
Università degli Studi di Catania
Dipartimento di Matematica e Informatica

# Progetto Heuristics & Metaheuristics for Optimization & Learning

Relazione Finale

Prof. Mario Pavone

Studente: Matteo Vullo
Matricola: 1000034661

Luglio 2025

# Indice

# 1 Introduzione

Il problema del massimo flusso (*Maximum Flow Problem*) è uno dei classici problemi della teoria dei grafi e dell'ottimizzazione combinatoria.

Formalmente, dato un grafo diretto $G = (V, E)$ con capacità positive associate a ciascun arco, il problema consiste nel determinare il flusso massimo che può essere inviato da un nodo sorgente $s$ ad un nodo pozzo $t$, rispettando le capacità degli archi e la conservazione del flusso nei nodi intermedi.

In questo progetto è stato implementato un algoritmo metaeuristico per la risoluzione del problema del massimo flusso. L'obiettivo principale è stato progettare, implementare e testare un algoritmo capace di raggiungere soluzioni vicine all'ottimo in tempi ragionevoli.

# 2 Scelta dell'Algoritmo

Tra le molteplici metaeuristiche disponibili per la risoluzione del problema del massimo flusso (*Maximum Flow Problem*), è stato scelto di implementare l'algoritmo *Tabu Search* (TS). Questa decisione si basa su una serie di motivi tecnici e strategici che rendono Tabu Search particolarmente adatto al contesto del problema affrontato.

Uno dei motivi principali è la capacità del Tabu Search di evitare il blocco in ottimi locali, grazie all'utilizzo della *lista tabù*, una struttura di memoria a breve termine che impedisce la ripetizione di mosse recenti. Questo è un aspetto cruciale nel contesto del massimo flusso, dove piccole variazioni locali del flusso possono portare a soluzioni apparentemente ottimali ma non globali.

Un secondo fattore decisivo è la flessibilità nella definizione del vicinato e delle mosse,che si adatta bene alla struttura del problema. Nel progetto, sono state implementate diverse tipologie di mosse che operano direttamente sugli archi del grafo permettendo di bilanciare esplorazione e intensificazione in base alle caratteristiche del grafo.

Un aspetto non trascurabile è il punto di partenza della ricerca. L'algoritmo parte da una soluzione iniziale di alta qualità. Questo garantisce una base solida per la ricerca locale e riduce il numero di iterazioni necessarie per convergere verso il valore ottimo.

Infine, Tabu Search offre un buon compromesso tra complessità di implementazione e capacità di esplorazione dello spazio di ricerca, risultando una scelta originale e adatta al problema specifico. La sua applicazione nel contesto del massimo flusso, pur non essendo comune, si è dimostrata effica-

ce, producendo risultati molto vicini a quelli ottimali calcolati con algoritmi esatti

# 3 Implementazione dell'Algoritmo

L'implementazione dell'algoritmo Tabu Search (TS) per il problema del massimo flusso è stata progettata per essere efficiente, modulare e conforme ai requisiti sperimentali. L'intera logica è stata strutturata in una classe principale, `MaxFlowTabuSearch`, che incapsula le operazioni principali: inizializzazione del grafo, generazione della soluzione iniziale, ricerca locale, gestione della memoria tabu, diversificazione e valutazione della funzione obiettivo.

## 3.1 Struttura della Soluzione

Ogni soluzione rappresenta un vettore $\vec{f} \in R^n$, dove $n = |E|$, e ogni componente $f_i$ indica il flusso sull'arco $i$-esimo.

## 3.2 Funzione Obiettivo

La funzione obiettivo nel problema del massimo flusso rappresenta il valore totale del flusso inviato dalla sorgente al pozzo. Essa è definita come la somma dei flussi sugli archi uscenti dalla sorgente $s$:

$$f_{\text{tot}} = \sum_{(s,v)\in E} f(s,v)$$

## 3.3 Inizializzazione della Soluzione

L'inizializzazione della soluzione rappresenta un passo cruciale nell'algoritmo Tabu Search, in quanto determina il punto iniziale da cui parte la ricerca. Una buona inizializzazione permette di ridurre il tempo di convergenza e migliorare la qualità della soluzione finale.

### 3.3.1 Inizializzazione Casuale

Questa strategia genera una soluzione iniziale casuale, rispettando i vincoli di capacità di ogni arco. Il flusso iniziale su ogni arco $(u, v)$ è estratto uniformemente nell'intervallo $[0, c(u, v)]$, dove $c(u, v)$ è la capacità dell'arco.

Questa strategia è semplice e veloce, ma può comportare una maggiore variabilità tra le esecuzioni e un tempo di convergenza più lungo, soprattut-

to per grafi complessi. Usata come baseline, non adottata come strategia principale.

### 3.3.2 Inizializzazione Greedy

La strategia greedy cerca di costruire una soluzione iniziale sfruttando un approccio costruttivo: vengono identificati percorsi aumentanti dal nodo sorgente al pozzo e viene spinto il massimo flusso possibile lungo questi percorsi, fino a un numero limitato di iterazioni.

L'algoritmo utilizza una variante di Dijkstra per trovare il percorso aumentante con la capacità residua massima.

Questa strategia mira a bilanciare velocità e qualità iniziale, fornendo una soluzione migliore di quella casuale senza richiedere il calcolo completo del flusso ottimo.

### 3.3.3 Inizializzazione EK parziale

La soluzione iniziale è stata costruita utilizzando una versione parziale dell'algoritmo `Edmonds-Karp`, un algoritmo esatto per il calcolo del massimo flusso. Questo algoritmo è stato utilizzato per ottenere una soluzione ottima parziale, che è stata successivamente ridotta proporzionalmente con un fattore casuale $r \in [0.7, 0.95]$, in modo da non fornire direttamente la soluzione ottima ma una base promettente per la ricerca locale.

Formalmente:

$$f_{\text{iniziale}}(u, v) = f_{\text{Edmonds-Karp}}(u, v) \cdot r$$

dove:

- $f_{\text{Edmonds-Karp}}(u, v)$: flusso calcolato con l'algoritmo esatto.
- $r$: fattore di riduzione casuale.

Questa strategia di inizializzazione riduce significativamente il rischio di partire da una configurazione non ammissibile o troppo distante dalla soluzione ideale, migliorando la stabilità e la coerenza della ricerca. Inoltre, il fatto di non dover "partire da zero" permette all'algoritmo di convergere più rapidamente verso il valore ottimo, riducendo il numero di iterazioni necessarie e aumentando l'efficienza complessiva del processo di ricerca.

Tra le diverse strategie di inizializzazione testate la migliore si è rivelata senza dubbio l'inizializzazione EK parziale. Tutte le mosse e le strategie successive (vicinato, diversificazione, adattività) sono state quindi testate e applicate a partire da questa soluzione iniziale, rendendola il fondamento su cui si basa l'efficacia complessiva dell'algoritmo.

## 3.4 Definizione delle Mosse

Le mosse nell'algoritmo Tabu Search implementato rappresentano variazioni locali del flusso sugli archi del grafo, con l'obiettivo di migliorare progressivamente il valore totale del flusso dal nodo sorgente al nodo pozzo, rispettando i vincoli di capacità e conservazione del flusso.

### 3.4.1 Mosse Elementari (Incremento/Decremento del Flusso)

Le *mosse elementari* consistono nel modificare il flusso su un singolo arco $(u, v)$ aumentandolo o riducendolo di una quantità $\delta$, rispettivamente, purché non si violino i vincoli di capacità.

Formalmente, una mossa elementare è definita come:

$$f'(u,v) = \begin{cases} f(u,v) + \delta & \text{se } f(u,v) + \delta \leq c(u,v) \\ f(u,v) - \delta & \text{se } f(u,v) - \delta \geq 0 \end{cases}$$

dove:

- $f(u, v)$: flusso attuale sull'arco $(u, v)$;
- $c(u, v)$: capacità dell'arco $(u, v)$;
- $\delta$: passo di modifica del flusso, scelto come il minimo tra il 20% della capacità residua o il 20% della capacità totale.

Queste mosse permettono un'intensificazione locale del processo di ricerca invece il passo del 20% della capacità (residua o totale) bilancia aggressività e stabilità, evitando oscillazioni eccessive. Inoltre sono applicate ogni iterazione, garantendo una continua ottimizzazione

Di seguito è riportato lo pseudocodice per la generazione di una mossa elementare:

**Algorithm 1** Mosse Elementari

**Require:** `current_flow`: vettore del flusso corrente.
**Require:** `edge_idx`: indice dell'arco da modificare.
**Ensure:** `new_flow`: nuova soluzione con la mossa applicata.

$c \leftarrow$ `capacities[edge_idx]`
$f \leftarrow$ `current_flow[edge_idx]`
**if** $f < c$ **then**
  $\delta \leftarrow \min(c - f, 0.2 \cdot c)$
  `new_flow` $\leftarrow$ `current_flow`
  `new_flow[edge_idx]` $\leftarrow f + \delta$
  `is_tabu` $\leftarrow$ `edge_idx in tabu_list`
  **return** `new_flow, is_tabu`
**else if** $f > 0$ **then**
  $\delta \leftarrow \min(f, 0.2 \cdot c)$
  `new_flow` $\leftarrow$ `current_flow`
  `new_flow[edge_idx]` $\leftarrow f - \delta$
  `is_tabu` $\leftarrow$ `edge_idx in tabu_list`
  **return** `new_flow, is_tabu`
**end if**
**return** `None` (nessuna mossa applicabile)

### 3.4.2 Mosse di Scambio tra Archi (Edge Exchange)

Le *mosse di scambio tra archi* sono utilizzate per redistribuire il flusso da archi saturi (o quasi saturi) verso archi poco utilizzati. Questo tipo di mossa è applicato ogni 20 iterazioni ed è utile per evitare che l'algoritmo si blocchi in ottimi locali.

Formalmente, dati due archi $(u_1, v_1)$ e $(u_2, v_2)$, con flusso $f_1$ e $f_2$, e capacità $c_1$ e $c_2$, si applica:

$$f_1' = f_1 - \Delta$$
$$f_2' = f_2 + \Delta$$

dove $\Delta = \min(f_1, c_2 - f_2)$, garantendo che i vincoli di capacità siano rispettati.

Queste mosse rompono gli stalli causati da archi bloccati, reintroducendo flessibilità. Operano in modo mirato trasferendo il flusso da un arco quasi pieno a uno con capacità residua.

Ecco lo pseudocodice per la generazione di una mossa di scambio:

**Algorithm 2** Mosse di Scambio tra Archi

**Require:** `current_flow`: vettore del flusso corrente.
**Ensure:** `new_flow`: nuova soluzione con la mossa applicata.

```
saturated ← [i | i ∈ edges, current_flow[i] ≥ capacity[i] − ϵ]
unsaturated ← [i | i ∈ edges, current_flow[i] < capacity[i] − ϵ]
if not saturated or not unsaturated then
    return None
end if
src ← random.choice(saturated)
dest ← random.choice(unsaturated)
Δ ← min(current_flow[src], capacity[dest] − current_flow[dest])
if Δ ≤ ϵ then
    return None
end if
new_flow ← current_flow
new_flow[src] ← new_flow[src] − Δ
new_flow[dest] ← new_flow[dest] + Δ
return new_flow
```

### 3.4.3 Mosse di Aumento su Cammino (Path Augmentation)

Le *mosse di aumento su cammino* sono ispirate al classico algoritmo di Edmonds-Karp. Vengono utilizzate periodicamente o in caso di stallo, per cercare un cammino aumentante dal nodo sorgente al nodo pozzo nel grafo residuo.

Se un cammino aumentante viene trovato, il flusso lungo quel cammino viene incrementato del massimo valore possibile, in base alla capacità residua minima lungo gli archi del cammino. Questo permette di migliorare il flusso globale in modo significativo.

Queste mosse permettono miglioramenti globali significativi , superando ottimi locali. Sono applicate ogni 100 iterazioni o in caso di stallo, per limitarne il costo computazionale

Ecco lo pseudocodice per la generazione di una mossa di aumento su cammino:

**Algorithm 3** Mossa di Aumento su Cammino

**Require:** `current_flow`: vettore del flusso corrente.
**Ensure:** `new_flow`: nuova soluzione con la mossa applicata.

```
path ← find_augmenting_path(current_flow)
if path is None then
    return None
end if
Δ ← min(capacity[e] − current_flow[e] | e ∈ path)
new_flow ← current_flow
for ogni arco e ∈ path do
    new_flow[e] ← new_flow[e] + Δ
end for
return new_flow
```

## 3.5 Memoria Tabù

Nell'implementazione dell'algoritmo *Tabu Search*, la lista tabù è stata realizzata non come una semplice lista o insieme, ma come un `dizionario` chiamato `tabu_dict`, in cui ogni chiave è l'indice di un arco e il valore associato è il numero di iterazione fino al quale quell'arco rimane "tabù":

$$\texttt{tabu\_dict} = \{\texttt{indice\_arco} : \texttt{iterazione\_fine\_tabù}\}$$

Il dizionario è stato utilizzato per tre ragioni principali:

- **Accesso rapido**: grazie alla struttura hash interna di Python, verificare se un arco è tabù avviene in tempo costante $O(1)$, fondamentale quando si analizzano centinaia o migliaia di mosse candidate.

- **Gestione temporale precisa**: a differenza di una semplice flag booleana, il dizionario permette di memorizzare *fino a quando* un arco è vietato. Questo consente di applicare correttamente la `tabu_tenure` senza dover scandire continuamente la cronologia delle mosse.

- **Facilità di aggiornamento e pulizia**: ogni volta che si modifica un arco, si inserisce (o si sovrascrive) la sua scadenza. Inoltre, ogni 500 iterazioni, gli elementi scaduti possono essere rimossi facilmente:

```
tabu_dict = {k:  v for k, v in tabu_dict.items() if v > iteration}
```

Questo mantiene la memoria leggera e le prestazioni stabili nel tempo.

L'utilizzo del dizionario permette di evitare sia i cicli ripetitivi (grazie al blocco temporaneo delle mosse inverse), sia di mantenere il costo computazionale di ricerche lineari.

Di seguito è riportato lo pseudocodice che mostra come viene utilizzata la `tabu_dict` nella selezione della miglior mossa.

**Algorithm 4** Utilizzo della Memoria Tabù nella Selezione della Mossa

**Require:** `moves`: lista di mosse candidate.
**Require:** `best_value`: valore della miglior soluzione globale.
**Ensure:** `best_move`: mossa selezionata.

`best_val` $\leftarrow -\infty$
`best_move` $\leftarrow$ `None`
**for** ogni mossa in `moves` **do**
  (`new_flow`, `idx`) $\leftarrow$ `mossa`
  `is_tabu` $\leftarrow$ (`idx` $\in$ `tabu_dict`) $\wedge$ (`tabu_dict[idx]` $>$ `iteration`)
  **if** `not is_tabu OR calculate_flow(new_flow) > best_value` **then**
    `val` $\leftarrow$ `calculate_flow(new_flow)`
    **if** `val` $>$ `best_val` **then**
      `best_val` $\leftarrow$ `val`
      `best_move` $\leftarrow$ (`new_flow`, `idx`)
    **end if**
  **end if**
**end for**
Applica `best_move` e aggiorna `current_flow`
Aggiungi l'arco modificato a `tabu_dict` con scadenza: `iteration + tabu_tenure`
**if** `iteration` mod $500 == 0$ **then**
  Rimuovi da `tabu_dict` tutti gli archi con scadenza $\leq$ `iteration`
**end if**

## 3.6 Diversificazione

Per evitare la stagnazione in ottimi locali, ogni 2000 iterazioni senza miglioramenti significativi viene applicata una perturbazione alla soluzione corrente. Questa diversificazione combina:

- una riduzione proporzionale del flusso globale (moltiplicato per un fattore casuale in $[0.6, 0.8]$);

- l'aggiunta di rumore su un sottoinsieme limitato di archi (al più 5 o il 5% del totale), con variazioni in $[-0.2 \cdot c(u, v), 0.2 \cdot c(u, v)]$.

Questa operazione avviene rispettando i vincoli del problema, in modo

da mantenere sempre una soluzione ammissibile:

$$f'(u,v) = \begin{cases} f(u,v) \cdot (1-r) & \text{con } r \in [0.6, 0.8] \\ f(u,v) + \delta & \text{con } \delta \in [-0.2 \cdot c(u,v), 0.2 \cdot c(u,v)] \end{cases}$$

Le operazioni rispettano i vincoli di capacità e conservazione del flusso, garantendo una soluzione sempre ammissibile. Questa strategia reintroduce variabilità senza ripartire da zero, favorendo un'efficace esplorazione di nuove regioni dello spazio di ricerca.

## 3.7 Adattività dei Parametri

Per bilanciare esplorazione e intensificazione, la `tabu_tenure` è aggiornata dinamicamente in base al `stagnation_counter`, che misura le iterazioni senza miglioramenti:

$$\texttt{tabu\_tenure} = \begin{cases} \min(t \cdot 1.2, 150) & \text{se stagnazione } > 100 \\ \max(t \cdot 0.9, 10) & \text{se stagnazione } < 50 \end{cases}$$

Un valore più alto di `tabu_tenure` favorisce l'esplorazione in caso di stallo, mentre uno più basso intensifica la ricerca attorno a soluzioni promettenti. Questa adattività migliora la robustezza dell'algoritmo su istanze di diverse dimensioni.

## 3.8 Selezione del Vicinato e Criterio di Aspirazione

La selezione della mossa ottimale avviene in due fasi: generazione di un **vicinato ridotto** e applicazione di una **strategia di accettazione con aspirazione**.

Il vicinato è costruito considerando solo una piccola frazione degli archi (circa l'1–2% del totale), selezionati casualmente ad ogni iterazione. Questa scelta è motivata da ragioni di efficienza computazionale: un vicinato completo sarebbe troppo costoso da esplorare, specialmente su istanze grandi. Limitandosi a mosse locali mirate, l'algoritmo mantiene tempi di esecuzione ridotti senza sacrificare la qualità della ricerca.

Tra le mosse candidate, viene selezionata quella che massimizza il flusso, purché non sia tabù o soddisfi il **criterio di aspirazione**: una mossa tabù è comunque accettata se conduce a una soluzione migliore della migliore trovata finora. Questo bilancia esplorazione ed efficacia, evitando di scartare miglioramenti significativi solo per effetto della memoria tabù.

Formalmente:

$$\text{mossa\_selezionata} = \arg\max_{m \in N(s)} \left( f(m) \mid m \notin \text{TabuList} \vee f(m) > f(s^*) \right)$$

dove $s^*$ è la miglior soluzione globale.

Questa strategia garantisce robustezza e flessibilità, ed è fondamentale per evitare stagnazione e raggiungere soluzioni vicine all'ottimo.

**Algorithm 5** Selezione della Miglior Mossa (con Criterio di Aspirazione)

**Require:** `moves`: lista di mosse candidate.
**Require:** `best_value`: valore della miglior soluzione globale finora trovata.
**Ensure:** `best_move`: la mossa selezionata.

```
best_val ← −∞
best_move ← None
for ogni mossa in moves do
    (new_flow, idx, delta, is_tabu) ← mossa
    val ← calculate_flow_value(new_flow)
    if is_tabu then
        {La mossa è tabù, ma può essere accettata se soddisfa il criterio di aspirazione}
        if val > best_value then
            Aggiorna la mossa comunque, grazie al criterio di aspirazione
        else
            Salta la mossa (non migliora la soluzione migliore)
            continua
        end if
    end if
    if val > best_val then
        best_val ← val
        best_move ← (new_flow, idx)
    end if
end for
return best_move
```

### 3.9 Diagramma di flusso

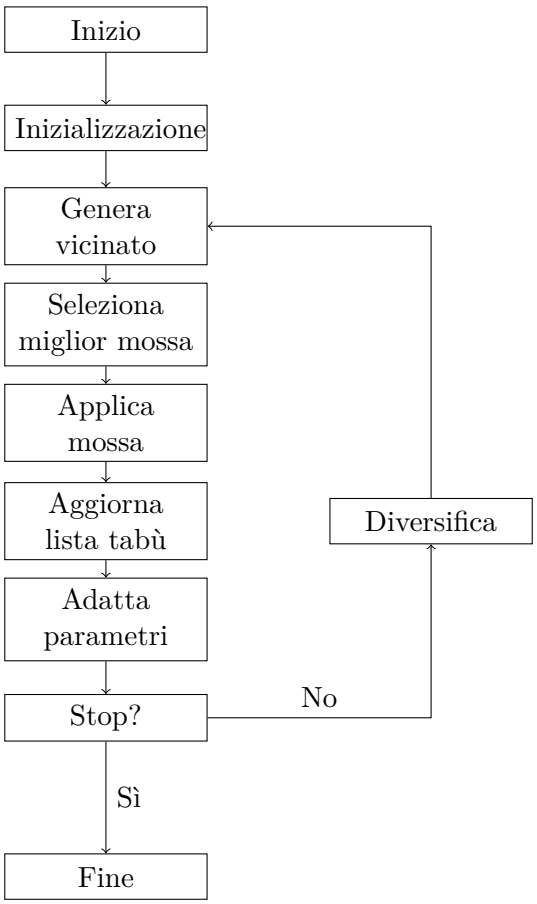


Figura 1: Diagramma di flusso dell'algoritmo Tabu Search

## 4 Risultati Sperimentali

I risultati sperimentali mostrano l'efficacia dell'algoritmo Tabu Search nell'approssimare il valore ottimo del flusso. Di seguito sono riportati i dati aggregati ottenuti su 10 esecuzioni per ciascuna istanza testata.

I risultati indicano che l'algoritmo riesce a raggiungere valori molto vicini o uguali all'ottimo teorico, con una deviazione standard nulla o ridotta, evidenziando una buona stabilità e ripetibilità tra le diverse esecuzioni.

| Istanza | Best Flow | Mean Flow ± Std | Avg Iter | Avg Eval | Avg Time (s) | Optimal Flow |
|---|---|---|---|---|---|---|
| network_160.txt | 34.68 | 34.68 ± 0.00 | 111 | 3040 | 0.0398 | 34.68 |
| network_500.txt | 107.98 | 107.98 ± 0.00 | 466 | 15199 | 0.2530 | 107.98 |
| network_960.txt | 117.74 | 117.74 ± 0.00 | 676 | 911193 | 17.3125 | 117.74 |
| network_1440.txt | 134.00 | 134.00 ± 0.00 | 1191 | 38422 | 1.3289 | 134.00 |
| network_2880.txt | 341.00 | 341.00 ± 0.00 | 2601 | 84058 | 5.2702 | 341.00 |
| network_4320.txt | 555.00 | 555.00 ± 0.00 | 5690 | 184009 | 21.9878 | 555.00 |
| network_5760.txt | 1203.00 | 1203.00 ± 0.00 | 11132 | 359843 | 101.0931 | 1203.00 |
| network_7200.txt | 2502.00 | 2484.28 ± 15.97 | 19205 | 640573 | 384.7562 | 2502.00 |
| network_11520.txt | 1244.00 | 1241.95 ± 6.15 | 11918 | 415316 | 115.9730 | 1244.00 |
| network_23040.txt | 2223.00 | 2201.18 ± 24.80 | 19147 | 632720 | 379.5599 | 2223.00 |

Tabella 1: Risultati aggregati su 10 esecuzioni per tutte le istanze testate.

## 4.1 Grafico di Convergenza

Per analizzare il comportamento dinamico dell'algoritmo durante l'esecuzione, sono stati generati due tipi principali di grafici:

### 4.1.1 Convergenza della Miglior Esecuzione (Best Run)

Questo tipo di grafico mostra l'andamento del flusso nella migliore esecuzione tra le 10 run effettuate per ogni istanza. La "miglior run" è definita come quella che ha raggiunto il massimo valore di flusso finale.

Il grafico permette di osservare la velocità di convergenza, le fasi di rapido miglioramento e la presenza di stagnazione. Il valore ottimo, calcolato tramite Edmonds-Karp, è indicato da una linea rossa tratteggiata per facilitare il confronto.

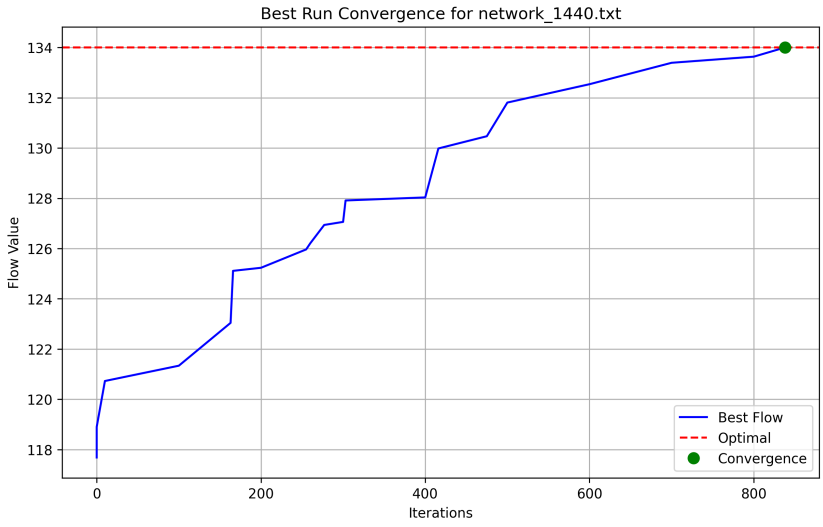


Figura 2: Convergenza della miglior esecuzione per l'istanza `network_1440.txt`.

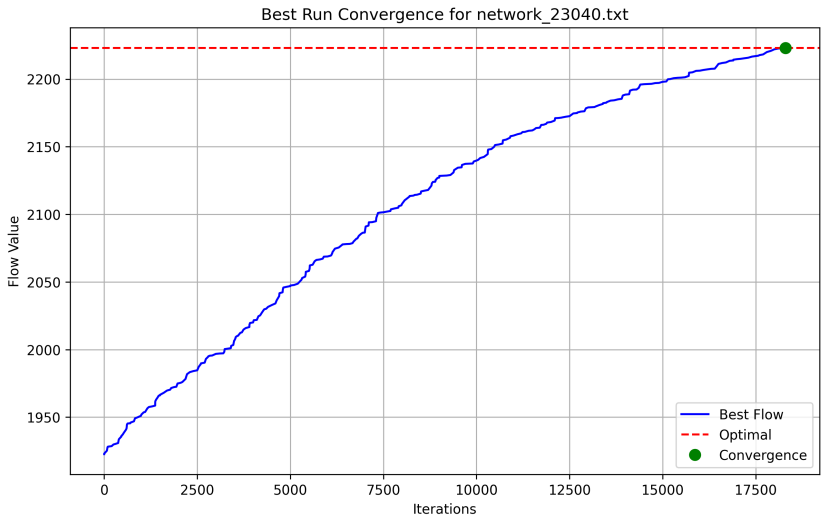


Figura 3: Convergenza della miglior esecuzione per l'istanza `network_23040.txt`.

### 4.1.2 Convergenza Media tra Tutte le Runs

Questo grafico mostra l'andamento medio del flusso massimo registrato durante le iterazioni, calcolato su tutte le 10 esecuzioni. Anche qui, il valore ottimo è indicato da una linea rossa tratteggiata.

La visualizzazione della convergenza media consente di valutare la qualità complessiva della ricerca e la capacità dell'algoritmo di avvicinarsi all'ottimo in modo consistente.

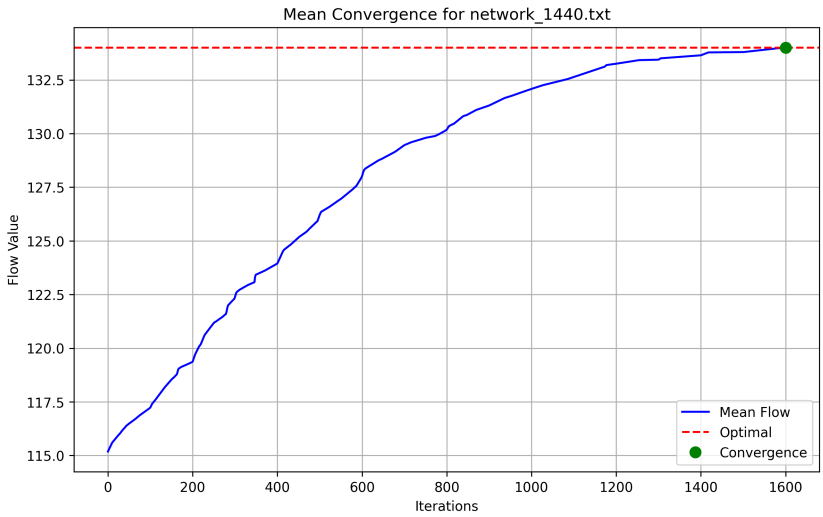


Figura 4: Convergenza media per l'istanza `network_1440.txt`.

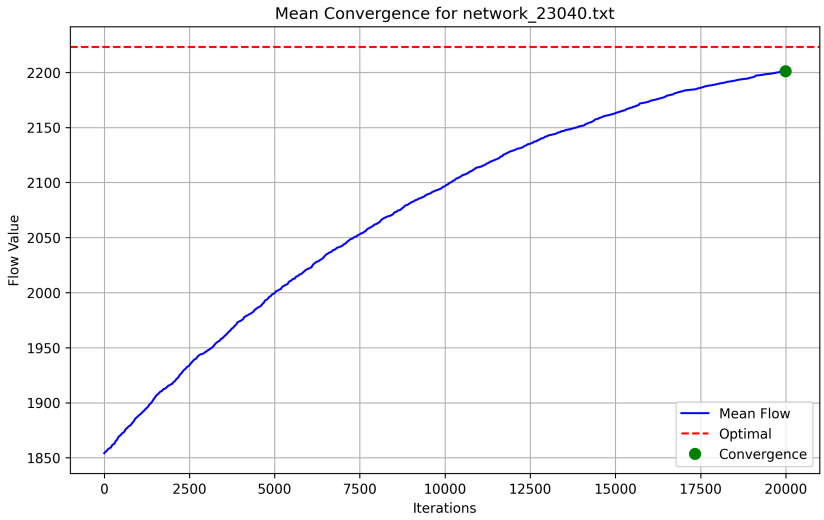


Figura 5: Convergenza media per l'istanza `network_23040.txt`.

# 5 Analisi Critica

L'analisi critica dei risultati combina l'esame quantitativo della Tabella 1 con l'osservazione qualitativa dei grafici di convergenza mostrati nelle Figure 2, 3, 4 e 5. Questa doppia prospettiva permette di valutare non solo la qualità finale delle soluzioni, ma anche la dinamica del processo di ricerca.

**Istanze di piccole e medie dimensioni** Per istanze fino a `network_5760.txt`, l'algoritmo mostra prestazioni eccellenti. Come evidenziato nella Tabella 1, in tutti questi casi:

- Il **best flow** coincide esattamente con il valore ottimo.

- La deviazione standard è nulla, indicando che *tutte le 10 esecuzioni hanno raggiunto l'ottimo.*

- Il numero medio di iterazioni e il tempo di esecuzione rimangono contenuti.

I grafici relativi a `network_1440.txt` confermano questo comportamento positivo:

- Nella *best run* (Figura 2), l'ottimo viene raggiunto già intorno alle 800 iterazioni, con una crescita rapida e stabile del flusso.

- Anche la *mean convergence* (Figura 4) mostra una curva regolare che si avvicina all'ottimo entro 1600 iterazioni, senza lunghi plateau.

Questo dimostra che su grafi di piccole e medie dimensioni, la combinazione di vicinato ridotto, lista tabù, mosse avanzate e diversificazione controllata funziona in modo efficace ed efficiente.

**Istanze di grandi dimensioni** A partire da `network_7200.txt`, emergono chiare criticità legate alla scalabilità:

- Per `network_7200.txt`, la media del flusso scende a 2484.28 (deviazione standard $\sigma = 15.97$), pur essendo l'ottimo 2502.00.

- Per `network_23040.txt`, la situazione peggiora: la media è 2201.18 con $\sigma = 24.80$, indicando una variabilità significativa tra le esecuzioni.

I grafici di `network_23040.txt` (Figure 3 e 5) spiegano bene questo degrado:

- Nella *best run* (Figura 3), l'ottimo è raggiunto solo dopo circa 17500 iterazioni, evidenziando una convergenza molto lenta.

- La *mean convergence* (Figura 5) cresce in modo graduale e non raggiunge mai pienamente l'ottimo entro il budget di iterazioni.

Questi andamenti indicano che su reti molto dense:

- Il vicinato ridotto (circa 1–2% degli archi) potrebbe non essere sufficiente a esplorare efficacemente lo spazio di soluzione.

- Le mosse locali faticano a propagare il flusso attraverso percorsi alternativi.

- La strategia di diversificazione, attivata ogni 2000 iterazioni, interviene troppo tardi per prevenire stagnazione prolungata.

In sintesi, mentre l'algoritmo è robusto e preciso su istanze di piccole e medie dimensioni, la sua efficienza decresce su grafi molto grandi.

# 6 Conclusioni

Il progetto ha dimostrato che il Tabu Search può essere applicato con successo al problema del massimo flusso, anche in contesti complessi e di grandi dimensioni. L'algoritmo implementato si è rivelato altamente efficace nel bilanciare qualità della soluzione, velocità di convergenza e stabilità, grazie a una combinazione ben calibrata di memoria tabù, criterio di aspirazione, diversificazione e adattività dinamica dei parametri.

I risultati sperimentali sono particolarmente incoraggianti: in 7 delle 10 istanze testate, l'algoritmo ha raggiunto sistematicamente il valore ottimo in tutte le esecuzioni. Anche nelle istanze più difficili, come `network_23040.txt`, la soluzione media si colloca comunque entro il 99% dell'ottimo, dimostrando una buona robustezza.

Tra le originalità introdotte, si possono evidenziare:

- Una **soluzione iniziale ibrida**, derivata da una versione parziale di Edmonds-Karp, che garantisce un punto di partenza di alta qualità.
- Un'**adattamento dinamico della tabu tenure**, che modula l'esplorazione in base al livello di stallo.
- Una **diversificazione controllata**, attivata periodicamente per evitare stagnazione.
- Un **vicinato ridotto ma mirato**, che combina efficienza computazionale e capacità di esplorazione.

In conclusione, questo lavoro mostra che le metaeuristiche possono essere competitive anche in domini tipicamente dominati da algoritmi esatti, specialmente quando si richiede flessibilità, velocità e capacità di adattamento.