilizzati in questa tesi.

L'elaborato di tesi svolto mostra le potenzialità di un algoritmo, il BB-BC, che trova sempre maggiore impiego nella risoluzione di diversi problemi in ambito scientifico e industriale. Si vogliono ricordare le applicazioni nel settore dell'ingegneria civile dove l'Algoritmo BB-BC può essere utilizzato per localizzare le zone di fragilità di una struttura, oppure il suo utilizzo nell'ottimizzazione dei costi economici e ambientali di un edificio o impianto industriale, ma anche nell'ambito dell'ingegneria informatica l'Algoritmo BB-BC trova applicazioni, come ad esempio nel software testing.

Bibliografia

[1] Anupam Biswas, KK Mishra, Shailesh Tiwari, and AK Misra. Physics-inspired optimization algorithms: a survey. *Journal of Optimization*, 2013, 2013.

[2] Marco Bramanti, Carlo Domenico Pagani, and Sandro Salsa. *Analisi matematica 2*. Zanichelli, 2008.

[3] Joaquín Derrac, Salvador García, Daniel Molina, and Francisco Herrera. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18, 2011.

[4] Agoston E Eiben, James E Smith, et al. *Introduction to evolutionary computing*, volume 53. Springer, 2003.

[5] Osman K Erol and Ibrahim Eksin. A new optimization method: big bang–big crunch. *Advances in Engineering Software*, 37(2):106–111, 2006.

[6] Salvador García, Daniel Molina, Manuel Lozano, and Francisco Herrera. A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the cec'2005 special session on real parameter optimization. *Journal of Heuristics*, 15(6):617–644, 2009.

[7] Zakaria Laboudi and Salim Chikhi. Comparison of genetic algorithm and quantum genetic algorithm. *Int. Arab J. Inf. Technol.*, 9(3):243–249, 2012.

[8] Andrea Malossini, Enrico Blanzieri, and Tommaso Calarco. Qga: a quantum genetic algorithm. 2004.

[9] Aleksandar Milajić, Dejan Beljaković, Nebojša Davidović, Nikolai Vatin, and Vera Murgul. Using the big bang–big crunch algorithm for rational design of an energy-plus building. *Procedia Engineering*, 117:911–918, 2015.

[10] Esmat Rashedi, Hossein Nezamabadi-Pour, and Saeid Saryazdi. Gsa: a gravitational search algorithm. *Information sciences*, 179(13):2232–2248, 2009.

[11] Ismael Rodríguez-Fdez, Adrián Canosa, Manuel Mucientes, and Alberto Bugarín. Stac: a web platform for the comparison of algorithms using statistical tests. In *2015 IEEE international conference on fuzzy systems (FUZZ-IEEE)*, pages 1–8. IEEE, 2015.

[12] Surender Singh and Parvin Kumar. Application of big bang big crunch algorithm to software testing. *International Journal of Computer Science and Communication*, 3(1):259–262, 2012.

[13] 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.

[14] S. Surjanovic and D. Bingham. Virtual library of simulation experiments: Test functions and datasets. Retrieved June 17, 2021, from `http://www.sfu.ca/~ssurjano`.

[15] Zahra Tabrizian, Ehsan Afshari, Gholamreza Ghodrati Amiri, Morteza Hossein Ali Beigy, and Seyed Mohammad Pourhoseini Nejad. A new damage detection method: Big bang-big crunch (bb-bc) algorithm. *Shock and Vibration*, 20(4):633–648, 2013.

[16] Peter JM Van Laarhoven and Emile HL Aarts. Simulated annealing. In *Simulated annealing: Theory and applications*, pages 7–15. Springer, 1987.

[17] M Vatandoost, A Ekhlassi, and SA Yazdanfar. Computer-aided architectural design: Classification and application of optimization algorithms.

[18] Huaixiao Wang, Jianyong Liu, Jun Zhi, and Chengqun Fu. The improvement of quantum genetic algorithm and its application on function optimization. *Mathematical problems in engineering*, 2013, 2013.

[19] Engin Yesil and Leon Urbas. Big bang-big crunch learning method for fuzzy cognitive maps. *World Acad. Sci. Eng. Technol*, 71:815–8124, 2010.

[20] J.H. Zar. *Biostatistical Analysis*. Prentice Hall, 2010.