



UNIVERSITÀ DEGLI STUDI DI CATANIA  
DIPARTIMENTO DI MATEMATICA E INFORMATICA  
CORSO DI LAUREA MAGISTRALE IN INFORMATICA

# **Uso del Machine Learning per Ottimizzare il Processo di Ricerca di un Algoritmo Evolutivo**

Simone Basile

RELATORE  
Prof. Mario Francesco Pavone

CORRELATORI  
Dott. Francesco Zito

Anno Accademico 2022/23

Spero che questo sforzo ti abbia reso felice,  
tanti auguri di  
buon compleanno mamma!

---

# Indice

<b>Indice</b>	<b>1</b>
<b>Sommario</b>	<b>2</b>
<b>1 Introduzione</b>	<b>3</b>
<b>2 Feedback Vertex Set</b>	<b>5</b>
2.1 Definizione . . . . .	6
<b>3 Algoritmi evolutivi</b>	<b>8</b>
3.1 Introduzione agli algoritmi evolutivi . . . . .	8
3.2 Fasi di un algoritmo evolutivo . . . . .	9
3.3 Famiglie di algoritmi evolutivi . . . . .	10
<b>4 Evolutive Algorithm with Machine Learning (EAML)</b>	<b>13</b>
4.1 Rappresentazione delle soluzioni . . . . .	13
4.1.1 Calcolo della fitness di una soluzione . . . . .	14
4.1.2 Criterio di validità di una soluzione e gestione della penalità . . . . .	14
4.2 Generazione della popolazione iniziale . . . . .	16
4.3 Crossover . . . . .	17
4.4 Mutazione . . . . .	18
4.4.1 Operatori di mutazione Pre-ML . . . . .	18
4.4.2 Operatori di mutazione Post-ML . . . . .	19
4.5 Sostituzione . . . . .	24
4.6 Ricerca locale . . . . .	25
<b>5 Risultati</b>	<b>27</b>
5.1 Parametri di funzionamento dell'algoritmo . . . . .	27
5.1.1 Parametri di mutazione Pre-ML . . . . .	28
5.1.2 Parametri di mutazione Post-ML . . . . .	29
5.2 Analisi dei risultati . . . . .	30
<b>Conclusioni</b>	<b>34</b>
<b>Bibliografia</b>	<b>36</b>

---

# Sommario

La presente tesi propone un approccio basato su un algoritmo evolutivo per la risoluzione del problema del Minimum Weighted Feedback Vertex Set. Dopo una chiara introduzione ai concetti fondamentali e alle peculiarità del problema, i capitoli successivi si concentrano sulla progettazione e l'implementazione dell'algoritmo proposto. Di particolare importanza nell'ambito di questo lavoro è l'implementazione del metodo di Local Search e l'applicazione di un algoritmo di Machine Learning specificamente progettato per calcolare dinamicamente la soglia mutativa per ogni generazione. Queste innovazioni hanno permesso di apportare miglioramenti significativi ai risultati ottenuti. Infine, i risultati ottenuti vengono confrontati con quelli forniti dall'algoritmo Hybrid-IA, per valutarne l'efficacia relativa.

---

# 1

## Introduzione

Questa tesi mette in luce un'innovativa strategia basata su un algoritmo evolutivo per la risoluzione del problema del Minimum Weighted Feedback Vertex Set. L'importanza di affrontare questo problema risiede nel suo significativo impatto sia nel dominio della teoria dei grafi che nelle applicazioni pratiche. Dal punto di vista teorico, il problema riveste un ruolo centrale nell'informatica e nella matematica. Dal punto di vista pratico, la sua risoluzione ha importanti ricadute in vari settori, quali l'ottimizzazione di reti, la bioinformatica e altri campi che richiedono un'efficiente gestione e interpretazione delle strutture di dati complesse.

La scelta di adottare un approccio evolutivo piuttosto che un metodo tradizionale è stata dettata dalla natura stessa del problema, caratterizzato da un ampio spazio delle soluzioni e potenziali minimi locali. Gli algoritmi evolutivi, grazie alla loro capacità di adattamento e miglioramento iterativo, offrono un'efficiente strategia di ricerca e ottimizzazione per questo tipo di sfide.

Un ulteriore elemento di innovazione di questa tesi è l'integrazione di tecniche di Machine Learning per il calcolo dinamico della soglia di mutazione applicabile a ciascuna generazione. Questa implementazione ha permesso di introdurre un elemento di dinamicità nell'applicazione della mutazione prevista dagli algoritmi evolutivi, consentendo una più ampia e profonda esplorazione dello spazio delle soluzioni, rispetto ad un approccio mutativo statico.

Infine, il percorso di sviluppo e implementazione dell'algoritmo proposto è stato caratterizzato da un rigoroso processo di verifica e confronto. I risultati ottenuti sono stati costantemente paragonati con quelli forniti dall'algoritmo Hybrid-IA<sup>[3][4]</sup>, al

fine di monitorare e perfezionare continuamente l'efficacia e l'efficienza dell'approccio proposto.

Ogni discrepanza tra i risultati ottenuti e quelli di Hybrid-IA è stata attentamente analizzata. In base alle discrepanze riscontrate, è stata intrapresa una fase di affinamento dell'algoritmo. Questo processo di affinamento ha seguito inizialmente un approccio parametrico, con l'obiettivo di ottimizzare i parametri dell'algoritmo per migliorare i risultati.

Successivamente, si è passati a un approccio implementativo, con l'introduzione di due significative innovazioni: un originale approccio di Local Search e l'implementazione di un metodo di Machine Learning. Quest'ultimo è stato specificamente introdotto per conferire dinamicità alla fase di mutazione dell'algoritmo evolutivo.

L'introduzione di questi nuovi approcci ha avuto un impatto notevole sui risultati finali, consentendo un netto miglioramento rispetto all'implementazione originale dell'algoritmo. La combinazione di un approccio deterministico, introdotto con la local search, e un approccio dinamico, dato dal machine learning, ha rappresentato un punto di svolta per la risoluzione del problema.

Il contributo essenziale dell'approccio dinamico introdotto dal machine learning è ulteriormente sottolineato dai risultati riportati alla fine della tesi, i quali vengono confrontati con quelli ottenuti dalla versione con approccio statito per per l'applicazione della mutazione. I risultati indicano chiaramente la superiorità dell'approccio dinamico, evidenziando l'efficacia del machine learning nell'ottimizzazione degli algoritmi evolutivi.

In conclusione, la presente tesi contribuisce al campo del Minimum Weighted Feedback Vertex Set, dimostrando come un approccio progressivo e metodico di ottimizzazione dell'algoritmo possa portare a risultati significativamente migliori. Inoltre, fornisce un nuovo strumento per affrontare problemi di ottimizzazione di questa complessità, con l'obiettivo di stimolare ulteriori ricerche in questo settore.

---

# 2

## Feedback Vertex Set

Il problema del *Feedback Vertex Set* (*FVS*) si occupa di individuare l'insieme minimo di vertici in un grafo, tale che la loro eliminazione comporti la rimozione di tutti i cicli presenti. In altre parole, l'intento è quello di trovare il sottoinsieme più piccolo di vertici che, una volta rimosso, renda il grafo aciclico.

Il problema diventa più complesso quando si considera il *Minimum Weighted Feedback Vertex Set*. In questa variante, a ogni vertice è assegnato un specifico *peso* o *costo*, e l'obiettivo diventa non solo quello di rendere il grafo aciclico, ma anche di minimizzare la somma totale dei pesi dei vertici eliminati.

È importante notare che questo problema è classificato come **NP-hard**, il che implica l'inesistenza di un algoritmo di complessità polinomiale capace di risolverlo efficacemente per tutti i possibili input. Di conseguenza, la ricerca di algoritmi euristici efficaci per questo problema rappresenta un'area di studio significativamente attiva nel campo dell'informatica.

Questo problema ha una vasta gamma di applicazioni pratiche in diversi settori. Ad esempio, nel campo delle **reti di comunicazione**<sup>[12]</sup>, la presenza di cicli può compromettere la trasmissione dei dati, pertanto identificare un FVS ottimale permette di individuare un insieme di nodi critici da rimuovere al fine di garantire una rete priva di cicli. Questa soluzione può migliorare notevolmente l'affidabilità e l'efficienza delle comunicazioni. Allo stesso modo, nel **settore dei circuiti integrati**, la presenza di cicli può causare ritardi dei segnali e un consumo energetico eccessivo. Pertanto, applicare il problema del FVS consente di identificare i nodi responsabili dei cicli e apportare modifiche per eliminarli, contribuendo a migliorare le prestazioni e l'efficienza dei circuiti integrati.

Nel contesto dell'**analisi dei dati** e dei **social network**<sup>[11]</sup>, il Feedback Vertex Set può essere utilizzato per individuare gli insiemi di nodi più influenti o critici. Ad esempio, nell'ambito del riconoscimento delle comunità, identificare un Feedback Vertex Set aiuta a individuare gruppi di utenti fortemente interconnessi che giocano un ruolo cruciale nella diffusione delle informazioni all'interno della rete. Inoltre, nell'ottimizzazione dei percorsi dei **sistemi di trasporto**, identificare cicli all'interno di una rete stradale o di trasporto pubblico che possono causare congestione e ritardi. L'applicazione del Feedback Vertex Set permette di individuare i punti critici della rete e apportare modifiche per migliorare la fluidità del traffico e l'efficienza dei percorsi.

Infine, nel campo della **bioinformatica**<sup>[8]</sup>, il Feedback Vertex Set viene impiegato nell'analisi di reti di interazione proteina-proteina o reti di regolazione genica. Identificare un insieme di nodi critici consente di comprendere le dinamiche delle interazioni tra proteine o la regolazione genica, fornendo informazioni importanti sulla struttura e sulla funzione dei sistemi biologici.

In conclusione, il problema del Feedback Vertex Set riveste un ruolo fondamentale nell'individuazione di elementi critici in sistemi interconnessi. Le sue applicazioni pratiche comprendono l'ottimizzazione delle reti di comunicazione, l'analisi dei circuiti integrati, l'individuazione di comunità in social network, l'ottimizzazione dei percorsi nei sistemi di trasporto e l'analisi delle reti biologiche. L'impiego di soluzioni basate sul Feedback Vertex Set consente di prendere decisioni informate per migliorare l'efficienza, la robustezza e la comprensione di una vasta gamma di contesti complessi.

## 2.1 Definizione

Dato un grafo orientato o non orientato  $G = (V, E)$ , un Feedback Vertex Set di  $G$  è un sottoinsieme  $S \subset V$  di vertici che se rimossi rendono  $G$  aciclico. Più formalmente, se  $S \subset V$  è possibile definire il sottografo  $G[S] = (V \setminus S, E_{V \setminus S})$  dove  $E_{V \setminus S} = \{(u, v) \in E : u, v \in V \setminus S\}$ . Se  $G[S]$  è aciclico, allora  $S$  è un Feedback Set.



Il problema del *Minimum Feedback Vertex Set* pone l'obiettivo di ricercare il Feedback Vertex Set di cardinalità minima. Se  $S$  è un Feedback Set, si definisce  $v \in S$  un nodo *ridondante* se il sottografo indotto  $G[S \setminus \{v\}] = ((V \setminus S) \cup \{v\}, E_{(V \setminus S) \cup \{v\}})$  è ancora aciclico. Da questa osservazione segue che  $S$  è un FVS minimo se non contiene alcun nodo ridondante.

Si associ ad ogni vertice  $v \in V$  un peso positivo  $w(v)$  e sia  $S$  un qualunque sottoinsieme di  $V$ , allora il peso di  $S$  sarà definito dalla somma dei pesi dei suoi vertici, ovvero  $\sum_{v \in S} w(v)$ . Il **Minimum Weigthed Feedback Vertex Set** è il problema di ricercare un Feedback Vertex Set di peso minimo<sup>[4]</sup>.

## Algoritmi evolutivi

### 3.1 Introduzione agli algoritmi evolutivi

Gli **algoritmi evolutivi** sono un tipo di algoritmo di ricerca ispirato ai processi evolutivi naturali. Essi utilizzano principi di selezione, crossover e mutazione per generare e migliorare iterativamente una popolazione di soluzioni candidate al problema considerato.

L'esistenza e l'utilità degli algoritmi evolutivi si basano su diverse considerazioni. In primo luogo, essi si ispirano al processo di selezione naturale che ha portato alla diversificazione e all'adattamento degli organismi viventi nel corso dell'evoluzione biologica. L'idea di applicare questi principi per la ricerca delle soluzioni al problema deriva dall'osservazione che la natura spesso trova soluzioni ottimali o vicine all'ottimo in modo efficiente. Gli algoritmi evolutivi cercano di sfruttare questo potenziale generando soluzioni candidate che possono evolvere nel tempo per raggiungere soluzioni migliori.

In secondo luogo, gli essi offrono un approccio flessibile e adattivo per risolvere problemi complessi. Molti problemi del mondo reale sono caratterizzati da una serie di vincoli, incertezza o da una complessità computazionale che rende difficile l'applicazione di metodi di ricerca tradizionali. Gli algoritmi evolutivi possono affrontare tali problemi attraverso l'esplorazione di uno spazio di soluzioni potenzialmente ampio e la scoperta di soluzioni non convenzionali o innovative.

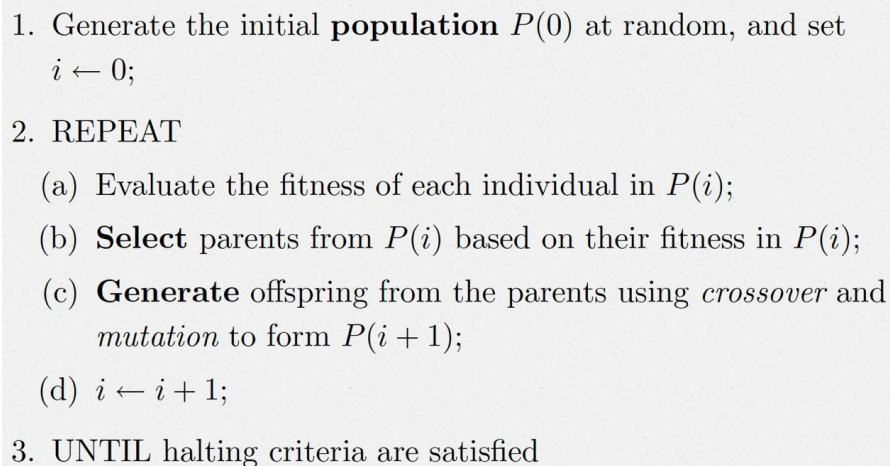
## 3.2 Fasi di un algoritmo evolutivo

La procedura tipica di un algoritmo evolutivo può essere suddivisa in sei step<sup>[7]</sup> fondamentali:

1. **Inizializzazione:** Si crea una popolazione iniziale di soluzioni candidate casuali o generate in modo intelligente. Queste soluzioni rappresentano gli individui all'interno della popolazione.
2. **Valutazione:** Ogni individuo nella popolazione viene valutato rispetto a una funzione obiettivo (*fitness*) che misura la loro idoneità (*feasibility*). Questa funzione può essere definita in base agli obiettivi specifici del problema che si sta affrontando.
3. **Selezione:** Si selezionano gli individui più adatti per la successiva generazione, basandosi sul loro punteggio di idoneità. Gli individui con punteggio più alto hanno una maggiore probabilità di essere selezionati, ma è anche possibile introdurre un elemento di casualità per preservare la diversità all'interno della popolazione.
4. **Crossover:** Gli individui selezionati vengono combinati tra loro attraverso l'applicazione di operatori di *crossover*. Questi operatori simulano il processo di ricombinazione genetica presente nella natura, mescolando le caratteristiche degli individui genitori per generare nuove soluzioni candidate, chiamate discendenti.
5. **Mutazione:** Alcuni individui discendenti subiscono una mutazione casuale, che introduce una piccola variazione nelle loro caratteristiche. Questa variazione casuale aiuta a esplorare lo spazio delle soluzioni in modo più ampio, consentendo la scoperta di nuove possibilità.
6. **Sostituzione:** La nuova generazione di individui, composta dalle soluzioni generate attraverso il crossover e la mutazione, sostituisce (totalmente o

parzialmente) la generazione precedente. Questo assicura che solo gli individui migliori o più promettenti continuino a evolvere nel corso delle iterazioni successive.

7. **Convergenza:** Il processo di selezione, crossover e mutazione viene ripetuto per un certo numero di iterazioni o fino a quando non viene soddisfatto un criterio di convergenza. Questo criterio può essere definito in base a una soglia predefinita di punteggio di idoneità raggiunto o a un numero massimo di iterazioni.



```
1. Generate the initial population  $P(0)$  at random, and set  
    $i \leftarrow 0$ ;  
2. REPEAT  
   (a) Evaluate the fitness of each individual in  $P(i)$ ;  
   (b) Select parents from  $P(i)$  based on their fitness in  $P(i)$ ;  
   (c) Generate offspring from the parents using crossover and  
       mutation to form  $P(i + 1)$ ;  
   (d)  $i \leftarrow i + 1$ ;  
3. UNTIL halting criteria are satisfied
```

Figura 3.1: Pseudocodice di un semplice algoritmo evolutivo

### 3.3 Famiglie di algoritmi evolutivi

In questa sezione approfondiremo le diverse categorie di algoritmi evolutivi<sup>[10]</sup>, evidenziando in particolare l'esistenza di cinque famiglie principali.

#### Algoritmi Genetici (AG)

Gli **Algoritmi Genetici**<sup>[5]</sup> rappresentano probabilmente la forma più conosciuta e diffusa di algoritmi evolutivi. Sono basati sull'idea di evoluzione delle specie e si

ispirano direttamente ai principi genetici e biologici. Le soluzioni ai problemi sono rappresentate come “individui” all’interno di una “popolazione”, e ogni individuo è codificato come una stringa di simboli, tipicamente binari, chiamata “cromosoma”.

Gli AG operano attraverso cicli di selezione (basata sulla fitness di ogni individuo), crossover (per creare nuovi individui) e mutazione (per introdurre variazione). Questi processi permettono alla popolazione di “evolvere” nel tempo, con l’obiettivo di trovare la soluzione ottimale al problema.

### **Programmazione Evolutiva (PE)**

La **Programmazione Evolutiva**<sup>[5]</sup> si distingue per l’attenzione che pone sulla manipolazione e l’ottimizzazione di strutture di programmi. Invece di utilizzare rappresentazioni basate su stringhe come gli AG, la PE lavora con rappresentazioni di programmi, che possono essere strutture di dati complesse come alberi di espressione, o sequenze di istruzioni operative. Questi programmi” sono poi sottoposti a processi evolutivi per ottimizzare le loro performance.

Ad esempio, la PE può essere usata per ottimizzare un algoritmo di apprendimento automatico, “evolvendo” la struttura dell’algoritmo per massimizzare la sua accuratezza su un insieme di dati.

### **Strategie di Evoluzione (SE)**

Le **Strategie di Evoluzione**<sup>[5]</sup> costituiscono un’altra famiglia di algoritmi evolutivi, le quali si concentrano sull’ottimizzazione di vettori di parametri reali. A differenza degli AG e della PE, le SE utilizzano modelli matematici per simulare il processo di evoluzione e selezione.

Le SE sono particolarmente utili per problemi di ottimizzazione continua, dove l’obiettivo è trovare il set di parametri che ottimizza una certa funzione obiettivo. Ad esempio, potrebbero essere usate per ottimizzare i parametri di un controllore in un sistema dinamico.

### Algoritmi Memetici (AM)

Gli **Algoritmi Memetici**<sup>[9]</sup> combinano principi dell'evoluzione genetica con idee provenienti dalla teoria dei memi, un concetto introdotto dal biologo Richard Dawkins per descrivere come le idee si diffondono e si evolvono all'interno di una cultura. In un Algoritmo Memetico, ogni individuo non solo ha una struttura genetica, ma anche un set di “memi” o strategie comportamentali.

Questi algoritmi combinano la ricerca globale dei metodi evolutivi con la ricerca locale di tecniche di ottimizzazione più tradizionali. Il risultato è un approccio ibrido che può esplorare efficacemente lo spazio delle soluzioni, mantenendo al contempo la capacità di sfruttare la conoscenza locale per trovare soluzioni ottimali.

### Algoritmi Culturali (AC)

Gli **Algoritmi Culturali**<sup>[6]</sup> si basano sull'idea che l'evoluzione culturale può fornire un ulteriore livello di adattamento oltre all'evoluzione genetica. In un Algoritmo Culturale, una “cultura” viene modellata come un insieme di credenze condivise tra gli individui di una popolazione.

Queste credenze influenzano il processo di selezione e mutazione, consentendo all'algoritmo di guidare la ricerca verso aree dello spazio delle soluzioni che si sono dimostrate promettenti in passato. Gli Algoritmi Culturali possono quindi adattarsi rapidamente ai cambiamenti nell'ambiente del problema e concentrarsi su parti specifiche dello spazio delle soluzioni.

In conclusione, gli algoritmi evolutivi offrono un approccio potente per affrontare problemi complessi di ottimizzazione e ricerca operativa. Attraverso la simulazione del processo di selezione naturale e l'uso di strategie di evoluzione, questi algoritmi sono in grado di generare soluzioni innovative e adattive che possono superare i limiti delle tecniche tradizionali di ottimizzazione. Le diverse famiglie di algoritmi evolutivi offrono approcci specifici per risolvere una vasta gamma di problemi, rendendoli uno strumento flessibile ed efficace in diversi campi di studio e applicazioni pratiche. La scelta della famiglia di algoritmi evolutivi più adatta dipende dalle caratteristiche specifiche del problema e dalle sue esigenze di ottimizzazione.

---

# 4

## Evolutionary Algorithm with Machine Learning (EAML)

Nel presente capitolo è documentata la fase di progettazione e implementazione dell'algoritmo evolutivo utilizzato per l'approccio al problema e la ricerca di soluzioni ottimali. Saranno inoltre esposte le considerazioni e le scelte operative da queste derivate, la cui applicazione ha consentito la generazione di risultati migliorativi.

### 4.1 Rappresentazione delle soluzioni

Prima di procedere con la progettazione dell'algoritmo, si è reso necessario scegliere la codifica più appropriata per rappresentare le soluzioni. L'obiettivo di questa fase era di garantire un'interpretazione chiara e intuitiva delle soluzioni, rendendo al contempo la fase di sviluppo più agevole. Il formato di una generica soluzione  $x$  è il seguente:

1. Una sequenza di  $n$  bit, rappresentanti gli  $n$  vertici del grafo, tale che, se l' $i$ -esimo bit è posto a 1, il nodo corrispondente non viene incluso nel FVS. Inversamente, se il bit è posto a 0, il nodo è presente nel FVS;
2. Un valore intero rappresentante la misura di fitness calcolata su  $x$ , eventualmente affiancato da una penalità addizionale;
3. Un bit finalizzato a indicare la validità della soluzione trovata in riferimento al problema.

Questa struttura ha fornito un quadro conciso ed efficace per la rappresentazione e l'analisi delle soluzioni, facilitando il processo di sviluppo dell'algoritmo.

#### 4.1.1 Calcolo della fitness di una soluzione

Data una soluzione generica  $x$ , la fitness viene calcolata mediante la seguente funzione:

$$fitness(x) = \sum_{i=0}^n x(i) \cdot w(i) \quad (4.1)$$

In questa equazione,  $x(i)$  denota il bit indicante la presenza o l'assenza dell' $i$ -esimo nodo nella soluzione, mentre  $w(i)$  rappresenta il peso associato a tale nodo all'interno del grafo. Questa funzione ha permesso di quantificare in maniera accurata la bontà delle diverse soluzioni, fornendo un meccanismo chiave per guidare l'evoluzione dell'algoritmo.

#### 4.1.2 Criterio di validità di una soluzione e gestione della penalità

Come deducibile dalle specifiche del problema, una soluzione  $x$  può essere considerata **feasible** se il sottografo da essa indotto risulta aciclico.

Per valutare l'ammissibilità di una soluzione, si è optato per l'implementazione dell'algoritmo **Depth-First Search (DFS)**. La DFS è un algoritmo che naviga in un grafo o albero, scendendo per quanto possibile lungo ogni percorso prima di risalire. Partendo da un nodo designato, visita un nodo adiacente non ancora esplorato, lo contrassegna come visitato e prosegue in modo ricorsivo. Qualora non vi siano più nodi adiacenti non visitati, l'algoritmo ritorna al nodo precedente e ripete il processo fino a che tutti i nodi non siano stati visitati.

Tuttavia, nel caso in cui una soluzione si dimostrasse **unfeasible**, la strategia prescelta è stata quella di conservarla, introducendo una penalizzazione, piuttosto che eliminarla. Questa scelta è nata dalla considerazione che una tale soluzione, pur non aderendo ai vincoli posti dal problema, potrebbe tuttavia svolgere un ruolo



cruciale nel corso dell'evoluzione, favorendo la scoperta di una soluzione feasible e potenzialmente ottimale per il grafo considerato.

Il parametro determinante per l'applicazione della penalità è  $\theta$ , il quale assume un valore costante durante l'esecuzione dell'algoritmo. Questo valore è stato calcolato secondo la seguente formula:

$$\theta = \sum_{i=0}^n w(i) \quad (4.2)$$

Il metodo di applicazione della penalità alla fitness di una soluzione ha subito varie modifiche e perfezionamenti, sia in seguito a osservazioni sia in base ai risultati sperimentali ottenuti.

Inizialmente, in presenza di una soluzione unfeasible, la penalità veniva applicata proporzionalmente alla fitness calcolata, in accordo con la seguente funzione:

$$x(n) = \begin{cases} fitness(x) & \text{se il sottografo indotto da } x \text{ è aciclico} \\ \theta + fitness(x) & \text{altrimenti} \end{cases} \quad (4.3)$$

I risultati sperimentali ottenuti, tuttavia, hanno mostrato scostamenti rilevanti rispetto ai valori ottimali delle istanze analizzate; di conseguenza, si è deciso di adottare un diverso sistema di penalità. Tale decisione si è basata sull'osservazione che, per raggiungere un valore ottimo di FVS, sarebbe necessario garantire una maggiore variabilità nella popolazione dell'algoritmo evolutivo. A tal proposito, si è notato che una soluzione unfeasible con un "peso" minore, sebbene più simile a una potenziale soluzione feasible, avrebbe ridotto il grado di variabilità delle soluzioni nella popolazione. Al contrario, una soluzione unfeasible con un "peso" maggiore avrebbe garantito un più alto grado di variabilità nella popolazione, consentendo all'algoritmo di superare più facilmente gli stalli dovuti al raggiungimento di ottimi locali.

Sulla base di tali assunzioni, la funzione di penalità è stata quindi modificata come segue:

$$x(n) = \begin{cases} fitness(x) & \text{se il sottografo indotto da } x \text{ è aciclico} \\ \theta + \frac{\theta}{fitness(x)} & \text{altrimenti} \end{cases} \quad (4.4)$$

Come si può osservare, la penalizzazione è determinata da un fattore inversamente proporzionale alla fitness calcolata sulla soluzione. Questo approccio ha permesso di ottenere fin da subito risultati migliori da parte dell'algoritmo.

## 4.2 Generazione della popolazione iniziale

Nel corso dello sviluppo dell'algoritmo, la creazione della popolazione iniziale ha subito diverse modifiche, volte a consentire la costruzione di soluzioni ottimali.

Inizialmente, ogni soluzione che componeva la popolazione iniziale veniva generata in maniera del tutto casuale, sulla base della seguente funzione:

$$x(i) = \begin{cases} 1 & \text{se } rnd \text{ è pari} \\ 0 & \text{se } rnd \text{ è dispari} \end{cases} \quad (4.5)$$

dove  $x(i)$  rappresenta il bit di presenza dell' $i$ -esimo nodo nella soluzione e  $rnd$  è un numero intero generato casualmente.

In seguito, con l'intento di migliorare le prestazioni dell'algoritmo, è stato modificato l'approccio, introducendo un metodo greedy combinato a una componente stocastica. L'obiettivo era di includere con maggiore probabilità in soluzione un nodo di peso minore piuttosto che uno di peso maggiore, pur mantenendo un grado di casualità. Questa strategia ha permesso, da un lato, di costruire soluzioni di partenza migliori, contenenti nodi di peso inferiore che molto probabilmente sarebbero entrati a far parte del FVS finale. Dall'altro lato, grazie alla componente stocastica, si è evitata la generazione di soluzioni identiche, ovvero contenenti tutti gli stessi nodi, conferendo così una maggiore variabilità alla popolazione iniziale. Su queste basi, è stata definita una probabilità  $p$  (definita in percentuale) attraverso la seguente formula:

$$p = \frac{1}{2} \left[ (rnd \bmod 100) + \left( \frac{w(i)}{\max w} \cdot 100 \right) \right] \quad (4.6)$$

dove  $rnd$  indica un valore intero generato casualmente,  $w(i)$  rappresenta il peso del nodo  $i$ -esimo nel grafo, e  $\max w$  si riferisce al peso massimo associato a un nodo all'interno del grafo. Come si può osservare, la probabilità calcolata, normalizzata

nell'intervallo  $[0, 100[$ , è definita dalla media di due fattori: il primo è stocastico, dato da un numero casuale normalizzato nello stesso intervallo; il secondo è deterministico, rappresentato dal rapporto percentuale tra il peso dell' $i$ -esimo nodo considerato e il massimo peso che un nodo può assumere nel grafo.

Di conseguenza, la funzione originale è stata modificata come segue:

$$x(i) = \begin{cases} 1 & \text{se } p \leq 50\% \\ 0 & \text{se } p > 50\% \end{cases} \quad (4.7)$$

Un nodo verrà quindi aggiunto alla soluzione solo se la probabilità calcolata su di esso è inferiore al 50%.

### 4.3 Crossover

L'operatore di crossover è una componente fondamentale che permette di combinare la sequenza di bit di due soluzioni scelte in modo casuale all'interno della popolazione, generando così due nuove soluzioni che saranno utilizzate per la generazione successiva.

In una prima fase, erano stati presi in considerazione tre tipologie distinte di crossover: 3-point, 2-point e 1-point. Questi operatori venivano applicati in modo dinamico man mano che l'algoritmo procedeva con le iterazioni. Il crossover di tipo 3-point veniva utilizzato nelle fasi iniziali, successivamente si transiva a un metodo 2-point e infine a un crossover di tipo 1-point nelle fasi finali.

Tuttavia, a seguito di successive analisi e prove sperimentali, è emerso che l'utilizzo del solo crossover di tipo 1-point forniva risultati migliori rispetto all'applicazione degli altri tipi di crossover o di una combinazione di essi. Pertanto, è stata presa la decisione di adottare come unica tipologia di crossover il metodo 1-point nella versione finale dell'algoritmo.

Inoltre, è stato sperimentato un quarto tipo di crossover, l'uniform crossover. Nonostante i numerosi test condotti, a causa dell'eccessiva variabilità introdotta nelle soluzioni generate, è stata presa la decisione di non includerlo nella versione finale dell'algoritmo.

## 4.4 Mutazione

Nella presente sezione, verranno illustrate le metodologie di mutazione implementate nell'ambito dell'algoritmo, con particolare attenzione alle loro evoluzioni apportate per migliorare la qualità dei risultati. Si esporrà inoltre la strategia di **machine learning**<sup>[2]</sup> impiegata per modulare dinamicamente il *valore soglia* di mutazione, contribuendo così alla generazione di soluzioni progressivamente migliori.

### 4.4.1 Operatori di mutazione Pre-ML

La mutazione rappresenta un operatore che, con una probabilità  $p$  (indicata in percentuale), consente la variazione di uno o più bit di una data soluzione.

Nel contesto del presente studio, sono stati sviluppati due operatori di mutazione distinti, con i quali si sono eseguiti vari esperimenti al fine di ottimizzare i risultati ottenuti. Questi operatori si basano su un criterio mutativo comune: fino a quando la probabilità di mutazione non supera una specifica soglia, è possibile mutare qualsiasi bit nella soluzione. Se tale probabilità eccede la soglia stabilita, solo i bit che attualmente sono a zero possono essere soggetti a mutazione. Il principio alla base di questa concezione sarà discusso in dettaglio nel prossimo capitolo.

Il primo operatore di mutazione opera generando, per ogni bit nella soluzione, un numero casuale compreso nell'intervallo  $[0, 100[$ . Se questo numero risulta inferiore a  $p$ , si applica la mutazione seguendo il criterio precedentemente delineato.

In contrasto con il primo, il secondo operatore di mutazione introduce una componente deterministica accanto all'approccio stocastico. In questo caso, la probabilità di mutazione di un bit è influenzata anche dal grado che il nodo corrispondente assume all'interno del sottografo. L'approccio è descritto dalla seguente funzione:

$$p(x(i)) = \frac{1}{2} \left[ (rnd \bmod 100) + \left( \frac{degree(x(i))}{n} \cdot 100 \right) \right] \quad (4.8)$$

dove  $rnd$  è un numero generato casualmente e  $degree(x(i))$  rappresenta il grado che l' $i$ -esimo nodo assume nel sottografo.

Come illustrato dalla formula, l'interazione tra determinismo e casualità è ottenuta attraverso la media aritmetica tra un valore percentuale generato casualmente e un valore percentuale derivato dalla topologia del grafo. Ancora una volta, il criterio di mutazione rimane quello descritto all'inizio del paragrafo.

Il secondo operatore di mutazione si basa sull'idea di mantenere costanti i bit relativi ai nodi di grado elevato, che saranno gestiti dall'algoritmo di ricerca locale, favorendo la scoperta di potenziali nuove soluzioni mutando i nodi di grado più basso, che sarebbero stati modificati con maggiore difficoltà durante l'implementazione dell'algoritmo.

In una prima fase, tra i due operatori, il secondo ha dimostrato di produrre risultati sperimentali più efficaci, ed è stato pertanto selezionato come l'unico operatore di mutazione definitivo per l'implementazione dell'algoritmo.

#### 4.4.2 Operatori di mutazione Post-ML

In questa sezione sarà illustrato il funzionamento dell'algoritmo di machine learning impiegato per la modifica dinamica della soglia di mutazione, così come le modifiche apportate all'operatore di mutazione con l'obiettivo di ottimizzare i risultati generati.

Per favorire una comprensione ottimale dell'algoritmo, si rende necessario introdurre preliminarmente il concetto di regressione lineare, che costituisce il fondamento dell'applicazione di machine learning. Successivamente, verranno delineati i principi operativi dell'algoritmo stesso.

##### Regressione lineare

La **regressione lineare**<sup>[1]</sup> è una tecnica utilizzata per studiare la relazione tra due o più variabili continue. Nella regressione lineare, si cerca di identificare la linea retta migliore che si adatta ai dati, minimizzando gli errori di previsione. La variabile di risposta (o variabile dipendente) è una variabile continua, mentre le variabili indipendenti (o variabili predittive) possono essere sia continue che categoriche. L'obiettivo è di stimare i coefficienti della regressione, che indicano la relazione tra le variabili indipendenti e la variabile dipendente. L'interpretazione dei risultati si ba-

sa sui coefficienti stimati, che indicano quanto varia la variabile di risposta al variare di una unità delle variabili indipendenti.

La regressione lineare può essere espressa matematicamente come:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_k x_k + \epsilon \quad (4.9)$$

dove  $y$  è la variabile di risposta,  $x_1, x_2, \dots, x_k$  sono le variabili indipendenti,  $\beta_0$  è l'intercetta (ovvero il valore di  $y$  quando tutte le variabili indipendenti sono pari a zero),  $\beta_1, \beta_2, \dots, \beta_k$  sono i coefficienti di regressione (ovvero l'effetto delle variabili indipendenti sulla variabile di risposta), e  $\epsilon$  rappresenta l'errore residuo (ovvero la differenza tra il valore osservato di  $y$  e il valore previsto dalla regressione).

La regressione lineare può essere effettuata utilizzando diversi metodi, tra cui il metodo dei minimi quadrati. In questo metodo, si cerca di minimizzare la somma dei quadrati degli errori residui, ovvero la somma delle differenze al quadrato tra i valori osservati e i valori previsti dalla regressione.

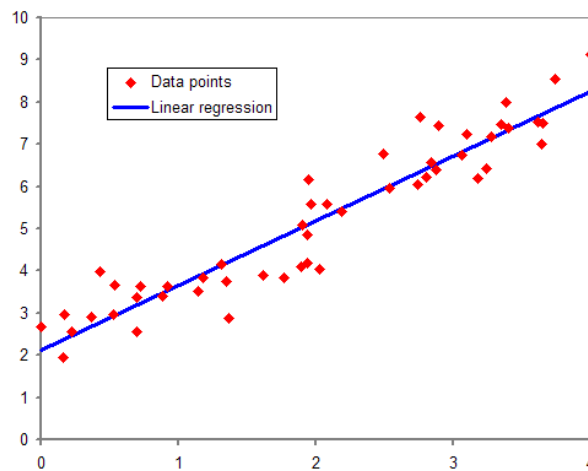


Figura 4.1: Esempio di regressione lineare con una variabile dipendente e una indipendente

Per interpretare i risultati della regressione lineare, è importante considerare i coefficienti stimati. Ad esempio, se il coefficiente di regressione per una variabile indipendente è positivo, significa che all'aumentare di quella variabile, la variabile di

risposta aumenterà. Viceversa, se il coefficiente di regressione è negativo, significa che all'aumentare di quella variabile, la variabile di risposta diminuirà. Il coefficiente di regressione può anche essere espresso in termini di unità di misura delle variabili, ad esempio come per ogni aumento di una unità di  $x$ , la variabile di risposta aumenta/diminuisce di  $\beta$  unità.

Per stimare i coefficienti di regressione nel modello di regressione lineare, uno dei metodi più comuni è l'**Ordinary Least Squares (OLS)**. L'obiettivo dell'OLS è minimizzare la somma dei quadrati delle differenze tra i valori osservati e i valori previsti dal modello di regressione, noti anche come residui. In sostanza, l'OLS cerca la linea di regressione che minimizza la distanza quadratica totale tra i punti di dati osservati e la linea stessa.

Il calcolo dei coefficienti di regressione con l'OLS richiede una serie di ipotesi. Queste includono linearità, indipendenza degli errori, varianza costante degli errori e normalità degli errori. La violazione di queste ipotesi può portare a stime inefficaci o fuorvianti.

### L'algoritmo di machine learning<sup>[2]</sup>

Per migliorare le prestazioni dell'algoritmo, è stato utilizzato come modello predittivo la tecnica dell'Ordinary Least Squares precedentemente introdotta, allo scopo di prevedere la bontà della popolazione nelle generazioni successive. Tale modello viene addestrato osservando come la fitness media della popolazioni è variata nelle ultime  $v$  generazioni, ed è formalizzato come segue:

$$y = \alpha x + \beta \quad (4.10)$$

Assumendo  $Y = (Y_1, \dots, Y_v)$ , il quale rappresenta il vettore contenente la media delle fitness della popolazione in  $v$  generazioni, e  $X = (X_1, \dots, X_v)$ , il quale rappresenta il vettore che contiene le  $v$  generazioni, e considerando inoltre che  $\sigma_X^2$  e  $\sigma_{XY}$  sono rispettivamente la varianza di  $X$  e la covarianza tra  $X$  e  $Y$ , i parametri  $\alpha$  e  $\beta$  possono essere calcolati come segue:

$$\alpha = \frac{\sigma_{XY}}{\sigma_X^2} = \frac{\sum_{i=1}^v (X_i - \bar{X})(Y_i - \bar{Y})}{\sum_{i=1}^v (X_i - \bar{X})^2} \quad (4.11)$$

$$\beta = \bar{Y} - \alpha \bar{X} \quad (4.12)$$

dove  $\bar{X}$  e  $\bar{Y}$  sono rispettivamente le medie di  $X$  e  $Y$ .

Considerando che  $X$  è un vettore costante che sarà sempre uguale a  $(1, 2, 3, \dots, v)$ , la sua media e la sua varianza saranno costanti e possono essere definite come:

$$\bar{X} = \frac{1}{v} \sum_{i=1}^v i = \frac{1}{v} \frac{v(v+1)}{2} = \frac{1}{2}(v+1) \quad (4.13)$$

$$\sigma_X^2 = \frac{1}{v} \sum_{i=1}^v (i - \bar{X})^2 = \frac{1}{v} \sum_{i=1}^v i^2 - \bar{X}^2 \quad (4.14)$$

Dal momento che  $\sum_{i=1}^v i^2 = \frac{v^3}{3} + \frac{v^2}{2} + \frac{v}{6}$ , ne segue che  $\frac{1}{v} \sum_{i=1}^v i^2 = \frac{v^2}{3} + \frac{v}{2} + \frac{1}{6}$ . Possiamo dunque riscrivere l'equazione (4.14) come:

$$\sigma_X^2 = \frac{v^2}{3} + \frac{v}{2} + \frac{1}{6} - \frac{1}{4}(v+1)^2 = \frac{v^2 - 1}{12} \quad (4.15)$$

Pertanto, questa conoscenza può essere utilizzata per regolare dinamicamente alcuni parametri dell'algoritmo genetico. Nel nostro caso, abbiamo utilizzato un modello predittivo simile per aggiornare la probabilità di mutazione e di crossover ad ogni generazione. Per di più, considerando che non è stato attuato alcun scostamento della retta di regressione rispetto all'origine, il parametro  $\beta$  non è stato ritenuto pertinente per lo sviluppo dell'algoritmo.

Nella sua applicazione pratica, l'algoritmo proposto utilizza una coda di capacità  $v$ , in cui vengono registrate le fitness medie delle ultime  $v$  popolazioni. In base a tale metodologia, sono stati ideati specifici metodi per il calcolo della fitness media sulla popolazione in una determinata generazione e sulla coda, al fine di raccogliere tutti i dati indispensabili per il calcolo del parametro  $\alpha$ .

Il parametro  $\alpha$  consente di stimare l'evoluzione della qualità degli individui nelle generazioni successive. Da una prospettiva geometrica, rappresenta la pendenza



della linea di regressione. Sulla base del suo segno, è possibile definire la regola di aggiornamento per il valore soglia di mutazione  $p_m^{(t)}$ . Fondamentalmente, dato un valore (piccolo)  $\epsilon \geq 0$ , ci sono tre casi:

1.  $\alpha > \epsilon$  La fitness media della popolazione tende ad aumentare, quindi la probabilità di mutazione diminuisce gradualmente di una quantità pari a  $\delta$  per generazione. La probabilità di mutazione  $p_m^{(t)}$  alla generazione  $t$  viene calcolata come segue:

$$p_m^{(t)} = (p_m^{(t-1)} - \delta) \Big|_{p_{max}}^{p_{min}} \quad (4.16)$$

Questo implica che  $x$ , come denotato dalla notazione  $(x)|_{p_{max}}^{p_{min}}$ , è limitato all'intervallo tra  $p_{min}$  e  $p_{max}$ . Il parametro  $\delta$  definisce di quanto la probabilità di mutazione deve essere aumentata o diminuita ad ogni generazione. Per semplicità, utilizziamo il parametro  $p_m^l$  per suddividere l'intervallo di probabilità  $[p_{min}, p_{max}]$  in intervalli omogenei di valori e, di conseguenza,  $\delta$  è definito come:

$$\delta = \frac{p_{max} - p_{min}}{p_m^l} \quad (4.17)$$

2.  $\alpha < \epsilon$  La fitness media sta diminuendo e quindi la probabilità di mutazione aumenta rapidamente di  $\delta$ .

$$p_m^{(t)} = (p_m^{(t-1)} + \delta) \Big|_{p_{max}}^{p_{min}} \quad (4.18)$$

3.  $-\epsilon \leq \alpha \leq \epsilon$  Quando  $\alpha$  è compreso tra  $-\epsilon$  e  $\epsilon$ , la fitness media oscilla in un piccolo intervallo e, di conseguenza, la probabilità di mutazione aumenta gradualmente di un valore pari a  $\frac{\delta}{p_m^f}$  al fine di introdurre causalità nella popolazione:

$$p_m^{(t)} = \left( p_m^{(t-1)} - \frac{\delta}{p_m^f} \right) \Big|_{p_{max}}^{p_{min}} \quad (4.19)$$

Il parametro  $p_m^f$  è un numero intero positivo utilizzato per aumentare la probabilità di una frazione di  $\delta$  quando la linea di regressione può essere approssimata a una costante ( $\alpha \approx 0$ ).

### Come avviene la mutazione Post-ML?

Nell'ottica dell'implementazione dell'algoritmo di Machine Learning esposto nella sezione precedente, si è resa necessaria l'introduzione di un rinnovato meccanismo di mutazione che potesse integrarsi efficacemente con la dinamicità imposta dal Machine Learning.

Nonostante la modalità di calcolo della probabilità di mutazione sia rimasta invariata, così come delineata nella formula (4.8), il criterio precedentemente basato sull'applicazione di un valore soglia, il quale prevedeva la mutazione esclusiva dei bit posti a 0, è stato rielaborato in maniera tale da amalgamarsi in modo ottimale con i parametri introdotti tramite il Machine Learning.

In questo specifico contesto, la mutazione dei soli bit posti a 0 avviene quando  $|\alpha| \leq \epsilon$ . Le ragioni che hanno condotto a questa decisione saranno ulteriormente esplorate e chiarite nel capitolo successivo.

## 4.5 Sostituzione

Nell'ottica di migliorare i risultati prodotti dall'algoritmo, sono state implementate tre diverse varianti di operatori di sostituzione, chiamati nel gergo del presente progetto di tesi **join**.

La prima variante consiste nell'includere nella generazione successiva solamente le soluzioni migliori, provenienti dalla combinazione delle popolazioni di genitori e figli. In modo simile, la seconda variante prevede di preservare le migliori  $\frac{m}{2}$  soluzioni da entrambe le popolazioni.

Durante le fasi preliminari dello sviluppo dell'algoritmo, si è deciso di implementare una alternanza dinamica tra queste due varianti, con l'obiettivo di favorire una maggiore variabilità all'interno della popolazione. In seguito, è stata introdotta una terza variante, con l'obiettivo di incrementare ulteriormente la diversità all'in-

terno della popolazione. Tale variante funziona unendo le popolazioni di genitori e figli, successivamente ordina il risultato in base alla fitness in modo crescente, dopodiché rimuove tutte le soluzioni che presentano la stessa sequenza di bit rispetto alla soluzione che le precede immediatamente nella lista.

Dopo aver condotto una serie di test, si è constatato che la terza tipologia di selezione ha fornito i migliori risultati sperimentali. Pertanto, si è deciso di adottarla come unica tipologia all'interno dell'algoritmo.

## 4.6 Ricerca locale

Prima di procedere con la generazione successiva, è stata applicata alla popolazione una tecnica di ricerca locale. Questo algoritmo mira a rimuovere un nodo  $u$  dal FVS, selezionandolo in base al suo **rapporto peso/grado** massimo, al fine di ottenere una possibile soluzione con peso inferiore. Se il reinserimento di  $u$  nel sottografo indotto non genera alcun ciclo, significa che il nodo era ridondante e può essere direttamente eliminato dalla soluzione. Se, invece, la reintroduzione di  $u$  crea un ciclo nel sottografo indotto, si procede cercando un nodo  $v$  nella componente connessa a  $u$ , seguendo un ordine crescente del rapporto peso/grado, in modo tale che la rimozione di  $v$  dal sottografo renda il quest'ultimo aciclico. Se non esiste un nodo  $v$  che possa rendere aciclico il sottografo, viene reinserito il nodo  $u$  nel FVS.

L'idea alla base di questa tecnica è quella di alleggerire la soluzione cercando un nodo  $u$  con un peso significativo, la cui reintroduzione nel sottografo generi con minor probabilità la formazione di un ciclo (essendo un nodo con grado basso). Se nonostante ciò si verifica un ciclo, si cerca il nodo di peso minimo nella componente connessa a  $u$  nel sottografo, in modo che la sua rimozione possa con maggiore probabilità render il sottografo aciclico (essendo un nodo con grado alto).

---

**Algorithm 1:** Local Search

---

**Input:** pop, adj, w, n, inf, sup, min**Output:** Updates pop and min with the locally searched solutions and the minimum weight of the best solution.

```

1 for  $i$  in range(inf, sup) do
2   copy = copy of pop[i]
3   it = solutionCardinality(pop[i], n)
4   for  $j$  in range(0, it) do
5     u = select node with maximum weight/degree ratio from copy
6     remove u from pop[i] and copy
7     if there exists a cycle in pop[i] starting from u then
8       Compute connected components of u in pop[i] and store them in
       conn
9       for  $k$  in range(0, n) do
10        v = select node with minimum weight/degree ratio from conn
11        add v to pop[i]
12        if there is no cycle in pop[i] starting from u then
13          Update pop[i][n] with pop[i][n] - w[u] + w[v]
14          Break
15        end
16      else
17        remove v from pop[i]
18      end
19    end
20    if no v was found then
21      reinsert u in pop[i]
22    end
23  end
24  else
25    decrease pop[i][n] by w[u]
26  end
27 end
28 end
29 Sort pop using sortcol
30 if  $min > pop[0][n]$  then
31   min = pop[0][n]
32 end

```

---

---

# 5

## Risultati

Nel presente capitolo, sarà delineata la fase di definizione dei parametri che influenzano l'operatività dell'algoritmo prima e dopo l'applicazione del metodo di machine learning, coerentemente con le considerazioni esposte nel capitolo antecedente. Inoltre, si procederà all'analisi dei risultati conseguiti, utilizzando come criterio di confronto i risultati ottenuti dalle corrispondenti istanze benchmark riportati dall'algoritmo Hybrid-IA<sup>[3][4]</sup>.

### 5.1 Parametri di funzionamento dell'algoritmo

L'algoritmo evolutivo è stato implementato con una popolazione di dimensione  $m = 100$  soluzioni, reiterando il ciclo evolutivo per  $generationNumber = 1000$  generazioni.

Come discusso nel capitolo precedente, inizialmente le varianti di crossover (*crossoverType*) sono state adattate dinamicamente durante il corso del ciclo evolutivo: la variante di partenza è stata impostata su 3-point (*crossoverType* = 3); successivamente, dopo aver elaborato il 50% delle generazioni, si è passati alla variante 2-point (*crossoverType* = 2); alla soglia del 75% delle generazioni, la variante di crossover è stata mutata in 1-point (*crossoverType* = 1).

La scelta di quest'approcci adottato di crossover, in una prima fase delle implementazioni, è stata guidata da una precisa strategia di dinamicità. Durante le fasi iniziali del processo, si è voluto promuovere un alto grado di variabilità attraverso l'applicazione di un 3-point crossover, con l'obiettivo di esplorare in maniera estesa lo spazio delle soluzioni. Successivamente, con l'avanzare delle generazioni, si è de-

ciso di contenere progressivamente questa variabilità passando prima a un 2-point e infine ad un 1-point crossover. Questa graduale riduzione del grado di variabilità è stata finalizzata a limitare il livello di *chaos* introdotto nel processo e a favorire, invece, il raffinamento delle soluzioni ottenute. Tale strategia ha permesso un'efficace bilanciamento tra esplorazione dello spazio delle soluzioni e l'exploitation delle soluzioni promettenti individuate.

In seguito a un'estesa serie di test, che hanno considerato sia questo approccio sia l'applicazione di un'unica variante di crossover, i risultati sperimentali ottenuti hanno indirizzato la scelta verso l'applicazione esclusiva del 1-point crossover, in quanto ha permesso di ottenere risultati migliori.

### 5.1.1 Parametri di mutazione Pre-ML

La soglia mutativa iniziale (*MUTATION\_ODD*) è stata stabilita al 5.00% e, in presenza di stagnazione in un ottimo locale, è stata incrementata in maniera dinamica. Tale incremento è avvenuto secondo il seguente principio: qualora la stagnazione avesse superato le 30 generazioni, la percentuale di mutazione sarebbe transitata al 15.00%; se avesse superato le 60 generazioni, avrebbe raggiunto il 25.00%; al raggiungimento delle 90 generazioni, sarebbe passata al 50.00%. Il valore sarebbe stato ripristinato a quello originale nel caso in cui fosse stato possibile oltrepassare la fase di stallo.

In aggiunta, è stato impiegato un parametro soglia per la mutazione (*MUTATION\_ODD\_THRESHOLD*), determinato alla percentuale del 25.00%, attraverso il quale è stato possibile modificare nel corso delle generazioni la modalità di operare del metodo di mutazione, come esposto nel Capitolo 4.4.1. La logica sottesa a questa applicazione può essere esposta come segue: fintantoché la percentuale di mutazione rimane inferiore alla soglia stabilita, la stagnazione in un ottimo locale non perdura per un periodo prolungato e si opta per mutare qualunque tipo di bit nelle soluzioni; al contrario, se la percentuale di mutazione oltrepassa la soglia preimpostata, la stagnazione perdura da molte generazioni e si sceglie di mutare esclusivamente i bit posti a 0. Questa metodologia si rivela particolarmente utile in

quanto, nonostante la rimozione di un nodo dal sottografo e la conseguente aggiunta alla soluzione possano “appesantire” quest’ultima, è più probabile che il sottografo indotto sia aciclico, permettendo così la scoperta, soprattutto durante le fasi iniziali dell’algoritmo, delle prime soluzioni feasible.

### 5.1.2 Parametri di mutazione Post-ML

L’integrazione dell’approccio di machine learning ha prodotto un impatto significativo sia sui risultati ottenuti, come sarà illustrato nella sezione seguente, sia nell’impiego dei parametri iniziali utilizzati dall’algoritmo per il calcolo della soglia mutativa.

Come delineato nel capitolo antecedente, l’approccio di machine learning si avvale dell’uso di alcuni parametri cruciali. I valori di tali parametri sono stati affinati nel corso delle sperimentazioni, sulla base dei risultati prodotti dall’algoritmo. In particolar modo, la soglia mutativa iniziale *MUTATION\_ODD* è stata fissata al 30.00%. Si è constatato che soglie mutative iniziali particolarmente basse o elevate (ad esempio 5.00% o 75.00%) avevano un impatto negativo sull’algoritmo, precludendo la generazione di soluzioni feasible, mentre l’adozione di una soglia mutativa intermedia (ad esempio 25.00% o 30.00%) agevolava l’esplorazione dello spazio delle soluzioni in modo efficiente, favorendo la generazione di soluzioni feasible.

Per quanto concerne i parametri dell’algoritmo esposti nel capitolo precedente, sono stati condotti numerosi test nel corso dei quali si è cercato di ottimizzare i valori assegnati a tali parametri. In seguito a una serie di esperimenti, si è giunti a definire i seguenti valori, in funzione dei risultati conseguiti:

- *DELTA* = 1, che corrisponde al parametro  $\delta$  delineato nella formula (4.17). Si specifica che, dato che si è scelto di dividere la scala mutativa in circa 100 parti, non è stato definito il parametro  $p_m^l$ .
- *EPSILON* = 0.5, che corrisponde al valore del parametro  $\epsilon$  adoperato come riferimento per l’applicazione della regola di aggiornamento della soglia di mutazione.

- $LAMBDA = 0.2$ , che corrisponde al parametro  $p_m^f$  dettagliato nella formula (4.19).

## 5.2 Analisi dei risultati

L'indagine sui risultati ottenuti è stata condotta confrontando le performance registrate dall'algoritmo Hybrid-IA con quelle raggiunte dall'algoritmo oggetto del nostro studio, sia prima dell'introduzione dell'approccio di Machine Learning (ML) che successivamente alla sua applicazione. Il confronto è stato effettuato sia sulle singole istanze che sulle medie calcolate su grafi con identiche caratteristiche topologiche ma semi di generazione differenti.

### Analisi dei risultati Pre-ML

Nell'ambito del confronto condotto sulle singole istanze, è stato riscontrato che circa il 49.17% dei grafi elaborati dall'algoritmo evolutivo ha raggiunto il valore ottimale di FVS, mentre il 75.00% ha mantenuto uno scostamento rispetto all'ottimale inferiore al 5.00%.

Nell'analisi delle medie, calcolate su grafi con le stesse caratteristiche topologiche ma semi di generazione differenti, si è osservato che il 16.67% ha raggiunto l'ottimo, il 70.83% ha mantenuto lo scostamento rispetto all'ottimo sotto l'1.00%, e il massimo scostamento rilevato si attesta al 3.02%.

Come mostrato nelle tabelle 5.1 e 5.2, l'algoritmo ha conseguito notevoli risultati su grafi random più densi, mantenendo la percentuale di scostamento rispetto a Hybrid-IA al di sotto dell'1.00%. D'altro canto, su grafi random più sparsi e su grafi grid, l'algoritmo ha riscontrato maggiore difficoltà nel rintracciare l'ottimo. Tale difficoltà è attribuibile al fatto che, sebbene nei grafi più sparsi ci sia maggiore variabilità nella popolazione, la ricerca di soluzioni ottimali risulta più complessa a causa della specifica topologia di questi ultimi. In contrasto, nei grafi densi, l'obbligo di inserire in soluzione un maggior numero di nodi rende più agevole il raggiungimento dell'ottimo mediante le manipolazioni stocastiche e deterministiche applicate.



Sulla base delle considerazioni sopra esposte, è plausibile ritenere accettabili i risultati ottenuti, considerato che lo scostamento massimo registrato non supera un limite massimo del 5.00%.

Si riportano di seguito i dettagli nelle tabelle sottostanti, differenziate per **grafi random** e **grafi grid**.

Tabella 5.1: Tabella dei risultati grafi Rand pre approccio ML

<b>n</b>	<b>e</b>	<b>l</b>	<b>u</b>	<b>EA</b>	<b>HybridIA</b>	<b>%</b>
100	247	10	25	507.6	498.4	1.846
100	247	10	50	851.2	833.8	2.087
100	247	10	75	1242.6	1207.2	2.932
100	3069	10	25	1134	1134	0
100	3069	10	50	2179	2179	0
100	3069	10	75	3229.2	3228.6	0.0186
100	841	10	25	831.2	826.8	0.532
100	841	10	50	1738	1724.4	0.789
100	841	10	75	2442	2420.4	0.892
50	232	10	25	387	386.2	0.207
50	232	10	50	710.4	708.6	0.254
50	232	10	75	954.8	951.6	0.336
50	784	10	25	602.2	602	0.033
50	784	10	50	1171.8	1171.8	0
50	784	10	75	1648.8	1648.8	0
50	85	10	25	176	174.6	0.801
50	85	10	50	283.6	280.8	0.997
50	85	10	75	348.2	348	0.057

### Analisi dei risultati Post-ML

In seguito all'esecuzione di numerosi esperimenti, sono stati accumulati i risultati derivanti dall'implementazione dell'approccio di Machine Learning (ML) all'interno dell'algoritmo.

I risultati più pregevoli hanno registrato l'ottenimento di circa il 70.83% dei valori ottimali riportati dall'algoritmo Hybrid-IA, mentre circa il 96.67% ha manifestato

Tabella 5.2: Tabella dei risultati grafi Grid pre approccio ML

<b>n</b>	<b>e</b>	<b>l</b>	<b>u</b>	<b>EA</b>	<b>HybridIA</b>	<b>%</b>
100	180	10	25	575.2	566.2	1.590
100	180	10	50	967	945.6	2.263
100	180	10	75	1603	1556	3.021
49	84	10	25	253.8	252	0.714
49	84	10	50	441.4	437.6	0.868
49	84	10	75	724.4	713.6	1.513

n = numero di nodi

e = numero di archi

l = estremo inferiore del peso di un nodo nel grafo

u = estremo superiore del peso di un nodo nel grafo

EA = valore di fitness registrato prima dell'applicazione dell'approccio ML

Hybrid-IA = valore di fitness riportato dall'algoritmo Hybrid-IA

% = Scostamento percentuale tra i due algoritmi

uno scostamento inferiore al 5.00%.

Nell'indagine sulle medie relative a grafi con identica topologia ma con semi di generazione differenti, è stato rilevato il raggiungimento dell'ottimo per il 41.67%, mentre il 100.00% delle istanze ha mantenuto lo scostamento rispetto all'ottimo al di sotto dell'1.00%.

Analogamente alla precedente casistica, come mostrato nelle tabelle 5.3 e 5.4, i risultati di maggior rilievo sono stati ottenuti su grafi random, in particolar modo quelli più densi. Per quanto riguarda i grafi grid, sebbene sia stata rilevata una notevole riduzione della percentuale di scostamento, non è stato possibile raggiungere il valore ottimale per nessuna famiglia di istanze.

Si riportano di seguito le tabelle con i dettagli dei risultati, distinte per grafi random e grafi grid.

Tabella 5.3: Tabella dei risultati grafi Rand post approccio ML

<b>n</b>	<b>e</b>	<b>l</b>	<b>u</b>	<b>EAML</b>	<b>HybridIA</b>	<b>%</b>
100	247	10	25	499.4	498.4	0.200
100	247	10	50	836.4	833.8	0.311
100	247	10	75	1213.4	1207.2	0.513
100	3069	10	25	1134	1134	0
100	3069	10	50	2179	2179	0
100	3069	10	75	3228.6	3228.6	0
100	841	10	25	826.8	826.8	0
100	841	10	50	1725	1724.4	0.034
100	841	10	75	2420.4	2420.4	0
50	232	10	25	386.2	386.2	0
50	232	10	50	709.8	708.6	0.169
50	232	10	75	951.6	951.6	0
50	784	10	25	602	602	0
50	784	10	50	1172	1171.8	0.017
50	784	10	75	1648.8	1648.8	0
50	85	10	25	175.4	174.6	0.458
50	85	10	50	283	280.8	0.783
50	85	10	75	348	348	0

Tabella 5.4: Tabella dei risultati grafi Grid post approccio ML

<b>n</b>	<b>e</b>	<b>l</b>	<b>u</b>	<b>EAML</b>	<b>HybridIA</b>	<b>%</b>
100	180	10	25	567.6	566.2	0.247
100	180	10	50	950.2	945.6	0.486
100	180	10	75	1566.8	1556	0.694
49	84	10	25	253.6	252	0.635
49	84	10	50	438.4	437.6	0.183
49	84	10	75	720.8	713.6	1.009

n = numero di nodi

e = numero di archi

l = estremo inferiore del peso di un nodo nel grafo

u = estremo superiore del peso di un nodo nel grafo

EAML = valore di fitness registrato dopo dell'applicazione dell'approccio ML

Hybrid-IA = valore di fitness riportato dall'algoritmo Hybrid-IA

% = Scostamento percentuale tra i due algoritmi

---

# Conclusioni

In questa tesi, è stata proposta un'euristica basata su un algoritmo evolutivo per affrontare il problema del Feedback Vertex Set. Quest'ultimo è un problema di rilevante importanza nella teoria dei grafi e trova molteplici applicazioni pratiche in vari settori, tra cui l'ottimizzazione di reti e la bioinformatica, grazie alla sua capacità di identificare insiemi minimi di nodi che, una volta rimossi, rendono un grafo aciclico.

Nel capitolo intitolato “*Evolutionary Algorithm with Machine Learning*”, si è discusso della creazione di vari operatori di crossover, mutazione e sostituzione. Questi strumenti hanno consentito la conduzione di una serie di esperimenti volti a esplorare largamente lo spazio delle soluzioni, al fine di migliorare i risultati ottenuti. L'innovazione più significativa introdotta in questa tesi è l'introduzione di un approccio basato sul Machine Learning per introdurre dinamicità alla soglia mutativa, che ha portato alla generazione di soluzioni notevolmente superiori rispetto alla versione originale dell'algoritmo.

Come illustrato nel capitolo “*Risultati*”, la discrepanza percentuale riportata dalla prima implementazione dell'algoritmo è significativamente superiore rispetto alla versione finale che fa uso di un approccio di Machine Learning. Questa differenza è attribuibile al fatto che la versione iniziale dell'algoritmo si basava su un approccio mutativo statico, vale a dire su soglie di mutazione fisse e predefinite. Al contrario, la nuova versione proposta adotta un approccio dinamico che ha permesso di superare in modo più efficace i punti di stallo derivanti dal raggiungimento di ottimi locali.

Guardando al futuro, le possibili estensioni di questo lavoro potrebbero includere la ricerca di parametri più efficienti per l'implementazione dell'approccio basato su Machine Learning. Allo stesso tempo, potrebbero essere prese in considerazione alcune ottimizzazioni del metodo di local search. Nonostante si sia dimostrato fondamentale per trovare soluzioni sempre migliori, questo metodo ha un costo

computazionale significativo che ha avuto un impatto considerevole sulle fasi di testing.

---

# Bibliografia

- [1] C. M. Bishop and N. M. Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.
- [2] C. Cavallaro, V. Cutello, M. Pavone, E.-g. Talbi, and F. Zito. Machine learning and genetic algorithms: a case study on image reconstruction. *Technical Report*, 2023.
- [3] V. Cutello, M. Oliva, M. Pavone, and R. A. Scollo. An immune metaheuristics for large instances of the weighted feedback vertex set problem. In *2019 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1928–1936. IEEE, 2019.
- [4] V. Cutello, M. Oliva, M. Pavone, and R. A. Scollo. A hybrid immunological search for the weighted feedback vertex set problem. In *Learning and Intelligent Optimization: 13th International Conference, LION 13, Chania, Crete, Greece, May 27–31, 2019, Revised Selected Papers 13*, pages 1–16. Springer, 2020.
- [5] G. Folino. Algoritmi evolutivi e programmazione genetica: strategie di progettazione e parallelizzazione. *ICARCNr*, 2003.
- [6] S. Jalili. Cultural algorithms: Recent advances. *Springer Nature*, 2022.
- [7] A. Lazzeri. Algoritmi evolutivi: cosa sono, in quali casi possono essere applicati. *ai4business.it*, 2020.
- [8] R. Li, C.-Y. Lin, W.-F. Guo, and T. Akutsu. Weighted minimum feedback vertex sets and implementation in human cancer genes detection. *BMC bioinformatics*, 22(1):1–17, 2021.
- [9] F. Neri, C. Cotta, and P. Moscato. *Handbook of memetic algorithms*, volume 379. Springer, 2011.

- 
- [10] A. G. Tettamanzi et al. Algoritmi evolutivi: concetti ed applicazioni. *Mondo digitale*, 2005(1):3–17, 2005.
  - [11] Y. Xu, J. Pan, and H. Xie. Minimum vertex covering and feedback vertex set-based algorithm for influence maximization in social network. *ResearchGate*, 38:795–802, 04 2016. doi: 10.11999/JEIT160019.
  - [12] D. Zhao, L. Xu, S.-M. Qin, G. Liu, and Z. Wang. The feedback vertex set problem of multiplex networks. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 67(12):3492–3496, 2020. doi: 10.1109/TCSII.2020.2997974.