Solving the Weighted Feedback Vertex Set Problem using a Hybrid Genetic Algorithm

Francesco Granata
Università di Catania
1000046547

10 Marzo 2025

Indice

1 Introduzione

1.1 Definizione

Il *Weighted Feedback Vertex Set Problem* (WFVS) è un problema di ottimizzazione combinatoria che consiste nel trovare un sottoinsieme di vertici il cui rimuovimento renda il grafo aciclico, minimizzando la somma dei pesi dei nodi rimossi.

1.2 Definizione Formale

Formalmente, dato un grafo non orientato $G = (V, E)$, un insieme di vertici di feedback di G è un sottoinsieme $S \subseteq V$ di vertici la cui rimozione, insieme a tutti i lati incidenti in S, rende il grafo aciclico, ovvero $G(V \setminus S, E \setminus S)$ è un grafo aciclico.

Inoltre, sia $w : V \to \mathbb{R}^+$ una funzione peso che associa un valore positivo a ciascun vertice $v \in V$ in G. Il peso dell'insieme di vertici di feedback $S \subseteq V$ è la somma dei pesi dei suoi vertici, cioè $\sum_{v \in S} w(v)$.

Risolvere il problema del *Weighted Feedback Vertex Set* richiede di trovare un insieme S di minimo peso, ovvero:

$$\min_{S} \sum_{v \in S} w(v) \tag{1}$$

1.3 Complessità Computazionale

Il *WFVS* è un problema *NP-Hard*, quindi è impossibile trovare una soluzione che di per certo è ottima in tempo polinomiale: Per questo motivo siamo costretti a utilizzare tecniche euristiche e metaeuristiche che approssimano la soluzione in tempi polinomiali.

1.4 Applicazioni

Questo problema ha importanti applicazioni:

- **Teoria dei Grafi**: Rimuove cicli per ottimizzare reti o risolvere dipendenze circolari.
- **Intelligenza Artificiale**: Garantisce che reti bayesiane siano acicliche, migliorando algoritmi di apprendimento.
- **Biologia Computazionale**: Analizza reti metaboliche e interazioni proteiche, identificando nodi chiave per interrompere cicli.

- **Ingegneria dei Sistemi**: Stabilizza sistemi di controllo e ottimizza processi industriali rimuovendo cicli.
- **Informatica Teorica**: È un problema NP-difficile, studiato per sviluppare algoritmi di approssimazione euristici.
- **Applicazioni Pratiche**: Migliora sistemi di raccomandazione e gestione di progetti eliminando conflitti ciclici.

2 Algoritmo Sviluppato

Per affrontare questo problema, è stato sviluppato un Algoritmo Genetico Ibrido **(HGA - Hybrid Genetic Algorithm)**, che combina le caratteristiche dei **Genetic Algorithms (GA)** con una fase di **ricerca locale** per migliorare la qualità delle soluzioni trovate. Gli Algoritmi Genetici sono ispirati alla selezione naturale e si basano su operatori di selezione, crossover e mutazione per esplorare lo spazio delle soluzioni. Tuttavia, una pura strategia evolutiva potrebbe soffrire di convergenza prematura o stagnazione su ottimi locali.

L'approccio ibrido integra quindi un meccanismo di **local search** che affina le soluzioni generate dal processo evolutivo, migliorando progressivamente la qualità dei candidati senza perdere la diversità della popolazione. Grazie a questa combinazione, l'algoritmo sfrutta la capacità esplorativa degli Algoritmi Genetici e la forza di intensificazione della ricerca locale, garantendo un compromesso efficace tra esplorazione ed exploitation dello spazio delle soluzioni.

In generale le fasi di questo algoritmo sono riassunte in questi punti:

- **Inizializzazione della popolazione**: Viene generata una popolazione iniziale di soluzioni casuali.
- **Fitness**: Per ogni individuo della popolazione, viene calcolata la sua funzione di valutazione (fitness) che potrebbe cambiare in base al problema.
- **Selezione dei genitori**: Viene utilizzata una strategia di selezione che prende un sottoinsieme della popolazione. Gli elementi selezionati saranno i padri della nuova popolazione. La scelta è basata sulla fitness.
- **Crossover**: Gli individui selezionati subiscono un crossover, che combina le caratteristiche dei genitori per generare nuovi individui (figli).

- **Mutazione**: Con una certa probabilità avvengono mutazioni alla popolazione. Serve a mantenere diversità nella popolazione e previene una convergenza prematura verso i minimi locali.
- **Ricerca locale**: Attraverso una ricerca locale si cerca di migliorare le soluzioni trovate per convergere prima verso la giusta soluzione.
- **Aggiornamento della popolazione**: Si prende la nuova popolazione creata e si integra con quella vecchia nel modo più opportuno.
- **Criterio di terminazione**: Un ciclo dei punti sopra elencati viene fatto fin quando, secondo un certo criterio di terminazione, viene interrotto e si ritorna la soluzione

Nel seguito del documento, verranno approfonditi i dettagli dell'implementazione, le scelte progettuali e le metodologie utilizzate per garantire prestazioni ottimali nell'identificazione dell'insieme di vertici di feedback a peso minimo.

3 Implementazione

L'algoritmo è stato implementato in Python, utilizzando la libreria NetworkX per la gestione dei grafi. Il codice è strutturato in modo modulare. Ecco un approfondimento sul come è stato deciso di sviluppare l'HGA.

3.1 Inizializzazione della popolazione

Crea una popolazione iniziale di soluzioni random. la popolazione è composta da vettori binari lunghi quanto il numero dei nodi del grafo su ogni vettore creato chiamiamo una local search in modo da ottimizzarlo subito.

La grandezza della popolazione scelta è:

```
pop_size = 100
```

La scelta è derivata dal non voler utilizzare una grandezza della popolazione troppo bassa (50) ma comunque tenersi sul medio-basso per limitare i tempi di esecuzione.

Algorithm 1 initialize population

Require: pop_size, num_nodes, graph
Ensure: population
1: population ← []
2: **for** $i = 1$ to pop_size **do**
3: solution ← vettore binario casuale di lunghezza num_nodes
4: solution ← LOCAL_SEARCH(solution, graph)
5: population.append(solution)
6: **end for**
7: **return** population

3.2 Fitness

La Fitness da calcolare per il problema del WFVS è la somma dei pesi dei vertici selezionati.

- Prende la soluzione inviata e crea un sottografo senza i nodi selezionati.
- Ritorna la somma dei pesi della soluzione.
- Se il grafo non è aciclico tornerà la fitness della soluzione con una penitenza calibrata in base al numero di cicli. Questo per penalizzare le soluzioni che hanno più cicli rispetto ad altre.

Algorithm 2 fitness

Require: solution, node_weights, graph
Ensure: fitness + pensalità in caso di cicli
1: Incrementa il contatore delle valutazioni *evaluation_count*
2: Determina l'insieme dei nodi da rimuovere:
3: $nodes\_to\_remove \leftarrow \{v \mid v \in V, solution[v] = 1\}$
4: Crea il sottografo senza i nodi selezionati: $subgraph \leftarrow graph - nodes\_to\_remove$
5: Conta il numero di cicli nel sottografo: $num\_cycles \leftarrow$ numero di cicli in $subgraph$
6: Calcola il valore della fitness:
7: $fitness \leftarrow \sum_{v \in nodes\_to\_remove} node\_weights[v] + 10000 \cdot num\_cycles$
return $fitness$

3.3 Selezione dei genitori

Per selezionare i genitori destinati alla riproduzione, è stata utilizzata la *k-Tournament Selection*, che funziona come segue:

- Si selezionano casualmente k individui dalla popolazione.
- Si confrontano i valori di fitness e si sceglie il migliore come genitore.

Come k si è scelto di utilizzare:

```
k = 5
```

per trovare più velocemente buone soluzioni.

Algorithm 3 tournament selection

Require: population, fitnesses (fitness di tutta la popolazione)
Ensure: soluzione vincente
1: Seleziona casualmente k individui dalla popolazione
2: Estrai i corrispondenti valori di fitness
3: Determina il miglior individuo tra i k selezionati (quello con fitness minima)
4: **return** soluzione vincente

3.4 Crossover

Come Crossover è stato utilizzato un *Three-Point Crossover* il cui funzionamento è:

- Si selezionano tre punti casuali nei cromosomi dei genitori.
- Si ordinano i punti
- Vengono scambiati i bit dei parent tra il punto 1 e il punto 2 e quelli dopo il punto 3
- I due nuovi vettori saranno i child da tornare.

Algorithm 4 Crossover

Require: parent1, parent2
Ensure: child1, child2
1: Seleziona tre punti distinti casuali e ordinali in ordine crescente
2: Copia *parent1* in *child1* e *parent2* in *child2*
3: Scambia i segmenti tra il primo e il secondo punto tra i genitori
4: Scambia i segmenti dopo il terzo punto tra i genitori
5: **return** *child1*, *child2*

3.5 Mutazione

Per mantenere la diversità nella popolazione, si applica una mutazione con probabilità P_m.

Come P_m si è scelto di utilizzare:

```
mutation_rate = 0.01
```

per evitare di cadere in ottimi locali.

Algorithm 5 mutate

Require: Una soluzione *solution* (array di bit) e un tasso di mutazione *mutation_rate*
Ensure: Una nuova soluzione mutata
1: Genera una maschera di mutazione casuale, dove ogni bit ha probabilità *mutation_rate* di essere selezionato
2: Inizializza una lista vuota per la soluzione mutata
3: **for** ogni bit nella soluzione originale **do**
4: **if** il bit è stato selezionato dalla maschera di mutazione **then**
5: Inverti il bit (da 0 a 1 o da 1 a 0) e aggiungilo alla soluzione mutata
6: **else**
7: Aggiungi il bit originale alla soluzione mutata
8: **end if**
9: **end for**
10: **return** la soluzione mutata

3.6 Ricerca locale

Viene implementata una ricerca locale di tipo greedy per ottimizzare una soluzione rappresentata come un array di bit

L'obiettivo è rimuovere nodi selezionati dalla soluzione, mantenendo la proprietà di aciclicità del grafo. Per evitare un numero eccessivo di iterazioni, viene utilizzato un limite massimo di iterazioni (*max_iter*).

Si è scelto come valore di *max_iter*:

```
max_iter = 30
```

Algorithm 6 local_search

Require: Una soluzione *solution* (array di bit) e un grafo *graph*
Ensure: Una soluzione migliorata *improved_solution*

```
Imposta un limite massimo di iterazioni max_iter ← 30
Inizializza un contatore count ← 0
Copia la soluzione originale in improved_solution
for ogni bit nella soluzione originale do
    if il bit è uguale a 1 then
        Imposta il bit a 0 in improved_solution
        if il grafo graph non è aciclico con improved_solution then
            Ripristina il bit a 1 in improved_solution
        end if
    end if
    Incrementa count di 1
    if count ≥ max_iter then
        Interrompi il ciclo
    end if
end for
return improved_solution
```

3.7 Aggiornamento della popolazione

Per aggiornare in modo opportuno la popolazione si è fatto nel seguente modo:

- Viene aggiunta la nuova popolazione a quella vecchia
- Si ordina rispetto alla fitness
- Si riduce il vettore della popolazione a *pop_size*

3.8 Criterio d'arresto

Si è impostato il criterio di arresto a:

```
generations = 1001
```

4 Analisi Sperimentale

4.1 Protocollo Sperimentale

Sono stati condotti esperimenti su un totale di 30 istanze del problema, con grafi non orientati di tipi e dimensioni variabili.

6 grafi diversi, per ognuno 5 istanze in cui varia la funzione peso w. In particolare questi 6 grafi sono grafi con 2 topologie diverse: 3 a griglia e 3 casuali.

Il protocollo utilizzato:

- Su ogni istanza sono state fatte 10 Run diverse per trovare la soluzione minima.
- I risultati di ogni istanza sono stati inseriti in un dataframe creato con la libreria Python "*Pandas*", poi salvati in un file *.xlsx*.
- Per ogni grafo si calcola la media, la deviazione standard e la media del numero di valutazioni della fitness sui minimi delle 5 istanze.

4.2 Risultati Sperimentali

La seguente tabella contiene i risultati sperimentali raccolti seguendo il protocollo precedentemente descritto.

Tipo Grafo	Media	Dev. Std.	Media Valutazioni Fitness
Grid_5_5	199,8	21,8	201301.0
Grid_7_7	254,8	5,8	201301.0
Grid_9_9	1184,4	134,9	201301.0
Rand_100_3069	1139,6	19,8	201301.0
Rand_100_841	1794,2	127,9	201301.0
Rand_200_3184	5366,0	867,2	201301.0

Tabella 1: Risultati sperimentali.

4.3 Grafi e Grafici

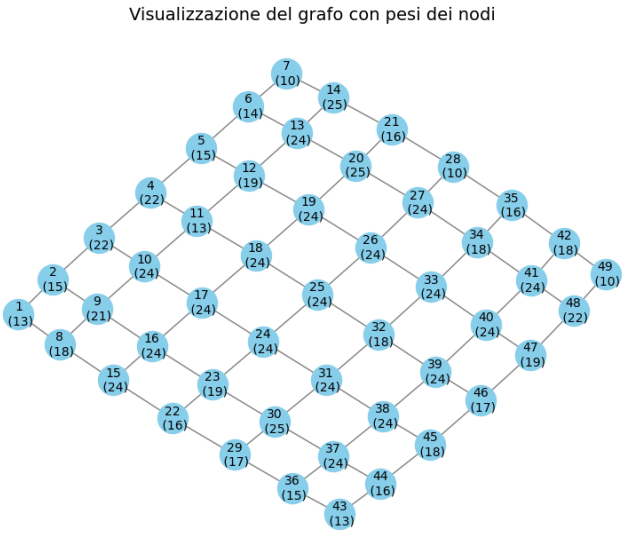

Figura 1: Grafo di tipo Grid_7_7 (INPUT)

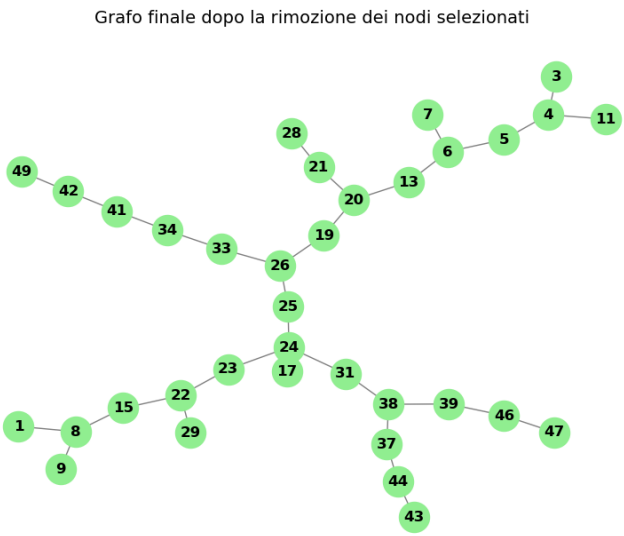

Figura 2: Grafo di tipo Grid_7_7 (OUTPUT)

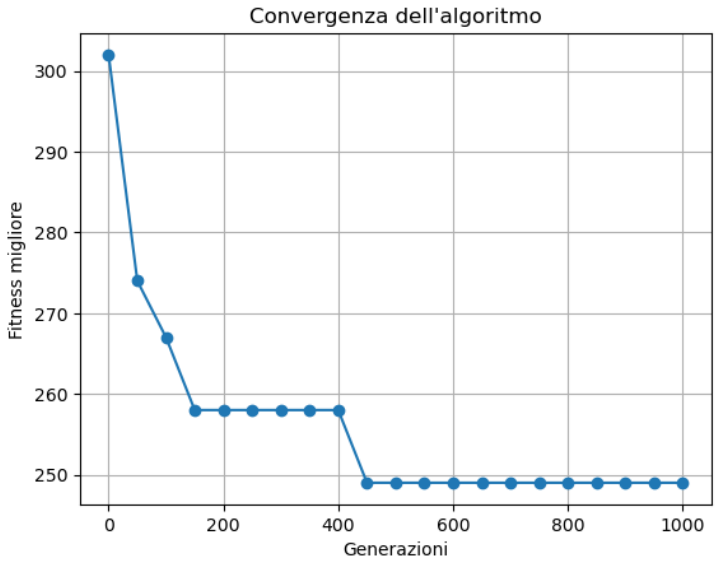

Figura 3: Grafo di tipo Grid_7_7 (CONVERGENZA FITNESS)

5 Conclusioni

In questo lavoro abbiamo affrontato il **Weighted Feedback Vertex Set Problem (WFVS)**, un problema **NP-Hard** con numerose applicazioni pratiche, sviluppando e testando un **Hybrid Genetic Algorithm (HGA)** per trovare soluzioni approssimate di alta qualità.

Possiamo dire che:

- L'analisi delle diverse istanze ha evidenziato come la topologia del grafo e la distribuzione dei pesi influiscano sulla difficoltà del problema, mostrando che l'HGA si adatta bene a scenari diversi. Inoltre, il protocollo sperimentale ha permesso di valutare in modo oggettivo le performance dell'algoritmo, evidenziando la stabilità dei risultati ottenuti su più esecuzioni indipendenti.

- Si nota che per i grafi più semplici l'algoritmo converge dopo poche iterazioni, mentre per istanze più grandi sono necessarie tutte e 1000 le generazioni.

- Il compromesso tra qualità delle soluzioni e tempo di esecuzione è pressochè buono.

- Bilancia esplorazione e sfruttamento, combinando la diversità della ricerca genetica con la precisione della ricerca locale.

Nonostante i risultati promettenti ottenuti con l'HGA, vi sono diversi aspetti che potrebbero essere migliorati per ottimizzare ulteriormente l'algoritmo:

- **Ottimizzazione dei parametri**: una ricerca automatica dei parametri tramite tecniche come l'ottimizzazione bayesiana o l'evoluzione differenziale potrebbe migliorare le prestazioni.
- **Strategie avanzate di mutazione e crossover**: l'introduzione di operatori adattivi o basati su euristiche specifiche per il WFVS potrebbe portare a una convergenza più rapida e soluzioni di qualità superiore.
- **Miglioramento Local Search**: implementare un altro tipo di local search potrebbe rendere l'algoritmo più veloce o performante.
- **Ibridazione con altri metodi**: combinare l'HGA con tecniche come il simulated annealing o la programmazione matematica potrebbe aumentare l'efficienza nell'esplorare lo spazio delle soluzioni.
- **Parallelizzazione**: implementare il calcolo parallelo per la valutazione della fitness o per l'esecuzione delle ricerche locali permetterebbe di ridurre significativamente i tempi di esecuzione, rendendo l'algoritmo applicabile a istanze ancora più grandi.

L'algoritmo sviluppato si è dimostrato un metodo efficace per affrontare il WFVS, fornendo soluzioni competitive in tempi ragionevoli. L'analisi sperimentale ha confermato la robustezza dell'approccio, bilanciando esplorazione e sfruttamento per migliorare progressivamente le soluzioni. Grazie alla sua flessibilità, l'HGA potrebbe essere ulteriormente adattato e migliorato, aprendo la strada a nuove ricerche nel campo dell'ottimizzazione combinatoria.

Riferimenti bibliografici

[1] John E. Beasley. An overview of genetic algorithms: Part 1, fundamentals. *University Computing*, 15(2):58–69, 1993. Una panoramica sugli algoritmi genetici.

[2] Mario Pavone. Intelligenza artificiale. Materiale didattico, 2024. Slides del corso di Intelligenza Artificiale, Università degli studi di Catania.