GreenTrails - Modulo AI

Roberta Galluzzo, Gabriele Di Stefano

26 marzo 2025

Team:	C03 GreenTrails
Progetto:	Progetto combinato IS/FIA 2023-2024
Repository:	https://github.com/gab-ds/fia

Indice

1 Definizione del problema

1.1 Introduzione

Negli ultimi anni, l'ecosostenibilità è diventata una priorità, poiché il mondo, in rapida crescita, deve prestare maggiore attenzione all'ambiente. In questo contesto, l'applicazione web "GreenTrails" si rivolge al settore turistico, con l'obiettivo di facilitare l'organizzazione di viaggi sostenibili, tenendo conto delle sfide che gli utenti affrontano in questo ambito. L'obiettivo di "GreenTrails" è permettere agli utenti di pianificare facilmente e rapidamente viaggi ecosostenibili, aiutandoli a individuare mete, strutture e attività adatte.

L'applicazione, dunque, mira a:

- Facilitare la ricerca di strutture eco-friendly;
- Generare automaticamente itinerari in base alle preferenze personali degli utenti;
- Velocizzare il contatto con le strutture selezionate.

1.2 Obiettivi

GreenTrails prevede, appunto, la **generazione automatica di itinerari multi-attività in base alle preferenze specificate dall'utente**, attraverso l'utilizzo di un algoritmo di intelligenza artificiale. Le preferenze sono così composte:

- **Località preferita**: una scelta tra *Mare*, *Montagna*, *Città*, oppure *Nessuna preferenza*.
- **Tipologia di alloggio preferita**: uno tra *Hotel*, *Bed & Breakfast*, *Villaggio turistico*, *Ostello* oppure *Nessuna preferenza*.
- **Preferenze alimentari**: *Vegan*, *Vegetarian*, *Gluten-free* o *Nessuna preferenza*.
- **Tipologia di attività preferita**: una categoria a scelta tra *All'aperto*, *Visite storico-culturali*, *Gastronomia* oppure *Nessuna preferenza*.
- **Animale domestico**: esprime la preferenza di portare con sé un animale da compagnia, con una semplice scelta tra *Sì* oppure *No*.
- **Budget**: *Basso*, *Medio*, *Alto* oppure *Flessibile*.
- **Souvenir locali**: esprime l'interesse nell'acquisto di prodotti artigianali locali, con una semplice scelta tra *Sì* oppure *No*.
- **Stagione preferita**: *Autunno-inverno*, *Primavera-estate* o *Nessuna preferenza*.

Inoltre, tale algoritmo dovrà rispettare il vincolo di dover generare un percorso valido entro il tempo massimo prestabilito dal sistema (**2 secondi**, come definito dal System Design Goal "*DG_12 - Tempo di risposta*").

1.2.1 Ulteriori considerazioni

In mancanza di ulteriori specifiche, che avrebbero potuto maggiormente impostare il problema, si è deciso di imporre dei vincoli aggiuntivi per facilitare la fase di formulazione:

- il modulo di intelligenza artificiale non cercherà su tutto il database dell'applicazione, ma soltanto su un subset fornito dal sistema stesso (in altre parole, **consideriamo soltanto le attività di una città o una specifica area** fornite dal sistema);
- il percorso dovrà **terminare con una e una sola struttura ricettiva**;
- il percorso dovrà **contenere almeno due attività turistiche, fino ad un massimo di quattro**;
- il percorso deve avere una **lunghezza massima di 5 km**.
- le attività del percorso **non possono essere ripetute**.

1.3 Specifica PEAS

- **Performance**: la misura di prestazione dell'agente, in questo caso, si basa sulla pertinenza del percorso fornito con le preferenze dell'utente e sulla sua distanza massima tra le attività.
- **Environment**: l'ambiente di riferimento è l'applicazione web di GreenTrails, la quale comunica direttamente con l'agente.
- **Actuators**: sono le soluzioni restituite all'applicazione web.
- **Sensors**: sono i dati in input (ovvero, la lista delle attività disponibili e le preferenze dell'utente) forniti dall'applicazione web.

1.4 Caratteristiche dell'ambiente

L'ambiente in cui opera l'agente ha le seguenti caratteristiche:

- è **completamente osservabile**, in quanto prende in considerazione tutte le possibili attività fornite dal sistema;
- è **stocastico**, poiché lo stato varia indipendentemente dalle azioni dell'agente;
- è **episodico**, dal momento che la soluzione scelta dall'agente varia in base alla lista di attività e alle preferenze dell'utente (dunque, ad ogni episodio);
- è **statico**, in quanto non subisce variazioni mentre l'agente sta valutando le possibili soluzioni;
- è **discreto**, perché è possibile identificare un numero finito di scelte (tutte le possibili combinazioni di attività e strutture ricettive);
- è **ad agente singolo**, dato che consente la presenza di un unico agente.

1.5 Analisi del problema

Considerando la natura del problema, che è di **ottimizzazione**, possiamo optare per l'utilizzo di un **algoritmo genetico**. Tale scelta è condizionata anche dalla **necessità progettuale** (per il progetto di Ingegneria del Software) di utilizzare un algoritmo genetico per la generazione degli itinerari.
Formalizzando:

- **Spazio degli stati**: tutte le possibili permutazioni delle attività contenenti almeno due attività turistiche ed esattamente una struttura ricettiva al termine, con distanza massima complessiva di 5 km;
- **Stato iniziale**: percorso vuoto;
- **Azioni**: aggiungere un'attività turistica (fino ad un massimo di 4), o una struttura ricettiva (se è al termine del percorso);
- **Modello di transizione**: restituisce un percorso contenente l'attività aggiunta alla fine dello stesso;
- **Test obiettivo**: il percorso deve contenere attività basate sulle preferenze utente, e deve essere realizzato entro 2 secondi;
- **Costo di cammino**: distanza dall'attività successiva, in km.

2 Soluzione del problema

2.1 Set di dati utilizzati

Per ciò che concerne il reperimento dei dati, sono disponibili due opzioni:

- la prima, che consiste nell'**utilizzare un dataset pubblicamente disponibile**;
- la seconda, che consiste nella **generazione di dati artificiali**.

Abbiamo optato per la generazione di dati artificiali, in quanto è irrilevante avere dati veritieri in questo contesto: **il focus del problema è quello di trovare dei percorsi affini in base alle esigenze dell'utente**, ed è perciò non necessario utilizzare informazioni di strutture e utenti reali.
Ci siamo avvalsi di *due script aggiuntivi* per la generazione casuale dei dataset di visitatori e attività.

2.1.1 Formattazione dei dati

I dati sono stati organizzati in file con formato CSV, in modo da renderli facilmente leggibili e interpretabili sia da un software, sia da una persona che li esamini manualmente.
Ecco la composizione del set di dati dei visitatori, con una riga di esempio:

ID	1
Località preferita	Città
Tipologia di alloggio preferita	Villaggio turistico
Preferenze alimentari	Vegetarian
Tipologia di attività preferita	Visite storico-culturali
Animali domestici	Sì
Budget	Medio
Souvenir locali	Sì
Stagione preferita	Primavera-estate

Ecco la composizione del set di dati delle attività, con una riga di esempio:

ID	1
Latitudine	45.66591297292027
Longitudine	10.475487956720567
Alloggio	Sì
Tipologia	Villaggio turistico
Prezzo	478
Cibo	Vegan
Animali domestici	Sì
Stagione rilevante	Nessuna informazione
Souvenir locali	Sì
Località	Montagna

2.2 Tecnologie utilizzate

Come linguaggio di programmazione, si è scelto di utilizzare **Python 3**. Le librerie utilizzate, invece, sono molteplici; di seguito, quelle utilizzate dal programma agente:

- **FastAPI**, per la realizzazione di un'API HTTP attraverso cui interagire con il sistema;
- **GeoPy**, per il corretto calcolo delle distanze geografiche;
- **SciPy**, per poter raggruppare i punti in un K-D Tree;
- **NumPy**, per velocizzare i calcoli, piuttosto che utilizzare le funzioni built-in di Python.

Ulteriori librerie utilizzate, che non fanno direttamente parte dell'agente, sono:

- **PyCristoforo**, utilizzato per la corretta generazione di coordinate casuali in Italia, da utilizzare nel set di dati delle attività;
- **Requests**, utilizzato per lo script di mock del sistema che dovrà interagire con l'endpoint HTTP esposto dall'agente;
- **Pandas**, utilizzato per lo script di mock del sistema per caricare i dati dai rispettivi dataset di attività e visitatori;
- **Matplotlib**, utilizzato per la generazione dei grafici.

2.3 Specifica dell'algoritmo genetico

2.3.1 Codifica degli individui

La nostra decisione è stata quella di determinare ogni individuo come un array a taglia fissa, di dimensione 5, contenente gli ID numerici delle attività, in cui l'ultimo elemento è sempre una struttura ricettiva. Per rappresentare eventuali percorsi con un numero di attività turistiche inferiori a 4, si è inoltre deciso di utilizzare un valore sentinella al posto dell'ID, con valore 0.

2.3.2 Funzione di fitness

La funzione di fitness *f(x)* è così definita:

$$f(x) = w_{distanza} * \sum_{i=1}^{n-1} f_{distanza}(x_i, x_{i+1}) + w_{attivita} * \sum_{i=1}^{n} f_{attivita}(x_i) + w_{presente} * \sum_{i=1}^{m} presente(x_i)$$

I suoi componenti sono:

- $w_{distanza}$: il peso della distanza nel fitness totale dell'individuo;
- $f_{distanza}$: il valore di fitness dipendente dalla distanza tra due punti, calcolato sottraendo a 5000 la distanza in metri ($5000 - distanza(x, y)$) e troncando il risultato ad un intero. Restituisce valore 0 se troppo distanti tra loro;
- n: è il numero di elementi (nel nostro caso, 5) che compongono ogni individuo;
- $w_{attivita}$: il peso della corrispondenza delle preferenze nel fitness totale;
- $f_{attivita}$: il valore di fitness dipendente dalla corrispondenza tra le preferenze utente e l'attività, dando un valore di reward di 2 per ogni corrispondenza perfetta (o nessuna preferenza espressa). Casi particolari sono la preferenza di stagione (nel caso in cui la struttura non abbia una stagione rilevante, ma l'utente abbia espresso una preferenza, la reward è dimezzata ad 1) e il budget (nel caso in cui non ci sia una corrispondenza perfetta, la reward sarà compresa tra -4 e +4, in base alla differenza tra budget preferito e prezzo effettivo). Restituisce la somma di tutte le reward;
- $w_{presente}$: il peso della presenza delle prime m attività nel percorso;
- $presente(x_i)$: restituisce 1 se l'attività non è vuota, 0 altrimenti;
- m: è il numero degli elementi presi in considerazione per la reward di presenza (nel nostro caso, 2).

La reward di presenza è stata aggiunta in seguito, in modo da **valorizzare percorsi contenenti più attività**: senza di essa, l'algoritmo tenderebbe a privilegiare soluzioni di lunghezza minore, a scapito della completezza del percorso, portando a itinerari con al più 3 attività.
La funzione di fitness **restituisce valore 0 nel caso in cui i vincoli non siano rispettati** (attività troppo distanti o ripetute, permutazioni non accettabili).

2.3.3 Inizializzazione

Per generare una popolazione iniziale con soluzioni accettabili, si è pensato di utilizzare il seguente algoritmo:

```
1. Separa l'array delle attività in un array di sole strutture ricettive
e un array di sole attività turistiche.
2. Costruisci due K-D Tree, uno contenente le coordinate delle sole
attività turistiche, l'altro delle sole strutture ricettive.
3. Poni la distanza totale del percorso a 0.
4. Prendi casualmente una struttura ricettiva.
5. Estrai dal K-D Tree delle attività turistiche al massimo 5 attività
più vicine all'attività di riferimento.
6. Scarta le attività turistiche che distano più di 5 km dall'attività
di riferimento, escludendo anche le attività già presenti nell'individuo,
lasciando soltanto quelle ammissibili.
7a. Se il numero di attività turistiche ammissibili è inferiore a
2, scarta la struttura ricettiva e torna al punto 3.
7b. Altrimenti, seleziona la prima attività più vicina, aggiungendo
alla distanza totale la distanza tra le due attività.
8a. Se la distanza totale supera i 5 km, e le attività presenti finora
nell'individuo sono almeno 3, aggiungi l'individuo alla popolazione
e vai al punto 10.
8b. Altrimenti, se il loro numero non supera i 2 elementi, scarta
la struttura ricettiva e torna al punto 3.
9. Ritorna al punto 4, utilizzando però l'ultima attività aggiunta
come riferimento, finché non abbiamo al più 5 elementi nell'individuo.
Se l'individuo ha esattamente 5 elementi, allora aggiungilo alla popolazione.
10. Ritorna al punto 3 finché il numero di individui è minore di n.
```

Nell'algoritmo sopra specificato, si consideri n la dimensione della popolazione.

2.3.4 Selezione

L'algoritmo di selezione considerato è il Roulette Wheel con Elitismo. Esso consiste nel preservare k individui promettenti e avviare una selezione casuale con probabilità per i restanti $n - k$ individui.
In questo modo, garantiamo che ogni generazione non sia peggiore della precedente, esponendoci, però, maggiormente al rischio di convergenza prematura: una mitigazione messa in atto è quella di ridurre il valore di fitness dei due elementi dell'élite del 20%, per garantire una possibilità maggiore anche per gli altri individui durante le prossime selezioni. Di seguito, la descrizione del procedimento.

1. Calcola il valore di fitness di ogni individuo della popolazione.
2. Seleziona i primi k individui con valore di fitness maggiore e copiali nella nuova generazione.
3. Riduci il valore di fitness calcolato dei due individui scelti del 20%.
4. Calcola il valore complessivo di fitness per l'intera generazione.
5. Calcola le probabilità per ogni individuo applicando la seguente formula:

$$\frac{fitness_{individuo}}{fitness_{totale}}$$

6. Avvia una selezione casuale dei restanti $n - k$ individui della nuova generazione, utilizzando le probabilità calcolate per ogni individuo.

2.3.5 Crossover

La strategia adottata è il Single Point Crossover, dividendo l'array al terzo elemento: in questo modo, avremo che i primi 2 elementi proverranno dal primo genitore, mentre i restanti 3 dal secondo. Per evitare individui non ammissibili, calcoliamo il fitness dei due figli ottenuti: se questo valore, per almeno uno dei due, è pari a zero, allora il crossover non si effettua.

2.3.6 Mutazione

L'approccio utilizzato per la mutazione è ibrido, in quanto si è deciso di utilizzare sia il Random Resetting - che consiste nella selezione di un gene in modo casuale e la sua conseguente sostituzione con un altro gene, a patto che questo dia vita a un individuo ammissibile - sia lo Scrambling - che prende un sottoinsieme di geni dell'individuo e li riordina casualmente.
Il Random Resetting viene effettuato senza restrizioni su qualsiasi attività, a patto che l'attività successiva a quest'ultima non sia pari a 0 (evitando individui con percorsi non ammissibili, es. $[1, 0, 2, 3, 4]$).
Lo Scrambling, invece, seleziona un subset di sole attività turistiche (evitando di spostare la struttura ricettiva), escludendo le attività nulle.

Infine, si è scelto di effettuare le mutazioni sugli individui adottando un approccio adattivo: al raggiungimento della convergenza, la probabilità di mutazione raddoppia ad ogni generazione, fino ad un massimo di 1.

2.3.7 Stopping condition

La principale condizione di arresto, nel nostro caso, corrisponde al limite massimo di tempo pari a **2 secondi**. Ulteriori stopping condition, poi, sono:

- **la mancanza di miglioramenti per 50 generazioni**;
- **la presenza di una popolazione composta da individui non ammissibili**;
- **il raggiungimento della convergenza**.

3 Conclusione

3.1 Esecuzione

Per testare l'agente, è stato deciso di **simulare richieste API tramite uno script** apposito, che replicherà il comportamento del sistema in ambiente reale **fornendo dati simili a quelli che l'algoritmo genetico riceverebbe in produzione**. In particolare, lo script seleziona casualmente un visitatore dal dataset pre-generato e, a sua volta, sceglie in modo casuale una struttura ricettiva dal dataset delle attività, includendo anche le attività nei dintorni, entro un raggio di circa 20 km. Inoltre, per garantire risultati variegati, **vengono considerati soltanto i gruppi di attività composti da almeno 30 elementi**, dove ciascun gruppo prevede almeno 10 strutture ricettive e 10 attività turistiche.

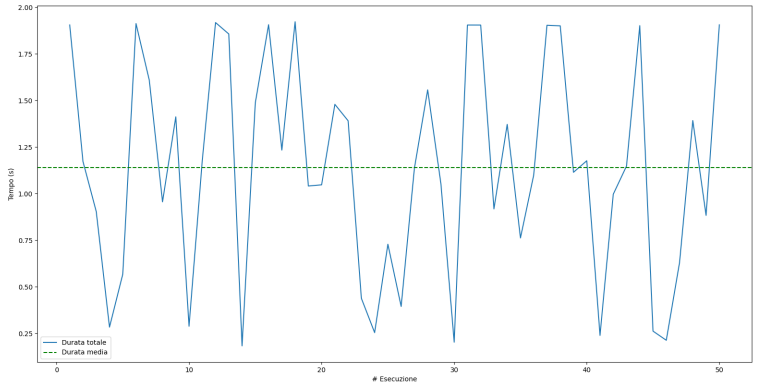
3.1.1 Parametri iniziali

Dimensione della popolazione: 10.
Peso delle corrispondenze tra preferenze e attività: 200.
Peso della distanza: 1.
Peso della presenza: 1000.
Dimensione dell'élite: 2.
Probabilità di crossover: 50%.
Probabilità di mutazione: 10%, con possibilità di **raddoppio in caso di convergenza prematura**, e **dimezzamento in caso di individui troppo vari**.

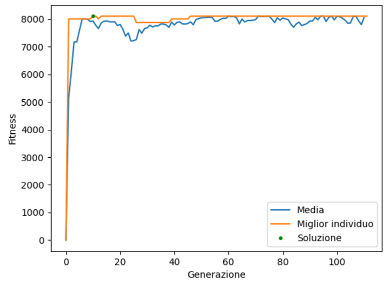
3.2 Risultati

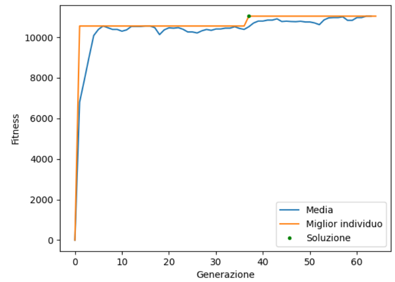
Analizzando i risultati ottenuti, si può notare che **la durata media di esecuzione è di circa 1.25 secondi**, e che **la soluzione migliore viene spesso trovata entro le prime 50 generazioni**: oltre tale limite si troveranno, in genere, solo miglioramenti marginali.
La buona performance è attribuibile a due fattori principali: un **efficace metodo di generazione della popolazione iniziale**, combinato con la **natura del dataset, composto da punti molto dispersi tra loro**, in quanto generati casualmente e non secondo criteri di densità urbana.
Non meno importante, poi, è l'**ottimizzazione adottata per il calcolo del valore di fitness**: attraverso la **memoizzazione**, a parità di tempi di esecuzione, l'algoritmo genetico **riesce a esplorare uno spazio degli stati più ampio**, poiché la computazione viene effettuata solo la prima volta per ogni individuo.
Infine, **la popolazione tende a convergere molto rapidamente**, il che ha suggerito l'implementazione di un **meccanismo di mutazione adattivo**, al fine di diversificare la popolazione molto rapidamente: **in caso di fallimento, ciò indica l'assenza di altre soluzioni migliori**, consentendo di terminare l'esecuzione anche entro le prime 20 generazioni.

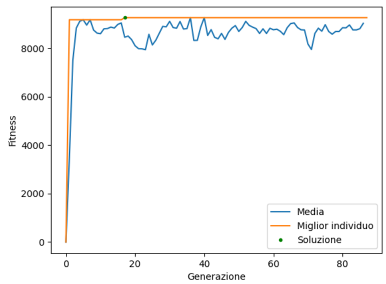
Un'analisi dei tempi di esecuzione è riportata qui sotto.

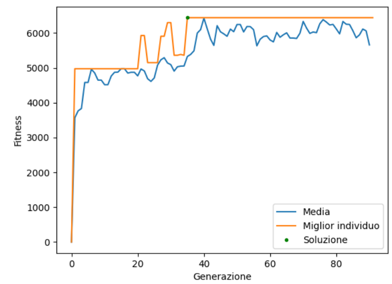


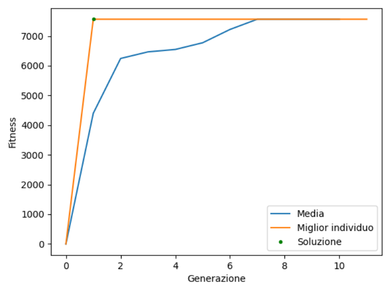
Di seguito, alcuni risultati presi a campione.

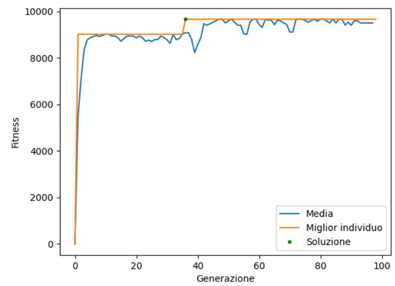

Fitness
8000
7000
6000
5000
4000
3000
2000
1000
0
0
20
40
60
80
100
Generazione
Media
Miglior individuo
Soluzione

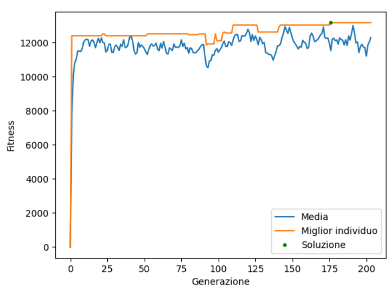

Fitness
10000
8000
6000
4000
2000
0
0
10
20
30
40
50
60
Generazione
Media
Miglior individuo
Soluzione

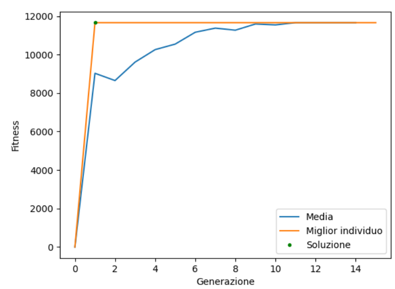

Fitness
8000
6000
4000
2000
0
0
20
40
60
80
Generazione
Media
Miglior individuo
Soluzione

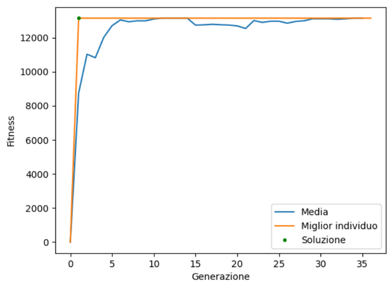

Fitness
6000
5000
4000
3000
2000
1000
0
0
20
40
60
80
Generazione
Media
Miglior individuo
Soluzione

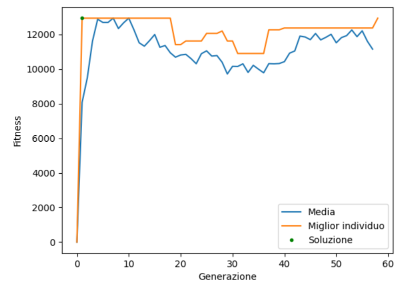

Fitness
7000
6000
5000
4000
3000
2000
1000
0
0
2
4
6
8
10
Generazione
Media
Miglior individuo
Soluzione

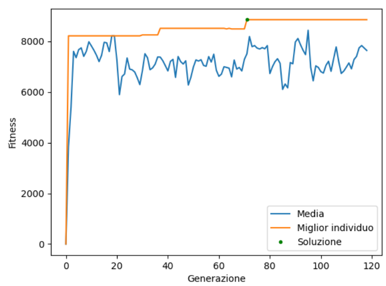

Fitness
10000
8000
6000
4000
2000
0
0
20
40
60
80
100
Generazione
Media
Miglior individuo
Soluzione

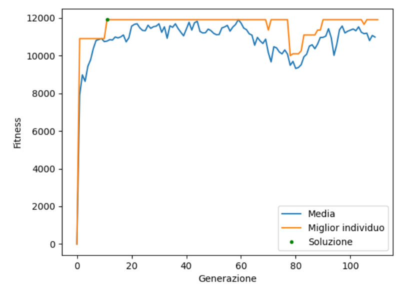

Fitness
12000
10000
8000
6000
4000
2000
0
0
25
50
75
100
125
150
175
200
Generazione
Media
Miglior individuo
Soluzione

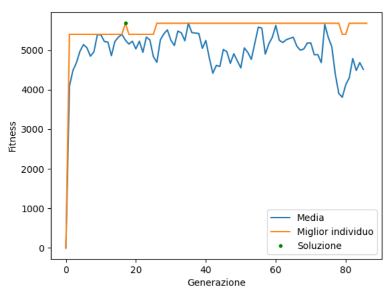

Fitness
12000
10000
8000
6000
4000
2000
0
0
2
4
6
8
10
12
14
Generazione
Media
Miglior individuo
Soluzione

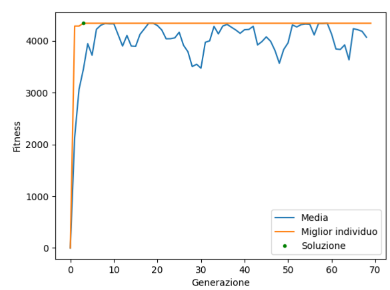

Fitness
Generazione
Media
Miglior individuo
Soluzione

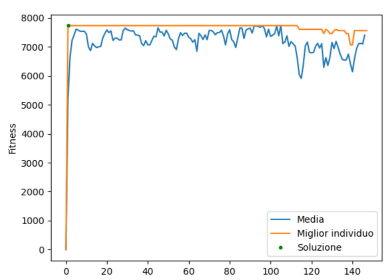

Fitness
Generazione
Media
Miglior individuo
Soluzione

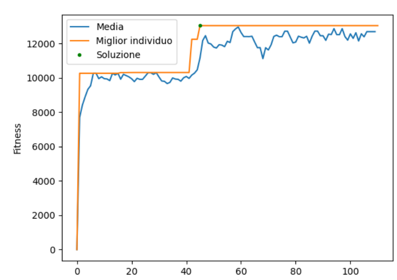

Fitness
Generazione
Media
Miglior individuo
Soluzione

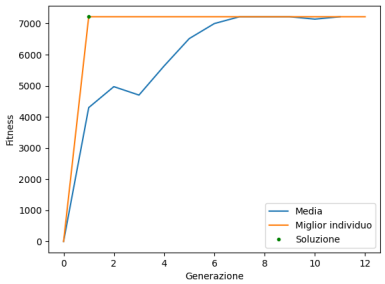

Fitness
Generazione
Media
Miglior individuo
Soluzione

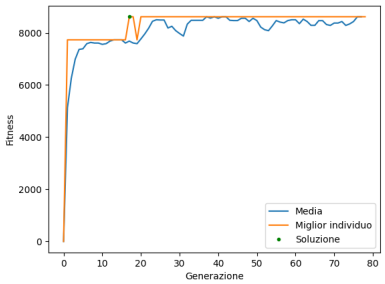

Fitness
Generazione
Media
Miglior individuo
Soluzione

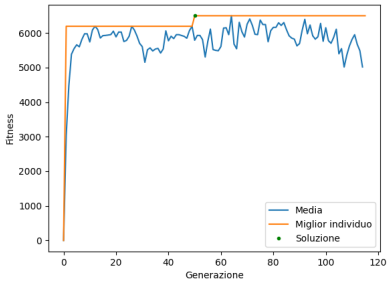

Fitness
Generazione
Media
Miglior individuo
Soluzione

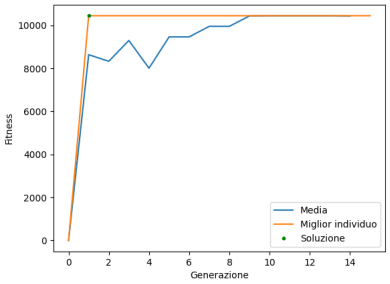

Fitness
Generazione
Media
Miglior individuo
Soluzione


Fitness
Generazione
Media
Miglior individuo
Soluzione


Fitness
Generazione
Media
Miglior individuo
Soluzione


Fitness
Generazione
Media
Miglior individuo
Soluzione


Fitness
Generazione
Media
Miglior individuo
Soluzione


Fitness
Generazione
Media
Miglior individuo
Soluzione

3.3 Considerazioni finali

Seppur fornisca un'ottima approssimazione, l'algoritmo è decisamente migliorabile sotto diversi aspetti: in particolare, **la presenza di dati veritieri consentirebbe lo studio più approfondito** di eventuali problematiche non riscontrate con un dataset artificiale.
Infine, si potrebbe **considerare l'utilizzo di strutture dati create su misura, da sostituire ai K-D Tree**, per una più rapida e accurata individuazione delle strutture adiacenti.

4 Glossario

- **Problema di ottimizzazione**: problema il cui obiettivo è quello di ricercare il miglior valore possibile per una funzione obiettivo, soggetta a vincoli.
- **Algoritmo genetico**: problema di ottimizzazione che simula il processo evolutivo al fine di trovare soluzioni ottimali a problemi complessi.I suoi individui, ovvero la popolazione iniziale, attraverso operazioni di selezione, incrocio e mutazione, genereranno individui evoluti, che saranno sempre più vicini alla soluzione ottima.
- **Python**: linguaggio di programmazione ad alto livello, che si presta ad un semplice utilizzo per via della vicinanza al linguaggio naturale.
- **Large Language Model**: modello di deep learning, addestrato su grandi quantitativi di dati testuali, con l'obiettivo di apprendere e generare linguaggio naturale.
- **K-D Tree**: struttura dati gerarchica per organizzare punti in uno spazio k-dimensionale euclideo. A ogni livello dell'albero, lo spazio viene suddiviso alternando sistematicamente le k dimensioni. Questi alberi sono utilizzati per l'organizzazione e la ricerca di dati multidimensionali. Sebbene le coordinate geografiche non rappresentino uno spazio euclideo, per distanze relativamente brevi sulla superficie terrestre gli effetti della curvatura possono essere trascurati, approssimando sufficientemente bene le distanze calcolate con le coordinate geografiche a quelle di uno spazio euclideo bidimensionale.
- **Funzione di fitness**: funzione di valutazione dei dati o di adattamento della soluzione candidata al problema in via di risoluzione. Essa assegna un punteggio ad ogni individuo della popolazione scelta, in base al livello di soddisfazione dell'obiettivo. Infatti, più alto è il punteggio di fitness di una soluzione, maggiore è la probabilità che essa venga selezionata e utilizzata per generare nuove soluzioni che si avvicineranno sempre di più alla soluzione ottimale.
- **Memoizzazione**: tecnica che consiste nel salvataggio in memoria dei valori restituiti da una funzione, in maniera tale da ricavarli facilmente per riutilizzi successivi, evitando di ricalcolare i risultati.