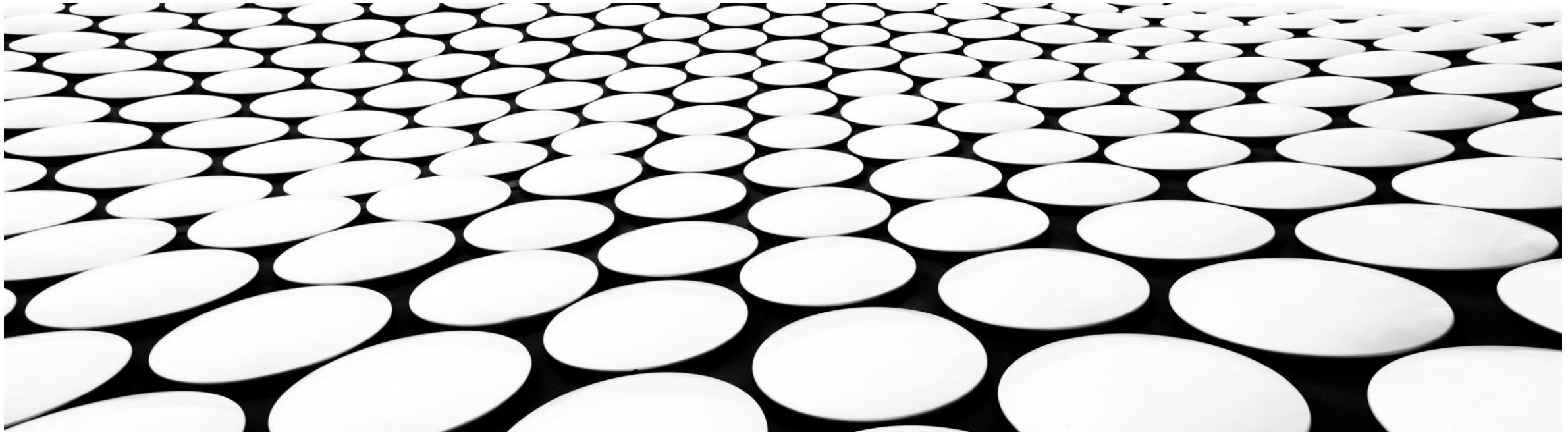
ALGORITMO *TABU SEARCH* PER LA SOLUZIONE DEL *PROBLEMA DEL COMMESSO VIAGGIATORE*



Gaetano Martedì M63001226

Biagio Salzillo M63001227

INTRODUZIONE

Un commesso viaggiatore deve visitare un certo numero di città.

Conosce tutte le distanze da una città all'altra e vuole determinare il percorso più breve che gli permetta di partire da una città e di farvi ritorno dopo aver visitato tutte le altre una sola volta.

Per cui, dato un grafo G(V, A) con N nodi (n ∈ V, che individuano le città da visitare) e archi pesati ∈ A (che individuano le distanze tra le varie città) bisogna trovare un percorso che sia:

- *un ciclo hamiltoniano* (*tour*):
  - ✓ inizia e finisce nello stesso nodo
  - ✓ passa per tutti i nodi una sola volta
- con peso totale (somma dei pesi) *minimo*

- ***FUNZIONE OBIETTIVO***

$$\min \sum_{(i,j)\in A} d_{ij}x_{ij}$$

- ***VINCOLI DI ASSEGNAMENTO (1)***

$$\sum_{i:(i,j)\in A} x_{ij} = 1 \quad j \in V$$

$$\sum_{i:(j,i)\in A} x_{ji} = 1 \quad j \in V$$

- ***VINCOLI DI ASSENZA DI SOTTOGIRI (2)***

$$\sum_{\substack{i\in S, j\in S: \\ (i,j)\in A}} x_{ij} \leq |S|-1 \quad \forall S \subset V : 2 \leq |S| \leq |V|-1$$

Variabili decisionali:

- x_{ij} = *1 se l'arco* (i,j) ϵ A *appartiene* al circuito hamiltoniano minimo,
- x_{ij} = *0* altrimenti

(1) In una soluzione ammissibile (circuito hamiltoniano) ogni nodo deve avere esattamente un arco entrante ed esattamente un arco uscente.

(2) In una soluzione ammissibile (circuito hamiltoniano) non ci possono essere cicli su un sottoinsieme proprio dell'insieme dei nodi V.

ALGORITMO DI TABU SEARCH PER IL TSP

Il Tabu Search è un algoritmo euristico che, se utilizzato in modo efficace, può ottenere una soluzione quasi ottimale per il TSP, in modo efficiente. La caratteristica principale di un algoritmo tabu search sta nella memorizzazione di informazioni relative alle iterazioni precedenti, che permettono di evitare che l'algoritmo vada in loop. Prima di individuare una nuova soluzione occorre verificare che questa non sia tabu consultando le informazioni relative al processo di ricerca. I passaggi di base applicati al TSP in questo elaborato sono presentati di seguito:

1. Rappresentazione della soluzione
2. Soluzione iniziale
3. Definizione di intorno e di mossa (2-opt)
4. Lista Tabu
5. Criterio di aspirazione
6. Diversificazione
7. Criterio di terminazione

Rappresentazione della soluzione

Una soluzione ammissibile è rappresentata come una sequenza di nodi, ed ognuno di essi appare una sola volta e nell'ordine in cui viene visitato. Il nodo di partenza è sempre fissato e nel nostro caso è il nodo 1.

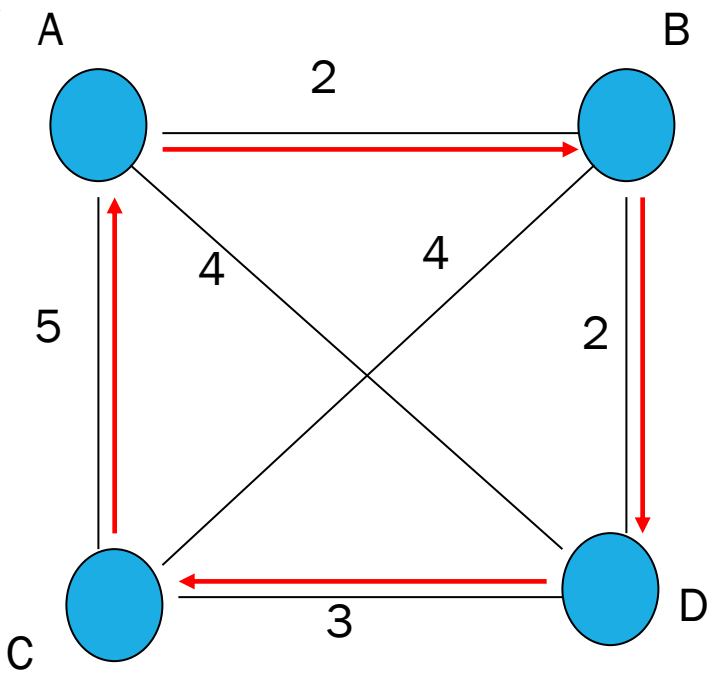
1	6	7	8	5	2	4	3

Soluzione iniziale

Una buona soluzione, ma non ottimale, per il TSP può essere trovata rapidamente utilizzando un approccio di tipo greedy. A partire dal primo nodo del tour, trova il nodo più vicino. Trova ogni volta il nodo non visitato più vicino al nodo corrente finché tutti i nodi non sono stati visitati (Algoritmo Nearest Neighbour).

ESEMPIO NEAREST NEIGHBOUR

NODO DI PARTENZA



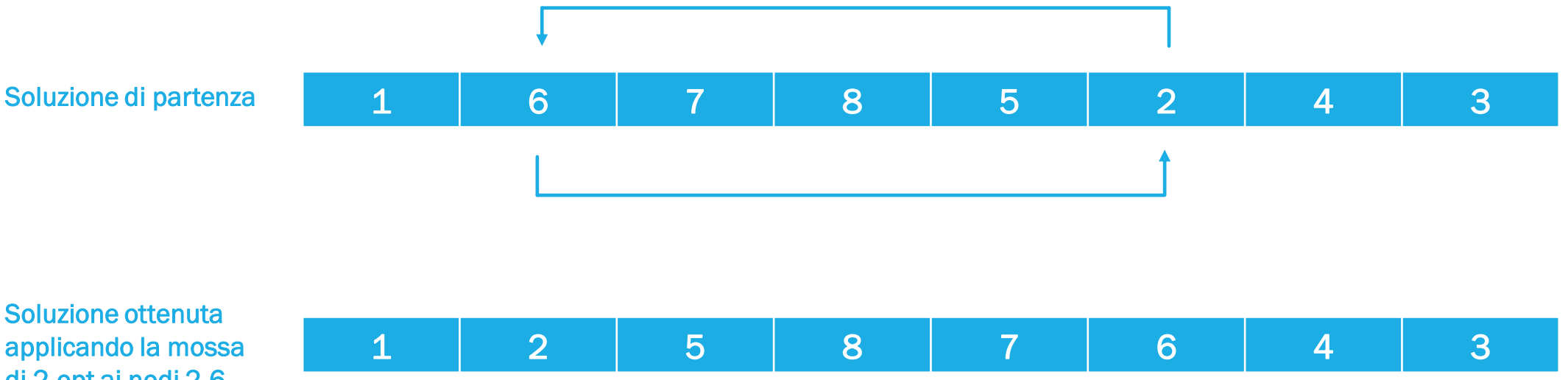
A	B	D	C

Definizione di intorno e di mossa (2-opt)

Un intorno, a una data soluzione, è definito come l'insieme di soluzioni ottenute invertendo tutte le possibili sequenze di nodi consecutivi, contenute all'interno della soluzione di partenza, che non contengano il nodo iniziale. Ciò garantisce che le nuove soluzioni generate siano sempre ammissibili (ovvero, non formino alcun sotto-giro). Se fissiamo il nodo 1 come nodo iniziale e nodo finale, per un problema di N nodi, ci sono

$$\frac{(N-1)(N-2)}{2}$$

tour ottenibili applicando 2-opt su una data soluzione. Ad ogni iterazione, viene selezionato il tour con il miglior valore della funzione obiettivo (distanza minima).



Pseudocodice

```
procedure 2optSwap(route, i, k) {
        take route[0] to route[i-1] and add them in order to new_route
        take route[i] to route[k] and add them in reverse order to new_route
        take route[k+1] to end and add them in order to new_route
        return new_route;
}
```

```
repeat until no improvement is made {
    best_distance = calculateTotalDistance(existing_route)
    start_again: for (i = 0; i <= number of nodes eligible to be swapped - 1; i++) {
                for (k = i + 1; k <= number of nodes eligible to be swapped; k++) {
                    new_route = 2optSwap(existing_route, i, k)
                    new_distance = calculateTotalDistance(new_route)
                    if (new_distance < best_distance) {
                        existing_route = new_route best_distance = new_distance
                        goto start_again
                    }
                }
        }
}
```

Ogni mossa è univocamente identificata dalla coppia di valori (i , k), che indicano rispettivamente il nodo di inizio e di fine della sequenza di nodi della soluzione da invertire.

Lista Tabu

Per evitare che l'algoritmo valuti sempre le stesse soluzioni, alcuni attributi di queste ultime vengono archiviati in una lista Tabu, rendendo meno frequente la valutazione di soluzioni già visitate. Per il nostro problema, l'attributo utilizzato è la coppia di valori (i , k) relativi alla mossa effettuata di recente.
Ad ogni iterazione viene aggiunto un nuovo attributo alla lista tabu, che viene implementata come una **coda di tipo FIFO.**

La lista tabu ha una certa lunghezza (l) ed ogni attributo che entra in lista all'iterazione **k**, ne esce all'iterazione **k+l.** Tale lunghezza dipende dal tipo e dalla dimensione del problema da risolvere e in generale può essere scelto in maniera statica o dinamica. Nel nostro elaborato, tale parametro è stato scelto sperimentalmente ad:

$$\left\lfloor \frac{N}{2} \right\rfloor$$

Criterio di aspirazione

In qualche caso può essere utile, ai fini della ricerca, eseguire particolari mosse anche se risultano tabu.
Un **criterio di aspirazione** è un test che se soddisfatto permette comunque l'esecuzione della mossa tabu.
Il criterio utilizzato per questo algoritmo è quello di consentire una mossa, anche se tabu, nel momento in cui quest'ultima se applicata fornisce una soluzione con valore di funzione obiettivo migliore del valore ottimo attuale.

Diversificazione

Molto spesso, l’algoritmo può rimanere intrappolato in uno spazio di ottimo locale. Per consentire al processo di esplorare nuovi intorni del dominio di ammissibilità (per cercare l'ottimo globale), è necessario diversificare il processo di ricerca, guidandolo in nuove regioni.
In questo elaborato ciò viene implementato utilizzando una "memoria basata sulla frequenza " , che tiene conto di quante volte una determinata mossa 2-opt sia stata effettuata durante tutta l’esecuzione dell’algoritmo. Se non si riesce a trovare una nuova soluzione ottima a seguito di un numero predefinito di iterazioni, si esegue la mossa che è stata eseguita il minor numero di volte. In questo modo è possibile ispezionare intorni ancora inesplorati.

Criterio d’arresto

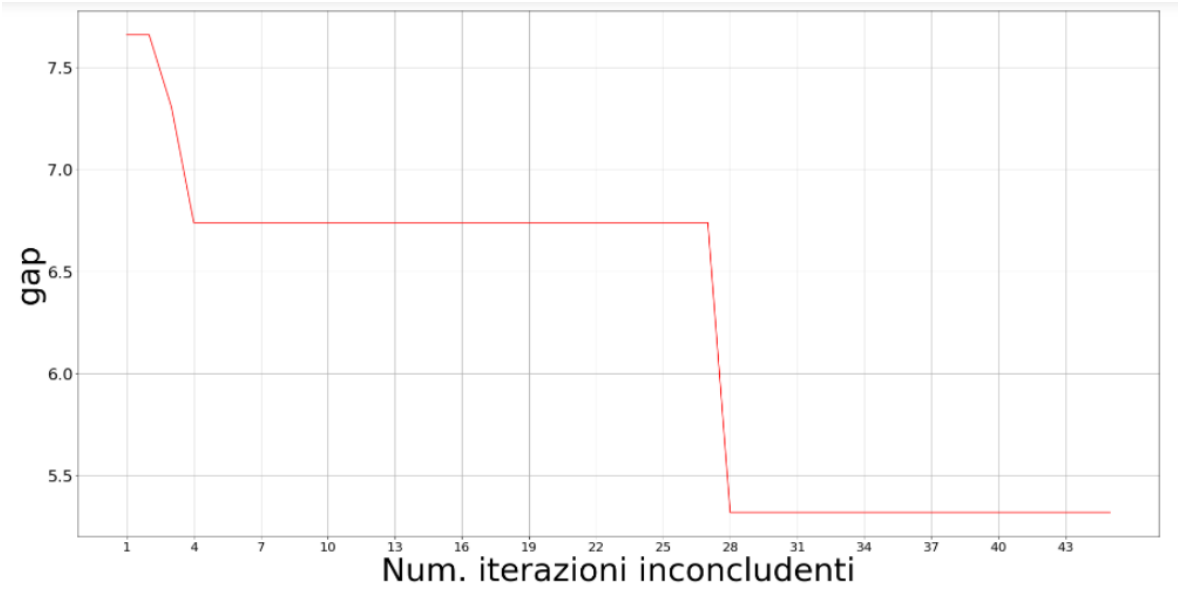
L'algoritmo termina se viene eseguito un numero prefissato di diversificazioni

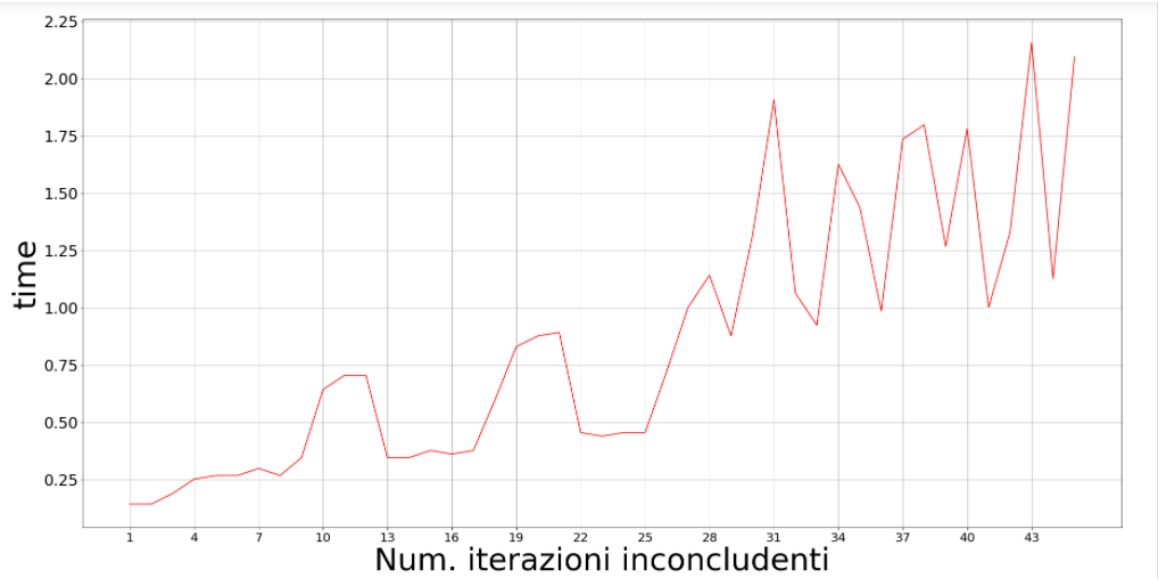
TUNING DEI PARAMETRI

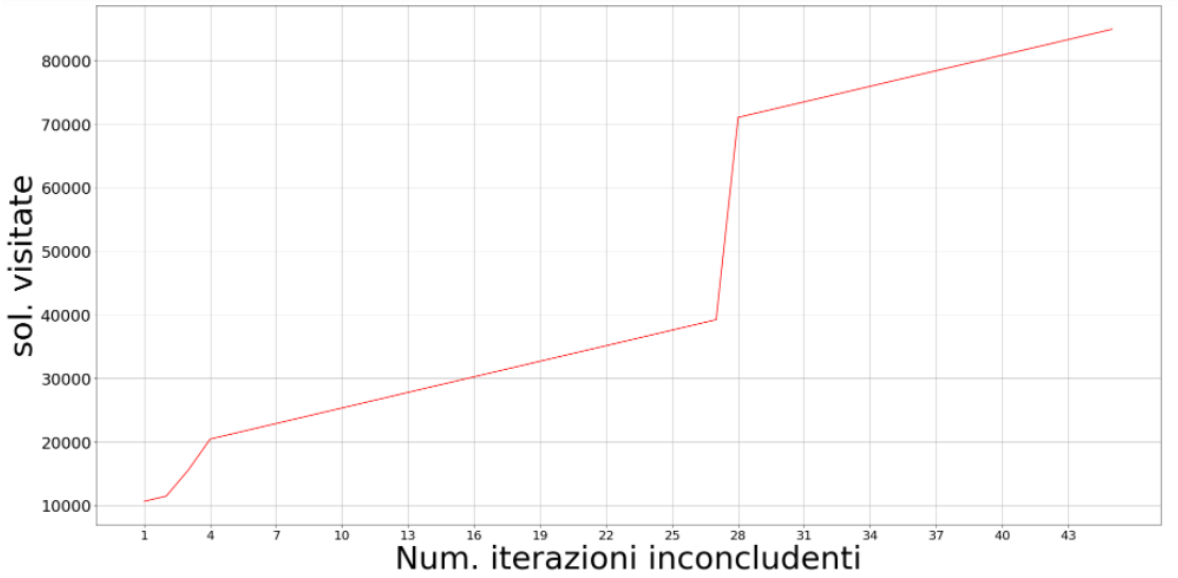
Tabu Search puro

Istanza di prova: «*Dantzig42.tsp*»

Lunghezza lista tabu: 21





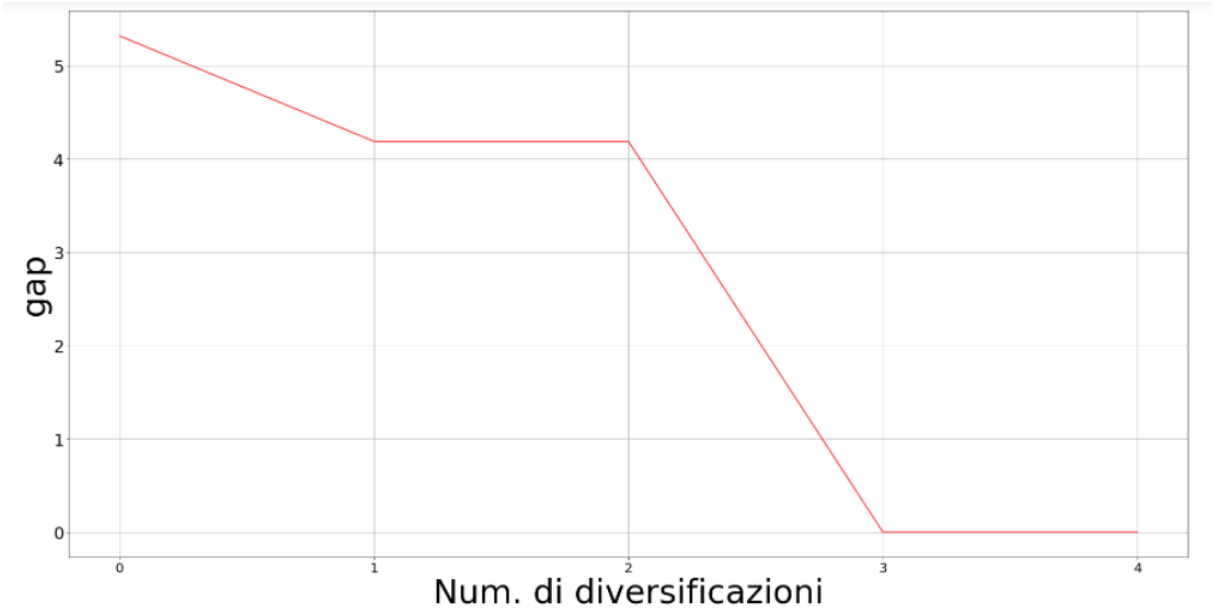


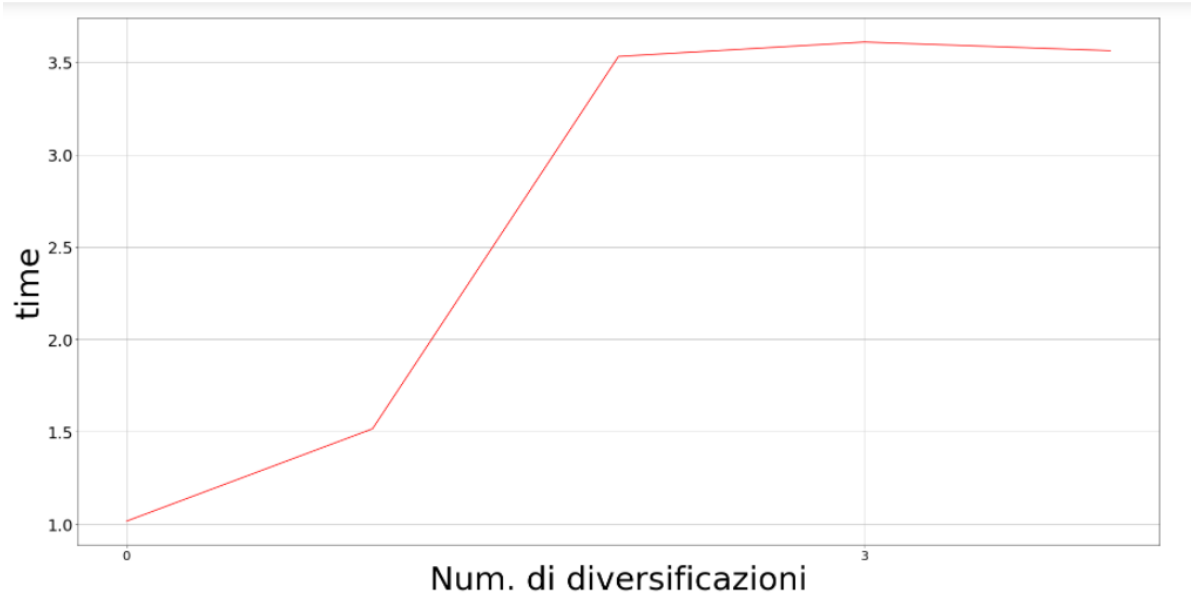
TUNING DEI PARAMETRI

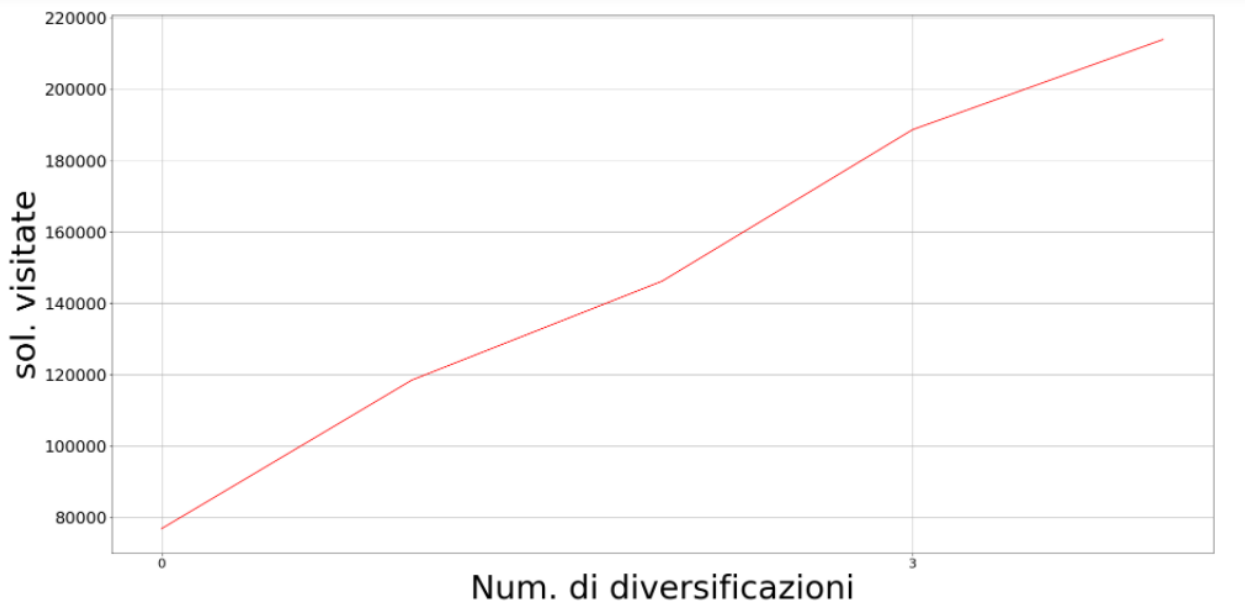
Tabu Search con diversificazione LFU

Istanza di prova: «*Dantzig42.tsp*»

Lunghezza lista tabu: 21
Num. di iterazioni inconcludenti: 31







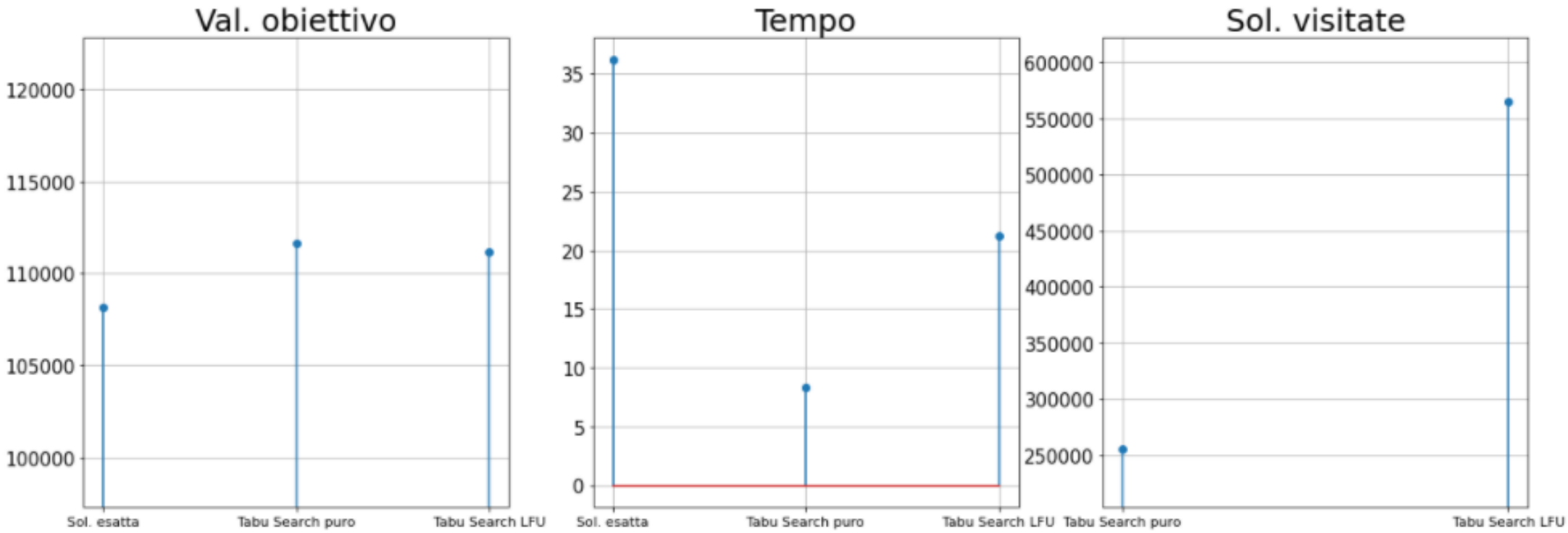
TUNING DEI PARAMETRI

$$Lunghezza\ lista\ tabu = \left\lfloor \frac{N}{2} \right\rfloor$$

$$Sia\ k = \left\lfloor \frac{N}{2} + \frac{N}{4} \right\rfloor, \qquad Num.\ iterazioni\ inconcludenti = \begin{cases} k & se\ k < 35 \\ 35 & altrimenti \end{cases}$$

$$Num.\ diversificazioni = \begin{cases} 0 & se\ N_{iterazioni\ inc.} < 20 \\ 1 & se\ 20 \leq N_{iterazioni\ inc.} < 30 \\ 3 & se\ N_{iterazioni\ inc.} \geq 30 \end{cases}$$

TABU SEARCH PURO VS. TABU SEARCH CON DIVERSIFICAZIONE LFU



Istanza di prova: *«pr76.tsp»*
Lunghezza lista tabu: 38
Num. iterazioni inconcludenti: 35
Num. diversificazioni: 3

Tabu Search Puro

ISTANZA	LUN_TABU	NUM_ITER_INC	GAP	TIME (s)	SOL_VISITED
burma14	7	10	10^-14	0.006	1195
ulysses16	8	12	0.0165	0.014	2325
ulysses22	11	16	0.7976	0.050	7012
bayg29	14	21	0.5090	0.086	9371
dantzig42	21	31	5.3183	0.922	73491
att48	24	35	2.6304	0.688	46277
eil51	25	35	1.5472	0.831	54979
berlin52	26	35	2.9572	2.654	166544
st70	35	35	1.4941	3.117	145249
eil76	38	35	1.9753	3.151	138570
pr76	38	35	3.200	5.635	254999
gr96	48	35	1.4145	6.81	240925
rat99	49	35	4.3819	14.646	512959
eil101	50	35	2.1904	12.650	430307
pr107	53	35	0.777	8.645	289200

Tabu Search con diversificazione

ISTANZA	LUN_TABU	NUM_DIV	NUM_ITER_INC	GAP	TIME (s)	SOL_VIS	TIME_GUROBI
burma14	7	0	10	10^-14	0.006	1195	0.03
ulysses16	8	0	12	0.0165	0.014	2325	0.08
ulysses22	11	0	16	0.7976	0.050	7012	0.09
bayg29	14	1	21	0.3329	0.203	21724	0.19
dantzig42	21	3	31	10^-14	2.14	170653	0.28
att48	24	3	35	0.6909	2.932	194821	0.46
eil51	25	3	35	0.9811	3.636	236937	0.57
berlin52	26	3	35	10^-14	5.596	350881	0.28
st70	35	3	35	10^-7	9.940	487243	1.62
eil76	38	3	35	0.8719	14.859	637430	1.20
pr76	38	3	35	2.786	14.096	565399	25.87
gr96	48	3	35	1.4145	20.225	709408	4.08
rat99	49	3	35	3.510	31.35	1097123	4.75
eil101	50	3	35	1.9177	35.375	1177245	4.75
pr107	53	3	35	0.3037	39.721	1234659	5.56

GAP : Soluzione ottima | 0% - 1.5% | 1.5% - 2% | > 2%