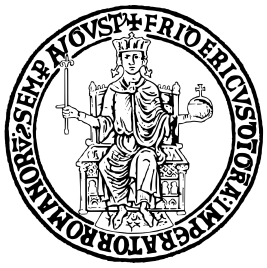
UNIVERSITÀ DEGLI STUDI DI NAPOLI
"FEDERICO II"



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 di Algoritmi Memetici

Relatori:
Dott.ssa Autilia Vitiello

Candidato:
Esposito Marina
Matr. $N85001064$

Anno Accademico 2018/2019

Indice

Introduzione

Tra le strategie più utilizzate nella risoluzione di problemi di ottimizzazione troviamo gli Algoritmi Evolutivi (EA), metaeuristiche *population-based* che si ispirano alla teoria evolutiva proposta da Charles Darwin; queste strategie sono largamente impiegate nella ricerca di soluzioni ottimali per un'ampia classe di problemi grazie alle loro capacità di esplorazione dello spazio di ricerca e alla caratteristica di essere facilmente adattabili a nuovi problemi.

Gli Algoritmi Memetici (MA) possono essere considerati un'estensione dei tradizionali Algoritmi Evolutivi. Ispirati alla nozione di *meme* come unità di informazione culturale proposta da Richard Dawkins, gli Algoritmi Memetici rappresentano una classe di metaeuristiche per l'ottimizzazione che combinano la natura di ricerca globale degli Algoritmi Evolutivi con la Ricerca Locale, con lo scopo di sfruttare il maggior numero di informazioni disponibili per ogni problema a cui sono applicati; dall'equilibrio delle tecniche di *exploration* ed *exploitation* gli Algoritmi Memetici riescono ad evitare la convergenza prematura a punti di ottimo locale (problematica tipica per le tecniche di Local Search), e allo stesso tempo raggiungono buone soluzioni più velocemente rispetto agli Algoritmi Evolutivi, grazie al *Local Improvement* delle soluzioni.

Le tecniche memetiche sono impiegate in un range di applicazioni sempre crescente con risultati apprezzabili, dalla risoluzione di problemi NP classici come il *graph partitioning* e il *Traveling Salesman Problem*, alle applicazioni nel campo scientifico ed industriale, che comprendono le moderne tecnologie di *nano-processing* come il *laser-writing* o la litografia 3D a fascio elettronico, aiutando a definire i parametri geometrici ottimali per la fabbricazione di nanostrutture ottiche.

In questo elaborato è stato implementato e valutato un Algoritmo Memetico, confrontando le sue prestazioni con quelle di altre due metaeuristiche: un Algoritmo Genetico e un algoritmo di Hill Climbing. Tutti gli Algoritmi sono stati implementati in Python, sfruttando principalmente la libreria evolutiva DEAP e arricchendola delle strutture necessarie alla realizzazione delle componenti memetiche.

Gli Algoritmi sono stati eseguiti su tre diverse funzioni di benchmark: la funzione della sfera, la funzione di Rosenbrock e la funzione di Rastrigin. Per prima cosa sono stati ottimizzati gli iperparametri degli algoritmi,

sfruttando i test statistici di Friedman e di Holm. Successivamente è stata eseguita una comparazione a coppie tra gli algoritmi sfruttando il test statistico di Wilcoxon.

Di seguito è riportato un breve sommario degli argomenti trattati in questo lavoro di tesi. Nel Capitolo 1 è fornito un 'background' per il resto dell'elaborato introducendo gli Algoritmi Evolutivi (EA) e la Ricerca Locale. Nel Capitolo 2 si passa a descrivere la potenza di queste due tecniche quando sono combinate a formare gli Algoritmi Memetici, e vengono discusse alcune delle questioni relative al design di queste strategie, oltre ad una descrizione delle istanze degli Algoritmi realizzati. Nel Capitolo 3 sono descritti Python e la libreria DEAP, oltre i test statistici non parametrici che sono stati utilizzati nell'analisi delle performance dei tre algoritmi, ovvero i testi di Friedman, di Holm e di Wilcoxon. Infine, nel Capitolo 4, sono presentate le funzioni di benchmark utilizzate per effettuare il tuning degli algoritmi, ovvero la funzione della sfera, la funzione di Rosenbrock e la funzione di Rastrigin, e successivamente sono riportati i risultati delle comparazioni.

Capitolo 1

Ottimizzazione e Metaeuristiche

1.1 Problemi di ottimizzazione

Un problema di ottimizzazione è un problema computazionale in cui si cerca la miglior soluzione fra tutte le soluzioni possibili. Questo compito si traduce nel massimizzare o minimizzare una funzione, detta *funzione obiettivo*, in un certo dominio che rappresenta il range di soluzioni possibili per un una certa situazione.
Formalmente possiamo definire il problema di ottimizzazione come la ricerca dell' *ottimo* $f(x)$, dove *ottimo* sta per *min* o *max*, $f : R^n \longrightarrow R$ è la funzione obiettivo e S è il set di soluzioni ammissibili.
In questo testo sono stati considerati problemi di minimizzazione.
Per *minimo globale* si intende un punto $x^* \in S$ tale che

$$f(x^*) \leq f(x) \quad \forall\, x \in S$$

Un problema di ottimizzazione ammette una soluzione se esiste il minimo globale $x^* \in S$ per la funzione obiettivo. Invece si definisce *minimo locale* un punto $\bar{x} \in S$ se esiste un intorno $I(\bar{x})$ tale che

$$f(\bar{x}) \leq f(x) \quad \forall\, x \in I \cap S$$

Ovviamente ogni minimo globale è anche locale ma non viceversa.
Per risolvere un problema di ottimizzazione bisogna quindi trasformarlo in termini matematici e successivamente sviluppare una strategia di risoluzione. Analizziamo in dettaglio alcune di queste strategie.

1.2 Algoritmi Evolutivi

Gli *Algoritmi Evolutivi* (anche noti come **EA**) sono metodi metaeuristici, ovvero strategie di problem solving basate su processi stocastici che hanno

lo scopo di trovare soluzioni approssimate a problemi con elevata complessità computazionale. Il loro nome deriva dal fatto che utilizzano meccanismi ispirati ai principi evolutivi naturali teorizzati da Charles Darwin.

BEGIN

INITIALIZE population with random candidate solutions;
EVALUATE each candidate;
REPEAT UNTIL (TERMINATION CONDITION is satisfied) DO

1. SELECT parents;
2. RECOMBINE pairs of parents;
3. MUTATE the resulting offspring;
4. EVALUATE new candidates;
5. SELECT individuals for the next generation;

OD

END

Tabella 1.1: Pseudo-codice di un *Algoritmo Evolutivo* [6]

In Figura 1.1 è illustrato uno pseudo-codice di un Algoritmo Evolutivo. Gli EA sono una tecnica population-based, ovvero fondata sullo studio di una popolazione e del suo sviluppo in successive generazioni attraverso dei meccanismi che emulano l' evoluzione biologica. La popolazione è composta da individui cioè potenziali soluzioni del problema.

Gli individui sono entità duali: si chiamano fenotipi nel contesto del problema originale da ottimizzare, mentre genotipi quando sono codificati nell' algoritmo. Lo spazio dei fenotipi può essere molto diverso da quello dei genotipi: il processo di ricerca avviene nello spazio dei genotipi, mentre sono i tratti fenotipici a determinare quanto l' individuo si adatta all' ambiente. Il genotipo si chiama anche cromosoma. Il cromosoma o genotipo è formato da variabili dette geni, pezzi della codifica responsabili dei tratti della soluzione. Una configurazione possibile di un gene è detta allele.

Un multiset di genotipi si dice popolazione. Fissata la sua dimensione, la popolazione è inizializzata attraverso la generazione casuale dei suoi individui. Successivamente si procede alla valutazione degli individui grazie ad una funzione di fitness (evaluation function), che assegna uno o più valori ad ogni individuo in base alla sua qualità come soluzione. La funzione di fitness è la trasformazione della funzione obiettivo; indica in che direzione deve andare il processo di ottimizzazione.

L' evoluzione è poi determinata dalla combinazione di due forze fondamentali: gli operatori di variazione (riproduzione e mutazione), che contribuiscono ad accrescere la diversità della popolazione; gli operatori di selezione, che si occupano di indirizzarne la qualità. Nella fase successiva alla valutazione interviene un meccanismo di selezione, il *parent selection operator*, che confronta i valori di fitness degli individui e, in base a queste misure, seleziona gli individui destinati alla riproduzione. È un processo stocastico, per cui gli individui migliori hanno maggior probabilità di essere selezionati. L'operatore di selezione opera a livello della popolazione, ed ha come obiettivo quello di sfruttare le migliori caratteristiche di un buon individuo affinchè queste possano trasmettersi nelle successive generazioni.

Dopo la selezione, interviene l' *operatore di ricombinazione* (detto anche di riproduzione o crossover): è un operatore di variazione binario, ovvero è applicato su una coppia di individui. Selezionati due individui, i loro cromosomi sono ricombinati ed il modo in cui avviene questo processo dipende da decisioni casuali. Il risultato della ricombinazione è uno o più individui, l'*offspring*.

Un altro operatore di variazione che può intervenire è quello di mutazione: si applica ad un unico individuo e restituisce un individuo leggermente modificato. Come la ricombinazione, è un processo stocastico, ovvero la scelta dei geni da variare è random.

Successivamente, avviene una nuova selezione, la *survivor selection*, simile alla parent selection ma a differenza di quest' ultima, è un processo deterministico. Infatti il suo compito è quello di scegliere gli individui che costituiranno la generazione successiva, basandosi sui loro valori di fitness.

Il processo di evoluzione si ripete seguendo questo procedimento fino al raggiungimento di un criterio di terminazione fissato all' inizio dal programmatore: questa condizione può riguardare il tempo massimo di elaborazione della CPU, il numero massimo di valutazioni di fitness effettuate dal programma, o il raggiungimento di una soglia di errore minima per la valutazione della fitness. La scelta dipende dal problema analizzato e in generale è possibile concatenare più condizioni di terminazione.

Il framework degli Algoritmi Evolutivi si mostra semplice da capire e da maneggiare, così da rendere questi algoritmi molto versatili. Infatti sono applicabili ad un' ampia gamma di problemi di ottimizzazione ottenendo in genere buoni risultati. Confrontandoli con gli algoritmi classici scopriamo che mentre questi ultimi funzionano meglio su problemi lineari, quadratici, convessi, unimodali e separabili, gli EA sono superiori per problemi con discontinuità, non differenziabili e multimodali.

Un problema comune riscontrato nell'utilizzo degli EA è quello della convergenza prematura, ovvero il fenomeno per cui la popolazione converge troppo rapidamente ad un punto di ottimo locale. Responsabile di questo pericolo è la perdita di diversità da parte della popolazione. Con diversità si

intende la misura del numero di soluzioni differenti presenti nella popolazione. Se presente, la diversità permette un' esplorazione globale dello spazio di ricerca, evitando così una convergenza prematura della popolazione.

Esistono diverse istanze di Algoritmi Evolutivi, che differiscono per dettagli tecnici legati alla rappresentazione degli individui oppure all' implementazione degli operatori definiti in precedenza. Queste differenze in generale hanno origine storica. Le versioni più popolari di EA sono:

- gli Algoritmi Genetici *GA*
- Strategie Evolutive (Evolution strategy) *ES*
- Evoluzione Differenziale (differential evolution)
- Evolutionary Programming *EP*
- Genetic Programming *GP*
- Particle Swarm Optimization *PSO*

1.2.1 Algoritmo Genetico

L'Algoritmo Genetico è tra i tipi più popolari di EA. In generale presenta queste caratteristiche:

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

Tabella 1.2: Caratteristiche dell'*Algoritmo Genetico*

In Figura 1.2 sono mostrate le caratteristiche principali di un Algoritmo Genetico. In questa rappresentazione i genotipi sono stringhe di bit. Il crossover utilizzato opera scegliendo un numero random tra 1 e $L-1$ dove L è la lunghezza del genotipo, dopodiché taglia i due individui genitori nel punto scelto e crea due individui figli scambiando le code. Il One-point crossover può essere facilmente generalizzabile ad un n-point crossover, dove il cromosoma è separato in n punti invece di uno.

La mutazione bit flip invece inverte gli 0 e gli 1. Il numero di valori cambiati non è fisso per ogni individuo: infatti corrisponde alla lunghezza L per p_m dove p_m è la probabilità di mutazione.
Con la *Fitness Proportionate Selection* ogni individuo ha una probabilità di accedere alla riproduzione proporzionale al suo valore di fitness. Questa selezione è implementata col metodo della Roulette Wheel. Il metodo della *Roulette Wheel* si può immaginare considerando un disco con n sezioni, dove n è il numero di individui, ed ogni sezione ha un'ampiezza proporzionale al

valore di fitness dell'individuo. La selezione dei superstiti è invece generazionale, ovvero l' intera popolazione è sostituita con gli individui della nuova generazione (age based).

Anche gli Algoritmi Genetici possono presentare delle variazioni. L'istanza presentata in questo caso è detta *Simple Genetic Algorithm* (SGA).

1.2.2 Particle Swarm Optimization

Un'altra metaeuristica appartenente alla classe degli Algoritmi Evolutivi è rappresentata dalla *Particle Swarm Optimization* PSO (letteralemnte ottimizzazione con sciami di particelle).

La PSO è un algoritmo ispirato al movimento degli sciami. Gli sciami in natura costituiscono una popolazione di individui interagenti che si adattano collettivamente alle condizioni ambientali. È quindi definibile una swarm intelligence (intelligenza di sciame), ovvero l'adattamento collettivo degli individui che ne fanno parte. Si ispira a modelli sociali di adattamento, come gli stormi di uccelli e i banchi di pesci.

Rappresentazione	Real-valued vectors
Ricombinazione	Nessuna
Mutazione	Adding velocity vector
Parent Selection	Deterministic
Survival Selection	Generational

Tabella 1.3: Caratteristiche dell'*Particle Swarm Optimization*

Le caratteristiche del PSO sono illustrate in Figura 1.3. La popolazione è composta da individui detti particelle, che sono coppie di posizioni e di velocità. La prima generazione è inizializzata in modo casuale. Successivamente l'individuo è aggiornato con nuovi valori di posizione e velocità calcolando un vettore di perturbazione v' che dipende dalla velocità di spostamento corrente, dalla conoscenza dello spazio di fitness e dalla miglior soluzione esaminata fino a quel punto.
Quindi invece di usare gli operatori genetici, ogni particella aggiusta la propria *traiettoria* basandosi sul suo percorso precedente nonchè su quello di tutto lo sciame. L'algoritmo inoltre utilizza dei piccoli jittering casuali per evitare l'intrappolamento in minimi locali.

1.3 Local Search

Una tecnica euristica alternativa agli EA è rappresentata dagli algoritmi di *Local Search* (Ricerca Locale). Un algoritmo di questa classe parte da una soluzione iniziale x e iterativamente si muove sulle soluzioni vicine nell'intorno I(x) al fine di trovare una soluzione x' che sia migliore di x. Se tale soluzione esiste, allora prende il posto della soluzione iniziale e il processo si ripete esaminando l'intorno I(x'). I Local search si muovono da una soluzione all'altra nello spazio delle soluzioni candidate applicando cambiamenti locali, fino a che la soluzione ottima è trovata o è raggiunto un altro criterio di terminazione. La scelta di quale intorno esaminare è presa utilizzando informazioni che riguardano solo le soluzioni nell'intorno I(x), da qui il nome di Ricerca *Locale*.

I meccanismi alla base di questi algoritmi si ispirano di solito a fenomeni naturali. Oltre ad essere facili da implementare, gli algoritmi di Local Search possono spesso risolvere problemi complessi e sono applicabili su un' ampia varietà di problemi; inoltre sono facilmente modificabili ed adattabili al problema da risolvere.

Purtuttavia, sono algoritmi *incompleti*, nel senso che non c'è nessuna garanzia che venga trovata la soluzione; sono anche inclini a visitare le stesse zone più volte, restando così incastrati in una zona dello spazio di ricerca senza poter liberarsi se non usando meccanismi specifici. Questo caso si verifica quando non ci sono configurazioni migliori rispetto a quella attuale negli intorni: l' algoritmo è così fermo ad un punto di ottimo locale. Questo problema può essere risolto facendo ripartire il programma con condizioni iniziali differenti, o utilizzando schemi di variazione più complessi basati sulle iterazioni, come succede nel Local Search Iterato, o sfruttando la memoria come fa il Simulated Annealing.

1.3.1 Hill Climbing

Il metodo di Local Search più facile da implementare è l'Hill Climbing. È una variante degli algoritmi *generate and test*. Come gli algoritmi generate and test, infatti genera delle possibili soluzioni, le testa per verificare se tra queste è presente quella attesa e se non è presente genera altre soluzioni ripetendo il processo. Dalla procedura di test però raccoglie delle informazioni che poi utilizza nella generazione delle successive soluzioni decidendo in che direzione muoversi nello spazio di ricerca.

Inoltre utilizza un approccio *greedy* (letteralmente *avido*): infatti qualunque sia il punto dello spazio di ricerca in cui si trova, le mosse successive dell'algoritmo sono solo quelle che ottimizzano la funzione di fitness.

```
x = initial solution
WHILE (termination criterion is not satisfied) DO:
    GENERATE a neighbor solution x'
    if f(x') ≤ f(x)
        REPLACE x with x'
```

Tabella 1.4: Pseudo-codice dell'*Hill Climbing* [21]

L'algoritmo parte da una soluzione candidata arbirtraria. Ad ogni iterazione, l'algoritmo cambia la soluzione corrente tipicamente applicando un operatore di mutazione con lo scopo di trovare una soluzione migliore. La ricerca si ferma quando viene trovata la soluzione ottima oppure quando è raggiunto il numero massimo di valutazioni di fitness. Ci sono diverse varianti dell'algoritmo di Hill Climbing. Quello presentato nello pseudo-codice è il Simple Hill Climbing. A seconda di come è costruito l'intorno, possiamo avere lo *Stochastic Hill Climbing*, che seleziona una soluzione vicina in modo random, e lo *Steepest Hill Climbing*, che invece considera l'estremo più vicino alla soluzione attuale.

1.3.2 Simulated Annealing

Seppur semplice da implementare, l'algoritmo di Hill Climbing è incapace di gestire spazi di ricerca irregolari perchè rischia di restare intrappolato nei molteplici ottimi locali. Per evitare questo pericolo la strategia adottata dall'Hill Climbing è quella di far ripartire il programma da punti iniziali differenti, ma consiste in una forzatura.

Esistono metodi di Local Search che adottano metodi più sofisticati per evitare questi minimi locali. Un esempio è il *Simulated Annealing* (SA), una metaeuristica che esegue una ricerca simile a quella dell'Hill Climbing, ma a differenza di quest'ultima permette anche spostamenti che "peggiorano" la soluzione.

Il Simulated Annealing (SA) è un algoritmo di Local Search. Si ispira al fenomeno del lento raffreddamento dei metalli, che è caratterizzato da una progressiva riduzione dei movimenti atomici con il raggiungimento di uno stato energetico più basso. Nasce come simulazione della tempra (annealing) dei solidi. Come nel fenomeno a cui si ispira, il SA lascia che le soluzioni varino in modo significativo quando la Temperatura è alta, diminuendo queste variazioni man mano che la Temperatura si abbassa. Una caratteristica che distingue questo metodo dagli altri di Local Search è la strategia che attua per evitare minimi locali: infatti permette movimenti della soluzione

verso punti con valori di fitness inferiori a quelli della soluzione considerata. L'accettazione di queste soluzioni meno convenienti permette di evitare in generale il problema degli ottimi locali.
La probabilità di queste sostituzioni dipende esponenzialmente dal run-time, essendo più alta all'inizio del processo e più bassa alla fine. In particolare, indicando con P la probabilità di sostituzione:

$$P = \begin{cases} 1 & \text{se } \triangle \text{f} > 0 \\ e^{-\frac{\triangle f}{T}} & \text{altrimenti} \end{cases}$$

dove T è un parametro dipendente dal tempo chiamato *Temperatura*. La dipendenza dal tempo non è tuttavia definita, può variare in base all'istanza di SA che si vuole implementare.

1.3.3 Tabu Search

Anche il *Tabu Search* è un'estensione dell' Hill Climbing. La caratteristica che distingue questa tecnica dagli altri metodi di Local Search è il fatto di rendere *proibite* (o *tabu*) le ultime mosse eseguite nel cammino di ricerca. Durante il processo infatti l'algoritmo tiene memoria, oltre che delle informazioni locali, anche di alcune informazioni relative alle mosse compiute. Tali informazioni guidano le mosse successive: viene così impedito alla ricerca di ripercorrere i propri passi, evitando che l'algoritmo ricada in minimi locali.

Capitolo 2

Algoritmi Memetici

Gli *Algoritmi Memetici* (MA) sono metaeuristiche che combinano il framework di ricerca globale basato sulla popolazione con uno o più metodi di ricerca locale. Mentre gli Algoritmi Genetici cercano di emulare l'evoluzione biologica, gli Algoritmi Memetici prendono ispirazione dall'evoluzione culturale. Come unità di evoluzione culturale si considera il *meme*, dal greco $\mu\iota\mu\eta\mu\alpha$, "imitazione": il meme è definito come informazione custodita nella memoria individuale che può essere imparata e trasmessa ad altri esseri umani. Alcuni esempi di meme sono le idee, le melodie, gli slogan ecc.

Come i geni si propagano "saltando" da corpo a corpo attraverso la riproduzione, così i memi si propagano passando da una mente all'altra, attraverso un processo che può essere definito di *imitazione*. La disciplina che studia i memi e la loro diffusione si chiama *memetica*: secondo questa teoria, i memi che meglio si adattano ad una situazione hanno più probabilità di diffondersi e resistere nel tempo.

Questa caratterizzazione dei memi suggerisce che nel processo di evoluzione culturale l'informazione non venga semplicemente trasmessa tra gli individui restando inalterata, bensì sia elaborata e migliorata dalla parte comunicativa della trasmissione. Negli MA questo miglioramento è realizzato grazie alle tecniche di ricerca locale.

Gli MA possiedono il potere di entrambe le evoluzioni, combinando l'adattamento evolutivo di una popolazione con l'apprendimento individuale dei suoi membri. Questi algoritmi hanno rappresentato un'innovazione nel problem solving euristico mostrando che i problemi di ottimizzazione possono essere maneggiati con più efficacia attraverso l'ibridizzazione di tecniche già esistenti ottenendo prestazioni migliori rispetto a quelle ottenute utilizzando questi metodi singolarmente.

Procedure Memetic Algorithm
Initialize: Generate an initial population;
while Stopping conditions are not satisfied **do**
    Evaluate all individuals in the population
    Evolve a new population using stochastic search operators
    Select the subset of individuals, Ω_{il}, that should undergo the individual improvement procedure
    for each individual in Ω_{il} **do**
        Perform individual learning using memes with frequency or probability of f_{il}, for a period of t_{il}
        Proceed with Lamarckian or Baldwinian learning
    end for
end while

Tabella 2.1: Pseudo-codice di un *Algoritmo Memetico* [14]

2.1 Un template standard di MA

Abbiamo detto che gli MA uniscono tecniche di ricerca globale e di ricerca locale: le prime si occupano dell' *exploration* (letteramente "esplorazione") cercando di individuare le regioni dello spazio di ricerca più promettenti; le altre esaminano l'intorno di alcune soluzioni candidate svolgendo la funzione di *exploitation* ("sfruttamento").

La ricerca locale è responsabile del *Local Improvement* dell'offspring: applicata ad una soluzione dall'offspring del processo evolutivo, la migliora analizzando le altre soluzioni nei suoi intorni. Quando il processo si arresta, a seguito di un criterio di terminazione definito dall'utente, la soluzione corrente è sostituita con la migliore trovata. L'obiettivo di questo processo è simile a quello degli operatori di mutazione evolutivi, e infatti la ricerca locale vista nel contesto degli Algoritmi Memetici può essere considerata come un operatore di "macromutazione". Per questo motivo, l'operatore di mutazione può essere anche omesso nell'Algoritmo Memetico: se inserito però può rafforzare la diversità della popolazione.

Una proprietà fondamentale degli MA è infatti quella di sfruttare tutta la conoscenza disponibile per un dato problema, migliorando alcuni individui della popolazione attraverso i metodi di ricerca locale.

Da un punto di vista generale, un MA standard può essere anche visto come uno o più processi di local search che agiscono su un set di soluzioni le quali interagiscono in maniera positiva attraverso la ricombinazione.

In Figura 2.1 è presentato lo pseudo-codice di un MA. Attraverso la pro-

cedura di inizializzazione viene creato il set iniziale delle soluzioni, ovvero la popolazione. A differenza degli EA, in cui la popolazione iniziale è generata in modo random, gli MA possono partire da soluzioni già di alta qualità utilizzando la ricerca locale nella prima generazione di individui.

Dopo l'inizializzazione, segue la fase *cooperate and improve*: per *cooperate* si intende il meccanismo di ricombinazione mentre con *improve* l'applicazione di strategie di ricerca locale sulle soluzioni. Infine la procedura *compete*, che è usata nella ricostruzione della popolazione alla fine del ciclo evolutivo sfruttando i vecchi e i nuovi individui.

2.2 Questioni relative alla progettazione

Nell'implementazione di un MA, le scelte da compiere riguardo il design dell'algoritmo riguardano soprattutto le strategie di ricerca locale, non solo dal punto di vista della parametrizzazione ma anche del comportamento interno della ricerca locale e del modo in cui interagisce con il resto dell'algoritmo. I parametri da configurare sono:

- il *metodo di ricerca locale*: ovvero quale algoritmo utilizzare (il *meme*);
- la *frequenza di ricerca locale*: indica quanto spesso va applicata la ricerca locale durante il processo evolutivo, ovvero su quanti individui della popolazione;
- l' *ordine rispetto agli operatori genetici*: se va applicata prima o dopo gli operatori di crossover e mutazione;
- il *meccanismo di selezione degli individui*: ovvero un criterio in base a cui selezionare gli individui da migliorare con la ricerca locale nel caso in cui la frequenza non sia massima;
- l' *intensità di ricerca locale*: indica quante iterazioni di ricerca locale vanno eseguite per ogni ciclo evolutivo;

In alternativa alla frequenza , è possibile configurare una probabilità di ricerca locale, richiamando quindi il processo in maniera probabilistica.
Il valore della probabilità di applicazione della ricerca locale dipende dalle conoscenze del problema, e la sua determinazione è spesso arbitraria. Per questo motivo sono stati sviluppati degli algoritmi *adattativi*, tali che l'algoritmo impari da sé quale sia il setting più appropriato. Per indicare le tecniche per cui non tutti gli individui sono sottoposti alla ricerca locale (ovvero quando la probabilità è diversa da 1), si usa il termine *partial Lamarckian*. Risulta in questi casi necessario definire un meccanismo di selezione degli individui.

Alcuni esempi di tali meccanismi sono:

- la *random selection*, per cui la scelta avviene in maniera casuale;
- la *fitness-based selection*, dove sono scelti i migliori tra gli individui;
- la *stratified selection*, per cui la popolazione è ordinata secondo i loro valori di fitness; successivamente gli individui sono divisi in n intervalli (dove n è il numero di individui che saranno selezionati) e da ogni range è selezionato un individuo.

Va considerato poi che la ricerca locale può essere integrata in due modi distinti nel framework evolutivo:

- uno è rappresentato dal cosiddetto *life-time learning*, ovvero l'applicazione del metodo di ricerca sulla soluzione candidata;
- l'altro consiste nell'applicazione della strategia locale durante la generazione della soluzione;

quest'ultimo metodo definisce una nuova classe di algoritmi memetici in cui viene selezionato l'offspring più conveniente tra tutti quelli possibili; ad esempio la ricerca locale può essere utilizzata per trovare il *cutting-point* più favorevole per il processo di 1-Point crossover dell'algoritmo genetico.

Un obiettivo nella realizzazione di un MA deve essere l'equilibrio tra *exploration* e *exploitation*. Questo si ottiene bilanciando la ricerca locale e quella globale dell'algoritmo memetico. Tipicamente la ricerca locale è eseguita in ogni iterazione, quindi con la più alta frequenza possibile, ed agisce fino a che non è trovato un ottimo locale. Per bilanciare questo potente operatore di *expoitation* vengono usate "forti" mutazioni prima di applicare la ricerca locale, aumentando la componente di *exploration*.

2.3 Estensioni degli Algoritmi Memetici

Gli Algoritmi Memetici sono tra i metodi più usati per la risoluzione di problemi. Seppur sia possibile delinearne lo schema generale, tuttavia esistono varie declinazioni di MA: un modello più sofisticato è rappresentato dagli *MA Multiobiettivo* (o **MOMA**, che sono applicati a problemi che esibiscono obiettivi multipli, e quindi più funzioni obiettivo da ottimizzare.

Abbiamo poi gli *Adaptive Memetic Algorithms*: questo tipo di algoritmo cerca i valori da assegnare ai parametri della ricerca locale sfruttando le conoscenze disponibili sul problema. Tra gli algoritmi *adattabili* ci sono gli *Algoritmi multi-Memetici* in cui ogni soluzione possiede un gene con le informazioni relative al metodo di ricerca locale da applicare sulla soluzione stessa. Questo modello può essere migliorato introducendo una popolazione

di operatori di ricerca locale che *coevolve* con la popolazione di individui, rendendo i due processi di evoluzione simbiotici.

Gli Algoritmi Memetici possono essere a loro volta combinati con altre tecniche di ricerca, anche esatte e non metaeuristiche. Queste tecniche esatte possono essere utilizzate per rinforzare la ricerca locale, esplorando intorni più estesi, o per indirizzarne la ricerca. La combinazione può essere di tipo *collaborativo* (quando gli algoritmi sono sequenziali) oppure *integrativo* (una tecnica è contenuta nel framework dell'altra). Tuttavia questi algoritmi *ibridi* non sono "completi", nel senso che non garantiscono l'arrivo ad una soluzione ottima.

2.4 Implementazione di un Algoritmo Memetico

L'istanza presentata in questa tesi vede la combinazione di un Algoritmo Genetico e un algoritmo di Hill Climbing. La Figura 2.2 ne illustra alcune caratteristiche.

Rappresentazione	valori reali
Ricombinazione	2-Points Crossover
Mutazione	Gaussiana ($\mu = 0$; $\sigma = 1$)
Parent Selection	tournment (size = 3)
Survivor Selection	Generazionale

Tabella 2.2: Caratteristiche dell'*Algoritmo Memetico* implementato

L' Algoritmo Genetico, che costituisce il framework dell'Algoritmo Memetico, è quello più usato nella realizzazione degli MA; infatti gli MA sono anche definiti GA-ibridi. la popolazione viene inizializzata attraverso la generazione casuale di $n = 100$ individui costituiti da 10 geni ognuno. La rappresentazione dei geni è a valori reali, secondo le direttive dettate nella competizione sull'ottimizzazione dei parametri reali che si è tenuta durante il Congresso sulla Computazione Evolutiva IEEE del 2005 (CEC'2005 Special Session on Real Parameter Optimization). Le soluzioni della prima popolazione sono generate in modo casuale all'interno dello spazio di ricerca. Vengono poi valutate attraverso la funzione di fitness. Definendo un rate di crossover e di mutazione, una parte degli individui è ricombinata oppure mutata attraverso i due operatori appositi.
Come operatore di ricombinazione è stato utilizzato il `2-Points Crossover`, equivalente ad eseguire il 1-Point Crossover due volte su due punti distinti del cromosoma. L'operatore di mutazione invece è gaussiano: aggiunge un valore random preso da una distribuzione Gaussiana di media 0 e varianza 1, ad uno o più geni di un individuo. Bisogna distinguere a tal proposito due probabilità relative alla mutazione: il rate di mutazione, che individua

la frazione di individui che sarà sottoposta alla mutazione, e la probabilità indipendente per ogni attributo di essere mutato, che indica la frazione di geni da mutare dell'individuo. Nell'istanza implementata la mutazione è effettuata su ogni individuo, quindi il rate di mutazione è massimo.

La mutazione e la ricombinazione potrebbero portare i geni a fuoriuscire dal dominio della funzione: per questo gli individui mutati e ricombinati sono nuovamente valutati e se i loro geni superano il range massimo sono scartati oppure modificati in modo da rientrare nel dominio della funzione.

Dopo i due operatori evolutivi interviene l'operatore di Local Search: i migliori individui sono selezionati e sottoposti alla Ricerca Locale attraverso il metodo dell'Hill Climbing; gli individui migliorati rimpiazzano gli individui inziali nella popolazione. Attraverso un metodo generazionale, viene sostituita la popolazione precedente con l'offspring, risultato della ricombinazione, della mutazione e della ricerca locale.
Il processo è ripetuto fino a che l'algoritmo raggiunge il criterio di terminazione, in questo caso posto ad un numero massimo di valutazioni di fitness.

Prima sono stati implementati i due algoritmi costituenti, ovvero il GA e l'Hill Climbing; Gli iperparametri, ovvero i parametri che controllano il processo di evoluzione dell'algoritmo, sono ottimizzati attraverso un metodo di *Grid Search*, ovvero di ricerca manuale valutando dei set di possibili valori.

Capitolo 3

Strumenti Utilizzati

3.1 Python e DEAP

Gli Algoritmi sono stati implementati su Python, un linguaggio di programmazione ad alto livello, orientato agli oggetti, che ha tra i principali obiettivi dinamicità, semplicità e flessibilità. Python è noto per essere un linguaggio immediatamente intuibile, possedendo una sintassi di programmazione molte semplice e dei codici leggibili e chiari. In Python è possibile utilizzare i *framework*, ovvero dei file che raggruppano costanti, funzioni e classi, che rendono più agile la programmazione.

In questo lavoro di tesi è stato utilizzato DEAP (**Distributed Evolutionary Algorithms in Python**) sviluppato al **Computer Vision and Systems Laboratory (CVSL)** all'Università Laval in Quebec City, Canada. DEAP è un recente framework di computazione evolutiva per la prototipazione rapida e il test delle idee. Gli Algoritmi Evolutivi possono assumere delle strutture molto articolate e per questo anche i framework progettati meglio possono rivelarsi complicati da comprendere. Per facilitare la programmazione, questi framework nascondono quanto più possibile i dettagli dell'implementazione e per questo motivo sono definiti "black-box" framework. Proprio per la loro natura a *scatola nera*, più elaborati diventano, più oscuri sono da comprendere, soprattutto per programmatori meno esperti. Il design di DEAP si distacca da questi framework in quanto punta a rendere gli algoritmi espliciti e le strutture dati chiare. DEAP combina 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.

DEAP è progettato per aiutare i ricercatori a sviluppare algoritmi evolutivi personalizzati. Il framework è basato su tre principi fondamentali:

- le strutture dati sono la chiave della computazione evolutiva; devono facilitare l'implementazione degli algoritmi ed essere molto semplici da personalizzare;

- gli operatori di selezione e i parametri degli algoritmi hanno una forte influenza sull'evoluzione, e sono spesso dipendenti dal problema; gli utenti dovrebbero essere capaci di parametrizzare ogni aspetto degli algoritmi con facilità;

- gli EA sono spesso paralleli richiedendo il calcolo simultaneo delle funzioni di fitness per gli individui di una popolazione; quindi meccanismi che implementano paradigmi di distribuzione dovrebbero essere facili da usare.

Con l'aiuto del progetto gemello SCOOP e la potenza del linguaggio di programmazione Python, DEAP realizza questi principi in un design semplice ed elegante.

3.1.1 I Moduli di DEAP

Il nucleo di DEAP è costruito attorno a diverse componenti che servono a definire parti specifiche dell'algoritmo evolutivo. Tra i moduli principali abbiamo il modulo *creator*, utile per costruire strutture dati personalizzate per il problema. Questo modulo permette di:

- creare classi con un unica linea di codice (*inherithance*);

- aggiungere attributi alle classi (*composition*);

- raggruppare classi in un singolo modulo (*sandboxing*);

Attraverso il creator è possibile creare nuove classi partendo da quelle standard; in questo modo, si possono aggiungere degli attributi a classi esistenti per creare nuovi tipi che siano più versatili nel contesto evolutivo. In particolare, questo modulo permette la creazione di genotipi e popolazioni a partire da una qualunque struttura come liste, set o dizionari.

Una struttura fondamentale fornita da DEAP è il *toolbox*, un contenitore per i tools (gli operatori) che si vogliono usare nell'algoritmo. Il toolbox fornisce tutti gli operatori necessari e i loro argomenti in una singola struttura. Viene riempito manualmente con gli operatori scelti a partire dal modulo *tools*. Il modulo tools contiene gli operatori standard della Computazione Evolutiva, ovvero quelli di inizializzazione, mutazione, ricombinazione e selezione. Questo modulo contiene anche diverse componenti utili a raccogliere informazioni nel processo evolutivo, come la **fitness statistics**, o la **hall of fame** che raccoglie gli individui migliori incontrati fino a quel momento. Abbiamo poi il modulo *base* che contiene delle strutture dati usate spesso negli EA ma non implementate nella libreria standard di Python. Inoltre permette di personalizzare gli **initializer**, che servono a generare l'individuo o la popolazione, e gli operatori evolutivi.

Oltre a questi moduli, DEAP presenta il modulo *algorithms*, che fornisce degli esempi di algoritmi evolutivi già implementati, e il modulo *dtm*

(Distributed Task Manager), che offre delle funzioni che possono sostituire quelle standard di Python come **apply** o **map**.

3.2 Test statistici non parametrici

Un metodo statistico si dice *non parametrico* se non fa nessuna ipotesi sulla distribuzione della popolazione o sulla dimensione del campione su cui è applicato, a differenza dei test statistici *parametrici* che invece assumono che il set di dati sia quantitativo, la distribuzione della popolazione sia normale e il campione di dati sia sufficientemente grande.

In generale le conclusioni estrapolate dai test non parametrici non sono solide come quelle dei test parametrici; tuttavia un test non parametrico pone meno condizioni e quindi è applicabile ad una classe più ampia di campioni; infatti i metodi non parametrici sono spesso usati per analizzare dati la cui distribuzione non ha i requisiti richiesti dai test parametrici. Inoltre sono facili da comprendere. Tuttavia c'è la possibilità di sprecare informazioni se applicati in un contesto in cui sia più appropriato un test parametrico.

Per il teorema del *No Free Lunch* sappiamo che non è possibile trovare un algoritmo che applicato ad un problema qualsiasi esibisca una performance che sia migliore di ogni altro algoritmo applicabile. Per questo, quando si tratta di confrontare degli algoritmi, i comportamenti vanno analizzati applicandoli sul singolo problema; in tal caso è lecito chiedersi quale algoritmo funzioni meglio degli altri. Poichè non sono disponibili dei criteri teorici per effettuare questa comparazione, ci affidiamo all'analisi dei risultati empirici attraverso i test non parametrici. I test utilizzati in questa tesi sono il test di Wilcoxon, il test di Friedman ed il test *post-hoc* di Holm.

3.2.1 Wilcoxon matched-pairs signed-ranks test

Il matched-pairs signed-ranks test di Wilcoxon è un'estensione del signed-ranks test di Wilcoxon (che è impiegato per l'analisi di un singolo campione) che coinvolge due campioni dipendenti. Il test confronta due campioni accoppiati e verifica se appartengono alla stessa distribuzione; mira ad identificare differenze che siano significative tra due campioni, e quindi può essere utilizzato per studiare le differenze tra le performance di due algoritmi. Il test calcola le differenze tra ogni coppia dei due campioni e le riordina. Se si ottiene una differenza significativa, questo indica che probabilmente i due campioni rappresentano due popolazioni differenti.

Un presupposto necessario per applicare il test è che il campione di n elementi sia selezionato in modo casuale dalla popolazione che rappresenta; si possono distinguere due varianti del test a seconda delle ipotesi da verificare:

- il *two-sided*, che ha come ipotesi nulla che la mediana delle differenze θ_D sia zero ($H_0 : \theta_D = 0$), contro l'ipotesi alternativa che sia diversa da zero ($H_1 : \theta_D \neq 0$); l'ipotesi alternativa è quindi *non direzionale*;

- il *one-sided*, che ha come ipotesi nulla che la mediana sia positiva, $H_0 : \theta_D > 0$ contro quella alternativa che sia negativa, $H_1 : \theta_D < 0$ o viceversa; l'ipotesi alternativa è quindi *direzionale*.

Per la computazione nel caso di test *two-sided*, si usa la seguente procedura. Sia d_i la differenza tra i risultati dei due algoritmi sull' i-esima di N run dell'algoritmo applicato ad una funzione. Le differenze $|d_i|$ sono ordinate ed enumerate in base al loro valore assoluto, assegnando il valore 1 alla differenza minore e n a quella maggiore (dove n è il numero di differenze diverse da zero); Dopo aver ordinato i valori assoluti delle differenze, il segno di ogni differenza è posto davanti al suo rank. Sia R^+ la somma dei rank che hanno segno positivo (quindi delle run per cui il secondo algoritmo si comporta meglio del primo), e R^- la somma dei rank con segno negativo. I rank nel caso di differenza nulla sono scartati. La relazione che lega R^+ e R^- è la seguente:

$$\sum R^+ + \sum R^- = \frac{n(n+1)}{2} \tag{3.1}$$

Il valore assoluto del più piccolo tra i valori di R^+ ed R^- è detto *Wilcoxon T test statistic*. Se i campioni derivano dalla stessa popolazione e quindi la mediana delle differenze è zero, i valori di R^+ e R^- sono uguali tra loro

$$\sum R^+ = \sum R^- = \frac{n(n+1)}{4} \tag{3.2}$$

Quest'ultimo valore è comunemente indicato come *valore atteso del Wilcoxon T statistic*. Se il valore di R^+ è significativamente più grande rispetto al valore di R^-, questo indica che c'è un'elevata probabilità che il primo campione rappresenti una popolazione con valori più alti rispetto alla seconda, mentre se R^- è significativamente più grande di R^+, questo indica un'elevata probabilità che il secondo campione rappresenti una popolazione con valori più alti della popolazione rappresentata dalla prima. In uno dei due ultimi casi riportati, i dati sono consistenti con una delle ipotesi alternative direzionale, che nel primo caso corrisponde a $H_1 : \theta_D > 0$, mentre nel secondo a $H_1 : \theta_D < 0$. Tuttavia la questione da definire è se la differenza sia significativa, ovvero se sia abbastanza grande da concludere che non sia frutto del caso.

Abiamo detto che il valore assoluto del più piccolo tra i valori di R^+ ed R^- è detto *Wilcoxon T test statistic*. Il valore di T è interpretato sfruttando la Tabella di Valori Critici di T per il Signed-Ranks and Matched-Pairs Signed-Ranks Tests di Wilcoxon (Table B.12 in [27]). Al fine di essere significativo, il valore di T ottenuto deve essere minore o uguale del valore critico di T tabulato ad un livello di significativià predefinito.

Se la dimensione del campione impiegato è relativamente grande (anche se non esiste una convenzione al riguardo), si può impiegare la distribuzione normale per approssimare il Wilcoxon T statistic. L'equazione

$$z = \frac{T - \frac{n(n+1)}{4}}{\sqrt{\frac{n(n+1)(2n+1)}{24}}} \tag{3.3}$$

fornisce l'approssimazione normale per il *Wilcoxon T*. In questa equazione, T rappresenta il valore calcolato del T di Wilcoxon, n il numero di rank per cui la differenza è diversa da zero; al numeratore il termine $n(n+1)/4$ rappresenta il valore atteso di T, mentre il denominatore di z rappresenta la deviazione standard attesa della distribuzione dei campioni del T statistico.

Se è impiegata un'ipotesi alternativa non direzionale, l'ipotesi nulla può essere rigettata se il valore assoluto di z è maggiore o uguale rispetto al valore critico tabulato per il livello di significanza α predefinito. Se invece è impiegata un'ipotesi alternativa direzionale, una delle due possibili ipotesi alternative direzionali sarà supportata se il valore assoluto di z è maggiore o uguale al valore critico tabulato. Quale ipotesi alternativa sia supportata dipende dalla predizione di quale tra R^+ ed R^- sia maggiore. L'ipotesi nulla può essere rigettata se l'ipotesi alternativa direzionale è supportata.

Il test di Wilcoxon è stato implementato sfruttando il pacchetto *stats* disponibile su SciPy, una libreria open-source di algoritmi e strumenti matematici per Python.

3.2.2 Test di Friedman

Il test di Friedman valuta se in un set di k campioni (con $k \geq 2$) ci siano almeno due campioni che rappresentano popolazioni con mediane differenti. Il test è utilizzato per rilevare differenze significative nei comportamenti di due o più algoritmi.

L'ipotesi nulla per il test di Friedman è $H_0 : \theta_1 = \theta_2 = ... = \theta_k$ (la mediana della popolazione rappresentata dal primo campione è uguale alla mediana della popolazione rappresentata dal secondo campione che è uguale alla popolazione rappresentata dal k-esimo campione ecc.). Riguardo ai dati del campione, quando l'ipotesi nulla è vera le somme dei rank (così come la media dei rank) di tutti i k campioni devono essere uguali. L'ipotesi alternativa si scrive H_1 : Not H_0 ed è adirezionale; indica che c'è una differenza

tra almeno due delle k mediane delle popolazioni. Se l'ipotesi alternativa è vera, le somme dei rank (come la media dei rank) di almeno due dei k campioni non sono uguali.

Riguardo alla computazione del test, viene calcolato il ranking dei risultati osservati per l'algoritmo (r_j per l'algoritmo j con k algoritmi) per ogni run, assegnando alla run migliore il rank 1, e alla peggiore il rank k. Sotto l'ipotesi nulla, impiegata supponendo che i risultati degli algoritmi siano equivalenti e quindi i loro rank simili, la statistica di Friedman

$$\chi_F^2 = \frac{12N}{k(k+1)} \left[\sum_j R_j^2 - \frac{k(k+1)^2}{4} \right] \tag{3.4}$$

è distribuita in accordo al χ_F^2 con *k-1* gradi di libertà, essendo $R_j = \frac{1}{N} \sum_i r_i^j$ e N il numero di run. Al fine di rigettare l'ipotesi nulla, il valore calcolato del χ_F^2 deve essere maggiore o uguale del valore critico tabulato di χ^2 al livello di significatività α predefinito.

Il test di Friedman è stato implementato sfruttando la libreria Python fornita dalla piattaforma STAC (**Statistical Tests for Algorithms Comparison**) ([19]), una recente piattaforma per l'analisi statistica e la verifica dei risultati ottenuti dagli algoritmi di intelligenza computazionale. La libreria di STAC è un'implementazione in Python di diversi test statistici che non fanno parte del modulo *scipy.stats*.

3.2.3 Test di Holm

Il test di Holm è una procedura di comparazione multipla che si serve di un algoritmo di controllo e lo confronta con i restanti algoritmi. In generale, l'algoritmo di controllo è quello che ha ottenuto il rank minore nel test di Friedman. Il test di Holm considera una famiglia di ipotesi ognuna legata al confronto tra l'algoritmo di controllo e uno tra gli algoritmi restanti. La variabile statistica per confrontare l'i-esimo e il j-esimo algoritmo è ottenuta dalla relazione

$$z = (R_i - R_j) / \sqrt{\frac{k(k+1)}{6N}} \tag{3.5}$$

Il valore di z calcolato viene utilizzato per trovare la probabilità corrispondente ai valori tabulati della distribuzione normale (il *p-value*) che è confrontato con un valore di significatività α predefinito. La procedura prevede il controllo sequenziale delle ipotesi ordinate in base al loro *p-value*, che possiamo indicare come $p_1, p_2, \ldots\ p_{k-1}$ tali che $p_1 \leq p_2 \leq \ldots \leq p_{k-1}$. Il test di Holm contronta ogni p_i con $\alpha/(k-i)$ a partire dal *p-value* più significativo. Se p_1 è più piccolo rispetto a $\alpha/(k-1)$, l'ipotesi corrispondente è rifiutata e si passa a confrontare p_2 con $\alpha/(k-2)$. Se anche la seconda ipotesi è rifiutata si procede con i *p-value* successivi. Fino a che una delle ipotesi nulle non può essere rifiutata, sono conservate tutte le restanti ipotesi.

Capitolo 4

Esperimenti e Risultati

4.1 Funzioni di benchmark

Le *funzioni di benchmark* (o *funzioni test* o *artificial landscapes*) sono pensate per testare il funzionamento e l'efficienza degli algoritmi di ottimizzazione. Le funzioni test sono spesso problemi artificiali che mettono alla prova l'algoritmo in situazioni "complicate", in questo caso nella ricerca del punto di minimo globale in funzioni con numerosi minimi locali. Le funzioni di benchmark utilizzate sono [15]:

- la funzione della sfera (in Figura 4.1), che appartiene alle funzioni di tipo *bowl-shaped*. La formula è descritta nell'eq. (4.1) dove d è la dimensione:

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

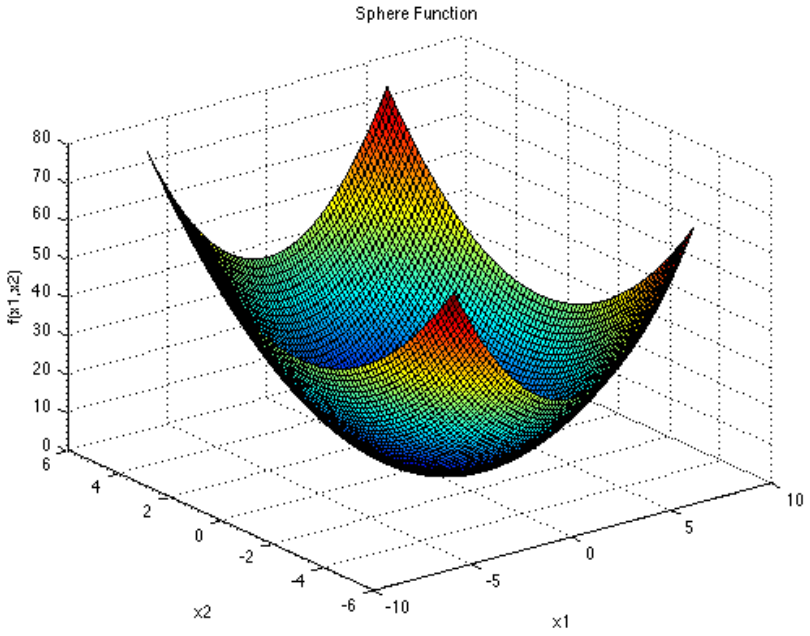
  La funzione è continua, convessa ed unimodale; è stata valutata nell'ipercubo $x_i \in [-5.12, 5.12]$ con $i = 1, ..., d$. In corrispondenza del minimo globale $x^* = (0, ..., 0)$ la funzione vale $f(x^*) = 0$.

- la funzione di Rosenbrock (in Figura 4.2), che appartiene alle funzioni *valley-shaped*; la funzione è descritta dall'eq. (4.2) dove d è la dimensione:

$$f(x) = \sum_{i=1}^{d-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2] \tag{4.2}$$

  È una funzione unimodale, ed il minimo globale si trova in una stretta "valle" parabolica. Nonostante questa valle sia facile da trovare, la convergenza al minimo è tuttavia difficile. (La funzione è di solito valutata all'interno dell'ipercubo $x_i \in [-5, 10]$ con $i = 0, ..., d$. In corrispondenza del minimo globale $x^* = (1, ..., 1)$ la funzione vale $f(x^*) = 0$.

Figura 4.1: Funzione della Sfera



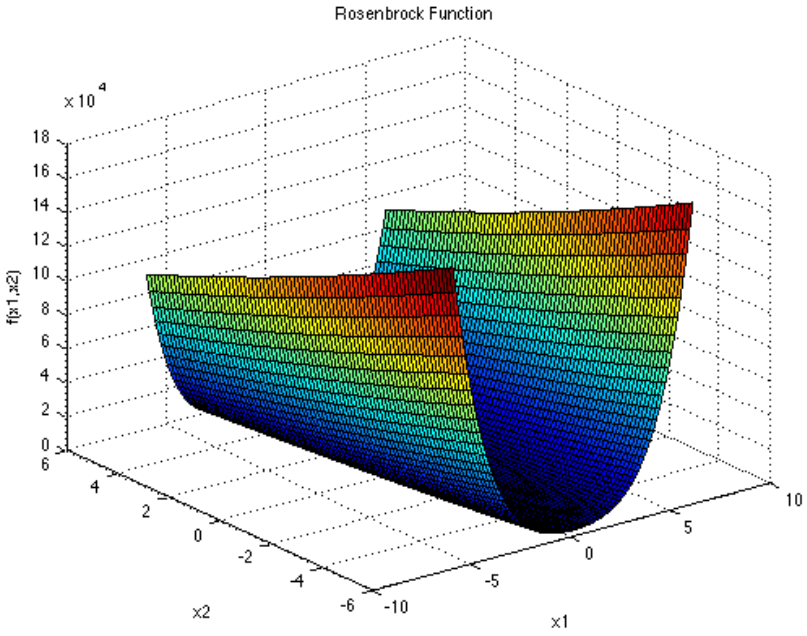


Figura 4.2: Funzione di Rosenbrock

- la funzione di Rastrigin (in Figura 4.3), che ha diversi minimi locali; la funzione è rappresentata dall'eq. (4.3) dove d è la dimensione:

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

  È estremamente multimodale, ma le posizioni dei minimi sono distribuite con regolarità. In generale la funzione è valutata all'interno dell'ipercubo $x_i \in [-5.12, 5.12]$, con $i = 0, ..., d$. In corrispondenza del minimo globale $x^* = (0, ..., 0)$ la funzione vale $f(x^*) = 0$.

4.2 Esperimenti

La sperimentazione di questo lavoro di tesi consiste nello studio delle prestazioni dell'Algoritmo Memetico progettato. In particolare, le prestazioni saranno comparate con altre due meta-euristiche: un Algoritmo Evolutivo e uno di Ricerca Locale. Come Algoritmo Evolutivo è stato implementato un Algoritmo Genetico mentre come Ricerca Locale un algoritmo di Hill Climbing; l'Algoritmo Memetico è quello indicato nel Capitolo II.

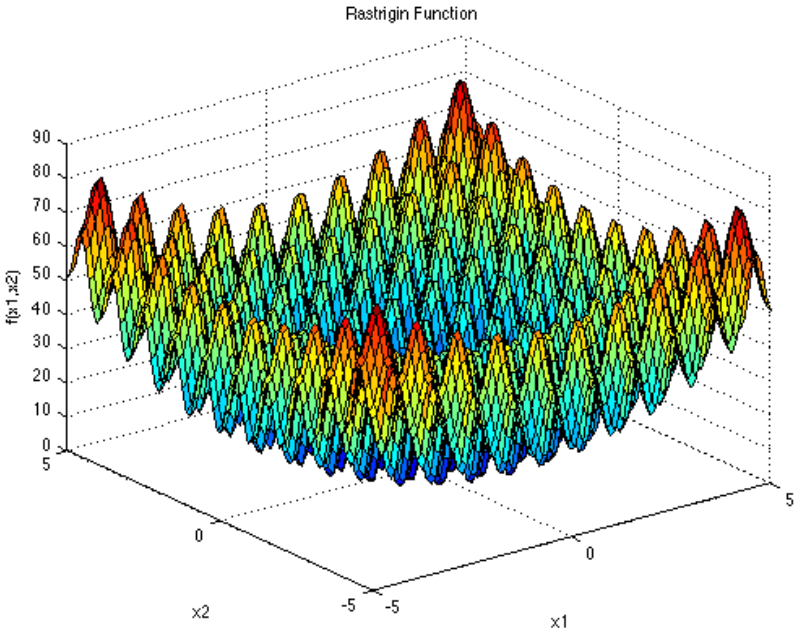


Figura 4.3: Funzione di Rastrigin

L'*Algoritmo Genetico* genera in modo casuale $n = 100$ individui che costituiscono la prima popolazione: tutti gli individui sono valutati attraverso l'operatore di valutazione che applica la funzione di fitness ed assegna un valore di fitness ad ogni cromosoma. Alcuni individui sono selezionati attraverso un operatore di selezione a " Torneo" (*Tournament*) che sceglie gli individui migliori da un campione di 3 individui scelti casualmente, questo per $k = 100$ volte; questi individui formeranno la base dell'offspring. Una parte degli individui è sottoposta ad un processo di crossover, in base ad una *probabilità di crossover* indicata come *cxpb*: come operatore di crossover si sfrutta il *Two-Points Crossover*. Successivamente tutti gli individui vengono mutati attraverso l'operatore di *mutazione gaussiana* di media $\mu = 0$ e deviazione standard $\sigma = 1$, e probabilità indipendente di mutazione per ogni attributo *indpb*. Gli individui restituiti formano la nuova popolazione.

Poichè la mutazione potrebbe portare i geni ad uscire dal range del problema, bisogna anche definire un metodo per gestire l'ammissibilità della soluzione; le strategie provate sono due:

- la sostituzione della soluzione, rimpiazzando il gene non valido con l'estremo più vicino del range;

- lo scarto della soluzione, per cui l'individuo non valido è eliminato.

Questi due metodi sono implementati e testati anche nella configurazione dell'Hill Climbing.

L'*Hill Climbing* parte da un individuo generato in modo casuale. L'individuo è valutato attraverso l'operatore di valutazione, e successivamente sono valutati i suoi intorni applicando una mutazione gaussiana di media $\mu = 0$, deviazione standard $\sigma = 1$ e probabilità indipendente di mutazione per ogni attributo *indpb*. L'individuo mutato è valutato ed il suo valore di fitness è confrontato con quello dell'individuo iniziale: se la fitness è migliore di quella precedente, l'individuo modificato prende il posto di quello corrente, altrimenti viene eliminato. È stato implementato anche un secondo metodo per la costruzione dell'intorno: ogni gene del cromosoma è mutato attraverso dei processi sequenziali ed indipendenti. Si parte dal primo gene, che viene prima aumentato e diminuito attraverso un set di quantità costanti $[-s, -1/s, 0, 1/s, s]$, e la variazione più conveniente è salvata come soluzione corrente; se nessuna variazione migliora il gene, la soluzione non viene sostituita; in questo caso, l'iterazione successiva riduce la quantità s e si ripete il processo per il gene successivo; in questo modo vengono analizzati intorni che si stringono sempre di più attorno ai geni. Possiamo definire questo metodo di variazione *dinamico*.

L'*Algoritmo Memetico* è ottenuto a partire dall'Algoritmo Genetico: dopo che una parte degli $n = 100$ individui sono stati ricombinati e mutati, definita f la frequenza, una frazione $n \cdot f$ di individui tra quelli migliori è selezionata per essere sottoposta al miglioramento locale attraverso l'algoritmo di Hill Climbing: il processo è eseguito per ogni individuo t volte, dove t è l'*intensità di Ricerca Locale*.

4.2.1 Configurazione degli Esperimenti

In primo luogo, si è proceduto al tuning dei tre algoritmi, ovvero alla scelta degli iperparametri ottimali utilizzando le funzioni test già citate. Si noti che questo processo è stato eseguito singolarmente per ognuna delle tre funzioni di benchmark, essendo la configurazione migliore di un algoritmo dipendente dal problema a cui esso è applicato. La Tabella 4.1 rappresenta gli iperparametri migliorati attraverso la procedura di tuning per ogni algoritmo, insieme al relativo set di valori valutati.

Gli iperparametri migliori per l'Algoritmo Genetico e l'Hill Climbing sono utilizzati poi nella configurazione dell'Algoritmo Memetico.

Algoritmi	hyperpar.	set
GA	ammis.	so, sc
	cxpb	0.4, 0.6, 0.8
	indpb	0.05, 0.1, 0.2
HC	ammis.	so, sc
	var.	din, gauss
	indpb(se gauss)	0.05, 0.1, 0.2
MA	freq.	0.065, 0.25, 0.5, 0.9 [10]
	intens.	100, 200, 300 [18]

Tabella 4.1: Iperparametri per i tre Algoritmi: *so* si riferisce agli algoritmi dove è implementata la correzione delle soluzioni, mentre *sc* allo scarto;

Gli esperimenti sono stati eseguiti seguendo alcune delle istruzioni indicate per la competizione *CEC'2005 Special Session* ([24]):

- 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;
- l'algoritmo esegue 100000 valutazioni della funzione di fitness; ogni run si ferma al raggiungimento delle 100000 valutazioni.

4.3 Scelta degli iperparametri

L'analisi è eseguita sfruttando il test di Friedman per la comparazione multipla, valutando se ci sia una differenza significativa tra i comportamenti delle configurazioni. Nei test eseguiti, il livello di significatività scelto è $\alpha = 0.05$. Dalle Tabelle 4.2, 4.3, 4.4 osserviamo che i *p-value* ottenuti sono inferiori rispetto ad α per almeno dieci ordini di grandezza, quindi possiamo rifiutare l'ipotesi nulla del test affermando che c'è una differenza significativa tra le configurazioni valutate.

Tabella 4.2: Risultati del Test di Friedman tra le configurazioni dell'Algoritmo Genetico

	Rank		
Configurazione	f_1	f_2	f_3
so_0.4_0.05	3.48	4.44	**1.80**
so_0.4_0.1	9.56	5.24	5.36
so_0.4_0.2	16.12	8.72	11.96
so_0.6_0.05	4.08	4.56	2.00
so_0.6_0.1	10.64	4.56	5.08
so_0.6_0.2	15.56	8.12	10.56
so_0.8_0.05	3.16	**3.08**	2.20
so_0.8_0.1	9.16	4.56	4.56
sc_0.8_0.2	15.28	8.32	10.68
sc_0.4_0.05	3.92	11.28	15.12
sc_0.4_0.1	8.44	13.20	14.92
sc_0.4_0.2	15.12	16.24	15.32
sc_0.6_0.05	**2.84**	10.92	12.72
sc_0.6_0.1	9.76	12.28	13.24
sc_0.6_0.2	15.4	16.2	12.56
sc_0.8_0.05	3.52	10.64	11.56
sc_0.8_0.1	9.44	12.24	11.20
sc_0.8_0.2	15.52	16.40	10.16
p-value f_1		$1.11 \cdot 10^{-16}$	
p-value f_2		$1.11 \cdot 10^{-16}$	
p-value f_3		$1.11 \cdot 10^{-16}$	

Tabella 4.3: Risultati del Test di Friedman tra le configurazioni dell'Algoritmo Hill Climbing

	Rank		
Configurazione	f_1	f_2	f_3
so_din	1.56	3.64	4.62
so_gauss_0.05	5.12	4.08	3.12
so_gauss_0.1	4.68	2.48	**2.28**
so_gauss_0.2	6.36	**2.12**	3.44
sc_din	**1.44**	5.48	5.10
sc_gauss_0.05	5.80	6.88	5.72
sc_gauss_0.1	4.56	6.44	5.36
sc_gauss_0.2	6.48	4.88	6.36
p-value f_1		$1.11 \cdot 10^{-16}$	
p-value f_2		$1.11 \cdot 10^{-16}$	
p-value f_3		$2.13 \cdot 10^{-12}$	

Tabella 4.4: Risultati del Test di Friedman tra le configurazioni dell'Algoritmo Memetico

	Rank		
Configurazione	f_1	f_2	f_3
0.065_25	3.20	5.96	3.08
0.065_50	3.08	4.92	1.84
0.065_100	**2.12**	**2.32**	**1.20**
0.25_25	6.84	7.96	5.92
0.25_50	6.68	7.36	5.12
0.25_100	2.60	4.88	4.00
0.5_25	9.80	9.20	9.84
0.5_50	9.60	6.56	8.04
0.5_100	5.00	4.56	6.92
0.9_25	11.20	10.68	11.96
0.9_50	11.40	8.24	11.00
0.9_100	6.48	5.36	9.08
p-value f_1		$1.11 \cdot 10^{-16}$	
p-value f_2		$1.11 \cdot 10^{-16}$	
p-value f_3		$2.13 \cdot 10^{-12}$	

Successivamente, è stata condotta un'analisi post-hoc sfruttando il test di Holm a coppie per rivelare differenze determinanti tra le configurazioni confrontate. Come configurazione di controllo si è presa quella che ha ottenuto il rank minore nel test di Friedman. Le configurazioni di controllo sono riportate in Tabella 4.8, mentre i risultati delle comparazioni a coppie sono riportati nelle tabelle 4.5, 4.6 e 4.7.

Nella maggior parte delle comparazioni le ipotesi nulle di uguaglianza sono rifiutate: quando questo non succede significa che le configurazioni portano a risultati simili e sono quindi equivalenti. Possiamo così considerare le configurazioni di controllo come quelle migliori. Nelle Tabelle 4.5, 4.6 e 4.7, sono indicate in corsivo le configurazioni per cui il *p-value* è minore di α.

Le configurazioni testate per l'Algoritmo Genetico sono 18; tra queste, 5 risultano equivalenti alla configurazione di controllo per ognuno dei problemi di benchmark:

- le configurazioni migliori sulla funzione della sfera risultano tutte quelle con il valore di probabilità di mutazione più basso *indpb=0.05*;

- per la funzione di Rosenbrock e Rastrigin le configurazioni migliori comprendono come criterio di ammissibilità delle soluzioni la sostituzione e utilizzano una probabilità di mutazione bassa.

Le configurazioni di Hill Climbing testate sono 8:

- sulla funzione della sfera sia la configurazione di controllo che quella equivalente prevedono come metodo di ricerca degli intorni quello dinamico;

- sulla funzione di Rosenbrock le due configurazioni equivalenti a quella di controllo prevedono come criterio di ammissibilità la sostituzione delle soluzioni;

- per la funzione di Rastrigin le configurazioni equivalenti a quella di controllo sono tre e sono le combinazioni del metodo di ricerca gaussiano con il criterio della sostituzione, oltre un metodo di ricerca dinamico.

Le configurazioni di Algoritmo Memetico testate sono 12; quelle equivalenti a quella di controllo prevedono sempre bassi valori di frequenza di Ricerca Locale.

- per tutte le funzioni funzionano bene valori di frequenza di Ricerca Locale bassi, come *f=0.065*.

Dalla Tabella 4.8 notiamo che per tutti gli algoritmi il criterio di ammissibilità delle soluzioni che funziona meglio sulla sfera è lo scarto, mentre per le altre due funzioni le configurazioni con performance migliori prevedono

la sostituzione; una motivazione per quest'ultimo caso potrebbe essere che il metodo dello scarto comporta una perdita di sforzo computazionale, in quanto scartando un individuo la sua fitness resta inutilizzata.

Le configurazioni migliori dell'Algoritmo Genetico prevedono sempre valori bassi di probabilità di mutazione *indpb*, mentre per il rate di crossover *cxpb* non è evidente nessuna tendenza.

Per l'Hill Climbing il metodo di ricerca migliore nel caso della sfera è quello *dinamico*, mentre per le altre due funzioni è quello gaussiano.

La configurazione migliore per il Memetico è costante per le tre funzioni di benchmark e prevede *f=0.065* e *t=100*.

Tabella 4.5: Risultati del Test di Holm tra le configurazioni dell'Algoritmo Genetico

configurazioni GA f_1	*p-value*	$\alpha/(k-i); \alpha = 0.05$
so_0.4_0.2	*0.0*	*0.002*
so_0.6_0.2	*0.0*	*0.003*
sc_0.6_0.2	*0.0*	*0.003*
sc_0.8_0.2	*0.0*	*0.003*
so_0.8_0.2	$2.8865 \cdot 10^{-15}$	*0.003*
sc_0.4_0.2	$5.3290 \cdot 10^{-15}$	*0.004*
so_0.6_0.1	$2.6352 \cdot 10^{-6}$	*0.004*
sc_0.6_0.1	$4.5861 \cdot 10^{-5}$	*0.005*
so_0.4_0.1	$7.7129 \cdot 10^{-5}$	*0.005*
sc_0.8_0.1	$9.8962 \cdot 10^{-5}$	*0.006*
so_0.8_0.1	*0.0001*	*0.007*
sc_0.4_0.1	*0.0012*	*0.008*
so_0.6_0.05	1	0.01
sc_0.4_0.05	1	0.012
sc_0.8_0.05	1	0.016
so_0.4_0.05	1	0.025
so_0.8_0.05	1	0.05
f_2		
sc_0.4_0.2	*0.0*	*0.002*
sc_0.6_0.2	*0.0*	*0.003*
sc_0.8_0.2	*0.0*	*0.003*
sc_0.4_0.1	$2.8755 \cdot 10^{-10}$	*0.003*
sc_0.6_0.1	$1.4419 \cdot 10^{-8}$	*0.003*
sc_0.8_0.1	$1.5701 \cdot 10^{-8}$	*0.004*
sc_0.4_0.05	$6.1787 \cdot 10^{-7}$	*0.004*
sc_0.6_0.05	$2.0786 \cdot 10^{-6}$	*0.005*
sc_0.8_0.05	$4.9825 \cdot 10^{-6}$	*0.005*
so_0.4_0.2	*0.0015*	*0.006*
so_0.8_0.2	*0.0036*	*0.007*
so_0.6_0.2	*0.0050*	*0.008*
so_0.4_0.1	0.7628	0.01
so_0.6_0.05	1	0.012
so_0.6_0.1	1	0.016
so_0.8_0.1	1	0.025
so_0.4_0.05	1	0.05

configurazioni GA f_3	*p-value*	$\alpha/(k-i); \alpha = 0.05$
sc_0.4_0.05	*0.0*	*0.002*
sc_0.4_0.1	*0.0*	*0.003*
sc_0.4_0.2	*0.0*	*0.003*
sc_0.6_0.1	$4.9737 \cdot 10^{-13}$	*0.003*
sc_0.6_0.05	$6.1888 \cdot 10^{-12}$	*0.003*
sc_0.6_0.2	$1.2400 \cdot 10^{-11}$	*0.004*
so_0.4_0.2	$1.8840 \cdot 10^{-10}$	*0.004*
sc_0.8_0.05	$1.0216 \cdot 10^{-9}$	*0.005*
sc_0.8_0.1	$4.3256 \cdot 10^{-9}$	*0.005*
so_0.8_0.2	$3.2638 \cdot 10^{-8}$	*0.006*
so_0.6_0.2	$4.6020 \cdot 10^{-8}$	*0.007*
sc_0.8_0.2	$1.8509 \cdot 10^{-7}$	*0.008*
so_0.4_0.1	0.0919	0.01
so_0.6_0.1	0.1193	0.0125
so_0.8_0.1	0.2027	0.016
so_0.8_0.05	1	0.025
so_0.6_0.05	1	0.05

Tabella 4.6: Risultati del Test di Holm tra le configurazioni dell'Algoritmo Hill Climbing

configurazioni HC	*p-value*	$\alpha/(k-i); \alpha = 0.05$
f_1		
g_sc_0.2	*2.4324* $\cdot 10^{-12}$	*0.007*
g_so_0.2	*7.4100* $\cdot 10^{-12}$	*0.008*
g_sc_0.05	*1.5557* $\cdot 10^{-9}$	*0.01*
g_so_0.05	*4.3461* $\cdot 10^{-7}$	*0.012*
g_so_0.1	*8.7527* $\cdot 10^{-6}$	*0.016*
g_sc_0.1	*1.3379* $\cdot 10^{-5}$	*0.025*
d_so	0.8624	0.05
f_2		
g_sc_0.05	*4.4793* $\cdot 10^{-11}$	*0.007*
g_sc_0.1	*2.7040* $\cdot 10^{-9}$	*0.008*
d_sc	*6.1811* $\cdot 10^{-}$	*0.01*
g_sc_0.2	*0.0002*	*0.012*
g_so_0.05	*0.0140*	*0.016*
d_so	0.0564	0.025
g_so_0.1	0.6033	0.05
f_3		
g_sc_0.2	*2.7202* $\cdot 10^{-8}$	*0.007*
g_sc_0.05	*4.1175* $\cdot 10^{-6}$	*0.008*
g_sc_0.1	*4.3824* $\cdot 10^{-5}$	*0.01*
d_sc	*0.0001*	*0.012*
d_so	0.0021	0.016
g_so_0.2	0.1881	0.025
g_so_0.05	0.2253	0.05

Tabella 4.7: Risultati del Test di Holm tra le configurazioni dell'Algoritmo Memetico

configurazioni MA	*p-value*	$\alpha/(k-i); \alpha = 0.05$
f_1		
0.9_25	*0.0*	*0.004*
0.9_50	*0.0*	*0.005*
0.5_25	*4.5363* $\cdot 10^{-13}$	*0.005*
0.5_50	*1.776* $\cdot 10^{-12}$	*0.006*
0.25_25	*2.5802* $\cdot 10^{-5}$	*0.007*
0.25_50	*4.6615* $\cdot 10^{-5}$	*0.008*
0.9_100	*9.5426* $\cdot 10^{-5}$	*0.01*
0.5_100	0.0189	0.012
0.065_25	0.8687	0.016
0.065_50	0.8687	0.025
0.25_100	0.8687	0.05
f_2		
0.9_25	*2.4424* $\cdot 10^{-15}$	*0.004*
0.5_25	*1.5156* $\cdot 10^{10}$	*0.005*
0.9_50	*5.7916* $\cdot 10^{-8}$	*0.005*
0.25_25	*2.5549* $\cdot 10^{-7}$	*0.006*
0.25_50	*5.4092* $\cdot 10^{-6}$	*0.007*
0.5_50	*0.0001*	*0.008*
0.065_25	*0.0017*	*0.01*
0.9_100	*0.0114*	*0.012*
0.065_50	0.0323	0.016
0.25_100	0.0323	0.025
0.5_100	0.0323	0.05
f_3		
0.5_25	*0.0*	*0.004*
0.9_25	*0.0*	*0.005*
0.9_50	*0.0*	*0.005*
0.9_100	*8.8817* $\cdot 10^{-14}$	*0.006*
0.5_50	*1.3890* $\cdot 10^{-10}$	*0.007*
0.5_100	*1.2215* $\cdot 10^{-7}$	*0.008*
0.25_25	*1.843* $\cdot 10^{-5}$	*0.01*
0.25_50	*0.0004*	*0.012*
0.25_100	0.0181	0.016
0.065_25	0.1305	0.025
0.065_50	0.5302	0.05

Tabella 4.8: Configurazioni migliori degli Iperparametri per ogni Algoritmo

Algoritmi	f_1	f_2	f_3
GA	sc_0.6_0.05	so_0.8_0.05	so_0.4_0.05
HC	sc_din	so_gauss_0.2	so_gauss_0.1
MA	0.065_100	0.065_100	0.065_100

Tabella 4.9: *p-value* per il test di Wilcoxon ($H_0 : \theta = 0$)

Algoritmi	f_1	f_2	f_3
MA-GA	0.000	0.000	0.000
MA-HC	0.000	0.002	0.000

Tabella 4.10: *p-value* per il test di Wilcoxon ($H_0 : \theta > 0$)

Algoritmi	f_1	f_2	f_3
MA-GA	1.000	0.000	0.000
MA-HC	1.000	0.001	0.000

4.4 Risultati

Le configurazioni migliori dei tre algoritmi sono state confrontate tra loro attraverso la comparazione a coppie del Test di Wilcoxon. Le comparazioni sono state eseguite rispettivamente tra l'Algoritmo Memetico e l'Algoritmo Genetico, e tra il Memetico e l'Hill Climbing. Il test è stato prima utilizzato con un ipotesi alternativa non-direzionale (vedi Tabella 4.9), per verificare la presenza di differenze significative di comportamento tra gli algoritmi, e poi con un'ipotesi alternativa direzionale (vedi Tabella 4.10), di tipo $H_1 : \theta < 0$. Il livello di significatività α è posto a 0.05.

Nel caso di ipotesi non direzionale i *p-value* sono minori di $\alpha = 0.05$, e quindi possiamo rifiutare l'ipotesi nulla a favore dell'ipotesi alternativa in quanto c'è una differenza significativa di comportamenti tra gli algoritmi confrontati.

Dalla Tabella 4.10 osserviamo che il *p-value* per la funzione della sfera è massimo per entrambi i confronti: questo ci porta ad accettare l'ipotesi nulla; infatti l'Algoritmo Memetico mostra prestazioni più scarse in questo caso rispetto alle altre due strategie.

Nelle funzioni di Rosenbrock e Rastrigin invece le performance migliori sono ottenute dall'Algoritmo Memetico. Le motivazioni dietro il risultato sulla funzione della sfera possono ricercarsi nella semplicità della funzione, poco adeguata alla complessità dell'Algoritmo Memetico. Inoltre il problema si mostra particolarmente compatibile con il metodo di ricerca dell'Hill Climbing che abbiamo definito *dinamico*: il minimo globale, trovandosi al centro dell'intervallo di ricerca, è individuabile con maggior precisione attraverso questa strategia in quanto il diametro degli intorni valutati si stringe sempre di più quando ci si avvicina al minimo.

Nelle Figure 4.4, 4.6 e 4.8 sono rappresentati i valori di fitness ottenuti degli algoritmi per i tre problemi di benchmark, mentre nelle Figure 4.5, 4.7 e 4.9 sono rappresentati i valori di fitness per gli algoritmi che ottengono performance migliori rispetto agli altri due.

Figura 4.4: Valori di fitness per i tre algoritmi sulla funzione della sfera

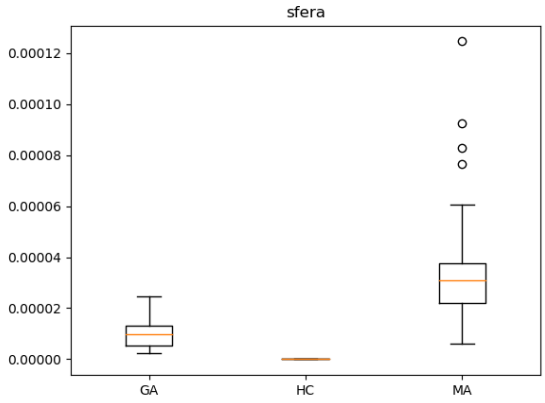


Figura 4.5: Valori di fitness per l'Hill Climbing sulla funzione della sfera

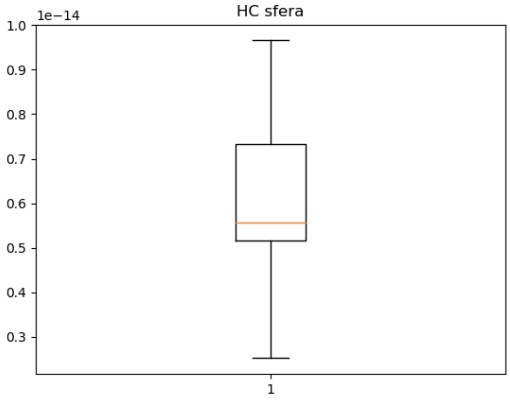


Figura 4.6: Valori di fitness per i tre algoritmi sulla funzione di Rosenbrock

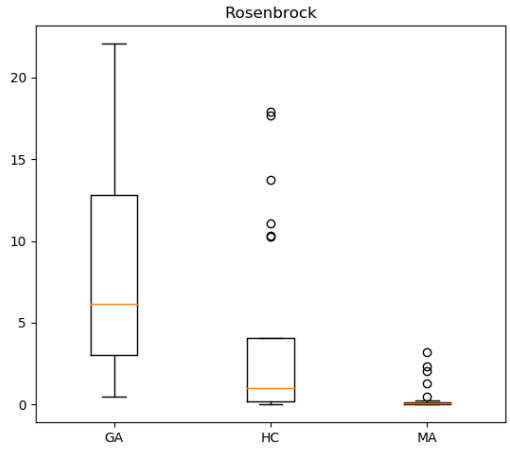


Figura 4.7: Valori di fitness per l'Algoritmo Memetico sulla funzione di Rosenbrock

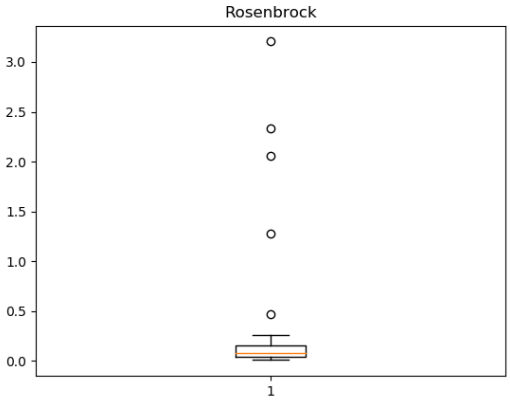


Figura 4.8: Valori di fitness per i tre algoritmi sulla funzione di Rastrigin

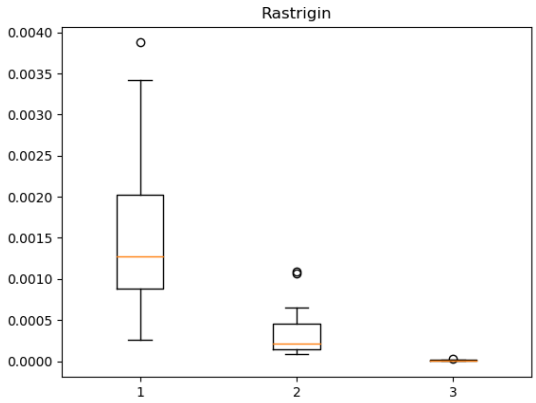
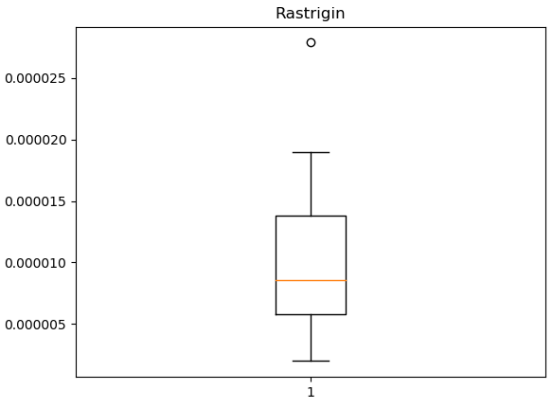


Figura 4.9: Valori di fitness per l'Algoritmo Memetico sulla funzione di Rastrigin



Conclusioni

In questo elaborato è stato implementato e valutato un Algoritmo Memetico, confrontando le sue prestazioni con quelle di altre due metaeuristiche: un Algoritmo Genetico e un algoritmo di Hill Climbing. Tutti gli Algoritmi sono stati implementati in Python, sfruttando principalmente la libreria evolutiva DEAP e arricchendola delle strutture necessarie alla realizzazione delle componenti memetiche.

Dopo aver implementato i tre algoritmi, si è proceduto a configurarne gli iperparametri: sono stati quindi definiti dei set di iperparametri e sono state valutate varie combinazioni per ogni algoritmo. Queste combinazioni sono state poi applicate a tre differenti problemi di benchmark: la funzione della sfera, la funzione di Rosenbrock e la funzione di Rastrigin. I risultati per ogni configurazione di un algoritmo e per ogni funzione sono stati confrontati attraverso il Test statistico di Friedman ed il test statistico post-hoc di Holm: attraverso questi test è stato possibile stabilire quale fosse la configurazione migliore degli iperparametri per ogni problema di benchmark, ovvero la configurazione che ottenesse le prestazioni migliori della maggioranza delle altre configurazioni.

I tre algoritmi, presi nelle loro configurazioni ottimali, sono stati confrontati tra loro attraverso una comparazione a coppie utilizzando il test di Wilcoxon: dalla sperimentazione è risultato che l'Algoritmo Memetico ha superato gli altri due algoritmi per due dei problemi di benchmark, ovvero quello di Rosenbrock e quello di Rastrigin, in accordo con le aspettative, dal momento che l'Algoritmo Memetico unisce i vantaggi della ricerca globale e di quella locale. Tuttavia l'Algoritmo Memetico ha ottenuto le prestazioni più scarse applicato alla funzione della sfera, probabilmente a causa dell'estrema semplicità della funzione non compatibile con la complessità di design dell'Algoritmo Memetico, e piuttosto adeguata alla tecnica di ricerca degli intorni *dinamica* definita per l'Hill Climbing.

Lavori futuri potrebbero ampliare la sperimentazione eseguita in questo elaborato utilizzando differenti funzioni di benchmark per le comparazioni tra gli algoritmi, come la funzione di Ackley o la funzione di Weierstrass, che possiedono numerosi minimi locali e possono mostrare in modo più esaustivo le potenzialità degli Algoritmi Memetici. Un'altra possibilità di sperimentazione futura potrebbe prevedere la parallelizzazione degli algoritmi, sfrut-

tando in questo modo i paradigmi di distribuzione implementati da DEAP che non sono stati utilizzati in questo lavoro di tesi. Numerose sono anche le applicazioni che si possono realizzare in campo fisico e industriale. Tra queste citiamo le applicazioni nel settore della simulazione ottica, in particolare delle nanotecnologie come il *laser writing* o la litografia a fascio elettronico, dove gli Algoritmi Memetici possono essere utilizzati per determinare i parametri geometrici ottimali per la fabbricazione di nanostrutture ottiche.

Bibliografia

[1] *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2005, 2-4 September 2005, Edinburgh, UK*. IEEE, 2005.

[2] Giovanni Acampora and Autilia Vitiello. Improving agent interoperability through a memetic ontology alignment: A comparative study. In *2012 IEEE International Conference on Fuzzy Systems*, pages 1–8. IEEE, 2012.

[3] Thomas Back. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

[4] François-Michel De Rainville, Félix-Antoine Fortin, Marc-André Gardner, Marc Parizeau, and Christian Gagné. Deap: A python framework for evolutionary algorithms. In *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*, pages 85–92, 2012.

[5] François-Michel De Rainville, Félix-Antoine Fortin, Marc-André Gardner, Marc Parizeau, and Christian Gagné. Deap: enabling nimbler evolutions. *ACM SIGEVOlution*, 6(2):17–26, 2014.

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

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

[8] Salvador García, Daniel Molina, Manuel Lozano, and Francisco Herrera. A study on the use of non-parametric tests for analyzing the evolutionary algorithmsâ behaviour: a case study on the cecâ2005 special session on real parameter optimization. *Journal of Heuristics*, 15(6):617, 2009.

[9] Jin-Kao Hao. Memetic algorithms in discrete optimization. In *Handbook of memetic algorithms*, pages 1–25. Springer, 2012.

[10] William Eugene Hart. *Adaptive global optimization with local search.* PhD thesis, University of California, San Diego, Department of Computer Science â, 1994.

[11] CA Hesse, JB Ofosu, and EN Nortey. Introduction to nonparametric statistical methods, 2017.

[12] Khalid Jebari, Mohammed Madiafi, and Abdelaziz Elmoujahid. Parent selection operators for genetic algorithms. *International Journal of Engineering*, 2, 2013.

[13] 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.

[14] Majdi Mafarja, Salwani Abdullah, and Najmeh S Jaddi. Fuzzy population-based meta-heuristic approaches for attribute reduction in rough set theory. 2015.

[15] Marcin Molga and Czesław Smutnicki. Test functions for optimization needs. *Test functions for optimization needs*, 101, 2005.

[16] Pablo Moscato and Carlos Cotta. A modern introduction to memetic algorithms. In *Handbook of metaheuristics*, pages 141–183. Springer, 2010.

[17] Ferrante Neri, Carlos Cotta, and Pablo Moscato. *Handbook of memetic algorithms*, volume 379. Springer, 2011.

[18] Quang Huy Nguyen, Yew-Soon Ong, and Natalio Krasnogor. A study on the design issues of memetic algorithm. In *2007 IEEE Congress on Evolutionary Computation*, pages 2390–2397. IEEE, 2007.

[19] 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.

[20] Hans-Paul Schwefel. Advantages (and disadvantages) of evolutionary computation over other approaches. *Evolutionary computation*, 1:20–22, 2000.

[21] Mohammad Shehab. A hybrid method based on cuckoo search algorithm for global optimization problems. *Journal of Information and Communication Technology*, 17(3):469–491, 2020.

[22] Clinton Sheppard. *Genetic algorithms with python.* Smashwords Edition, 2017.

[23] David J Sheskin. *Handbook of parametric and nonparametric statistical procedures*. Chapman and Hall/CRC, 2003.

[24] 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.

[25] Andrea Tettamanzi. *Algoritmi evolutivi per l'ottimizzazione*. PhD thesis, Ph. D. dissertation, Univ. Milan, Milan, Italy, 1995.

[26] Andrea Tettamanzi. Algoritmi evolutivi: concetti ed applicazioni. 2005.

[27] Jerrold H Zar. *Biostatistical analysis*. Pearson Education India, 1999.