



# lgvRouting

Luca Bartoli, Massimiliano Bosi  
Modena, 24/09/2021

# Dataset

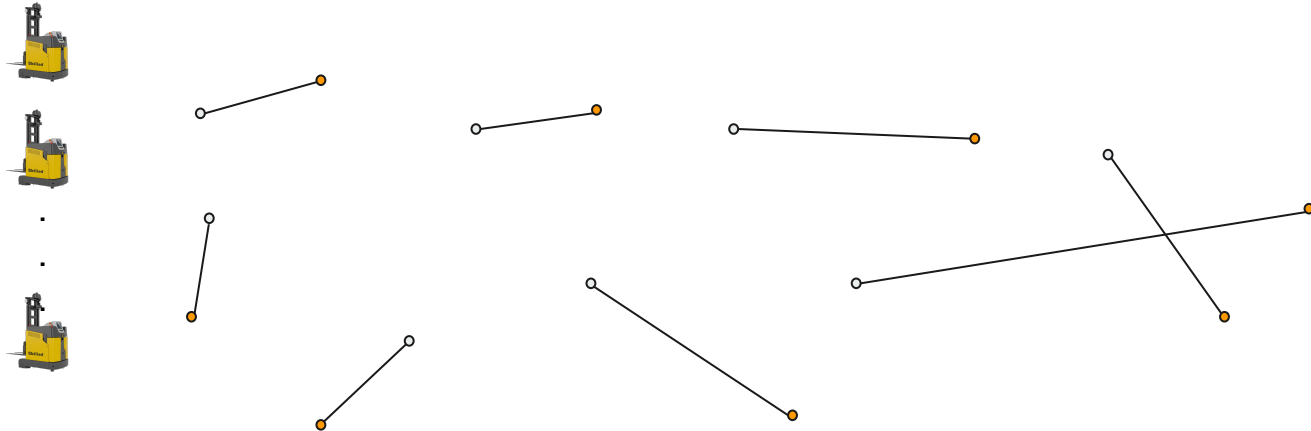
Si è scelto di utilizzare il dataset di VRPPD P2 proposto da Breedam. Il dataset contiene 60 istanze, ognuna di esse formata da 100 stop.

Non sono state valutate la capacità dei singoli lgv e le richieste degli stop poiché si ipotizza di spostare un pallet alla volta.

```
0  50  50  0 9999 9999 9999  0  0  0 (depot)
1   5   5  0 9999 9999 9999 10  0  0 (stop 1)
...
100 95  95 0 9999 9999 9999 10  0  0 (stop 100)
|   |   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |   | delivery=0 / pickup=1
|   |   |   |   |   |   |   |   | service time
|   |   |   |   |   |   |   |   | demand
|   |   |   |   |   |   |   |   | closing time TW2
|   |   |   |   |   |   |   |   | opening time TW2
|   |   |   |   |   |   |   |   | closing time TW1
|   |   |   |   |   |   |   |   | opening time TW1
|   |   |   |   |   |   |   |   | Y coordinate
|   |   |   |   |   |   |   |   | X coordinate
|   |   |   |   |   |   |   |   | stop nr.
```

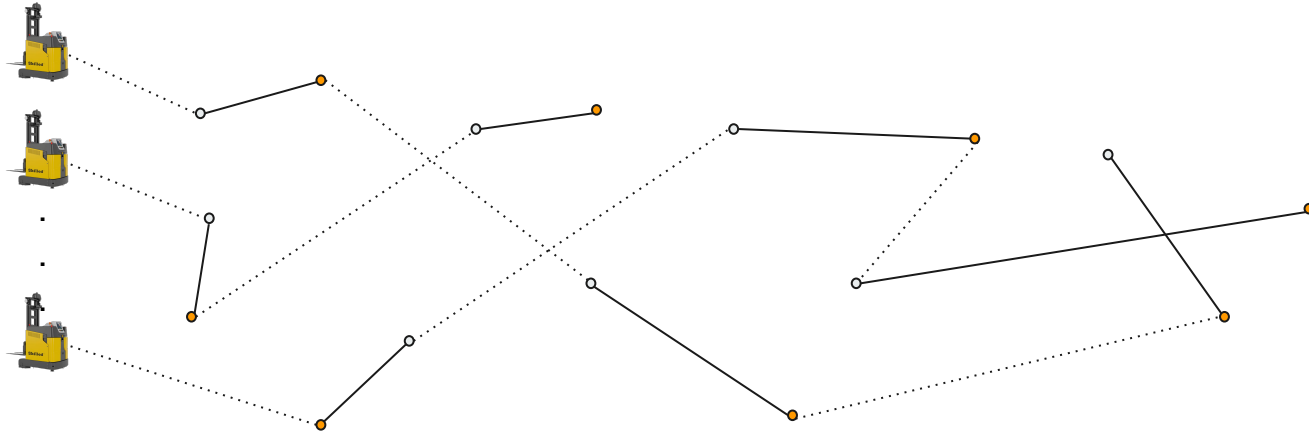
# Problem

Date delle coppie di missioni ( $start_i, end_i$ ) ed  $N$  navette, dobbiamo trovare il routing ottimale per eseguirle tutte nella distanza minore (di conseguenza in tempo minore).



# Problem

Date delle coppie di missioni ( $start_i, end_i$ ) ed  $N$  navette, dobbiamo trovare il routing ottimale per eseguirle tutte nella distanza minore (di conseguenza in tempo minore).





## Premessa

- Distanza calcolata secondo euclide. (possibilità manhattan o maximum)
- Tempi sono calcolati in media sul dataset P2 utilizzando tutti i 60 problemi.
- Come euristica costruttiva le missioni sono ordinate in ordine crescente di peso.



# Swap

- Con uno swap intendiamo l'assegnazione casuale di una missione ad un lgv diverso da quello deciso dalla soluzione costruttiva o randomica.
- Questa metodologia ci permette di muovere all'interno del vicinato di una soluzione, o di allontanarci impostando un numero di swap elevato.



# Algoritmi sviluppati

1. Constructive
2. LocalSearch
3. DepthLocalSearch
4. Multistart (Singlecore - Multicore - GPU)
5. SimulatedAnnealing
6. TabuSearch



# Constructive algorithm

- L'algoritmo è sviluppato secondo un euristica costruttiva. Date  $N$  navette e  $M$  missioni da compiere, l'algoritmo ordina le missioni in base al peso e l'assegna alla navetta con il minor carico.
- Parametri:
  - ◆ Ordine crescente o decrescente.



## Result - Constructive algorithm

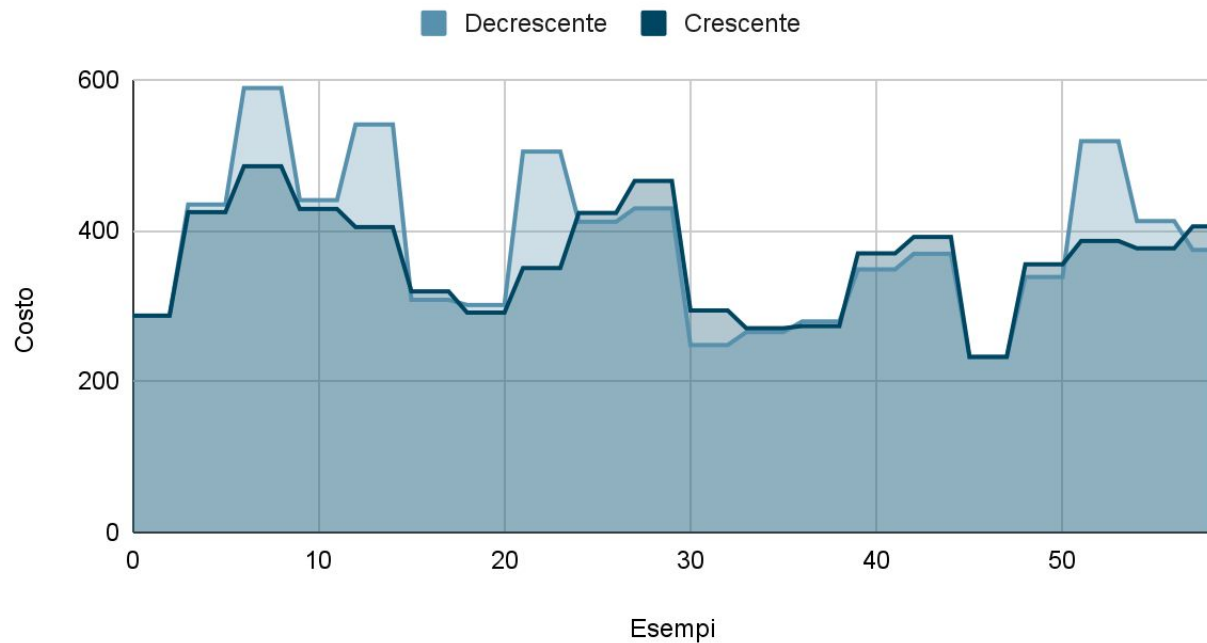
L'ordinamento crescente in media funziona meglio.

Avg:

Decrescente: 375.716

Crescente: 353.404

## Decrescente e Crescente



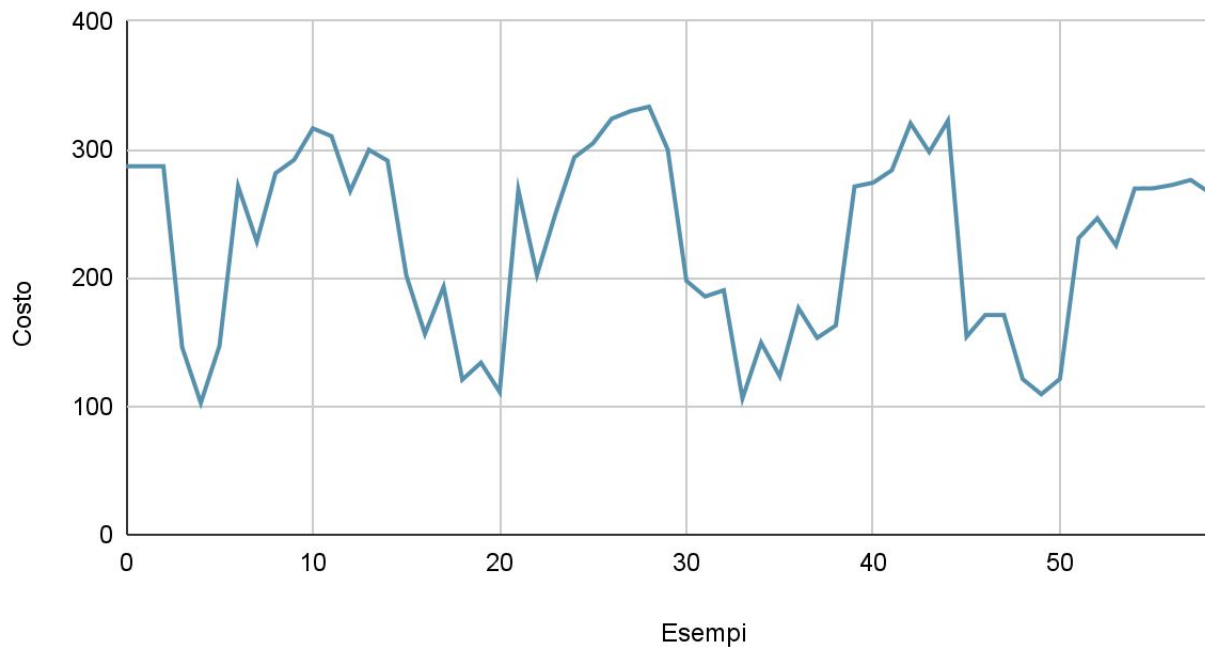


# LocalSearch

- Partendo dall'euristica costruttiva precedente, l'algoritmo si muove all'interno del vicinato attuale prendendo la soluzione migliore tra l'applicazione di uno swap per 1000 iterazioni alla soluzione attuale.
- Parametri:
  - ◆ Timeout.

## Result - LocalSearch

Confronto swap





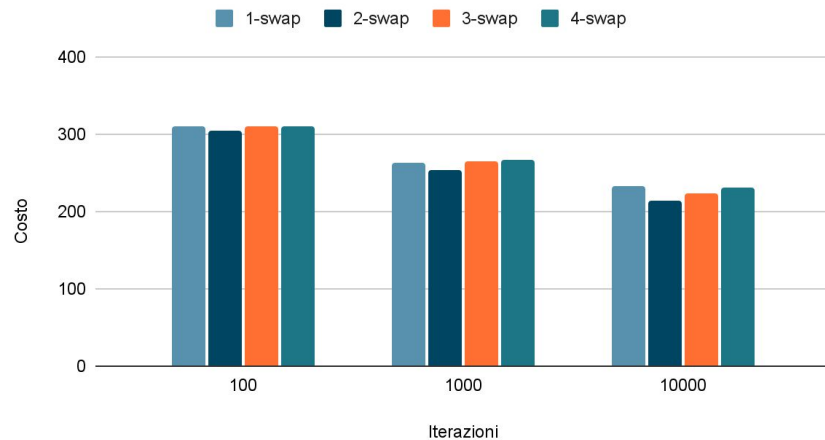
## DepthLocalSearch algorithm

- Modifica alla condizione di terminazione del LocalSearch. Partendo da una soluzione iniziale, si effettuano N swap. Se la soluzione è migliore della precedente si utilizza quella per la prossima iterazione. La condizione di terminazione è data dal numero di iterazioni massime o dal timeout.
- Parametri:
  - ◆ Numero degli swap da applicare alla soluzione iniziale;
  - ◆ Timeout;
  - ◆ Iterazioni massime.

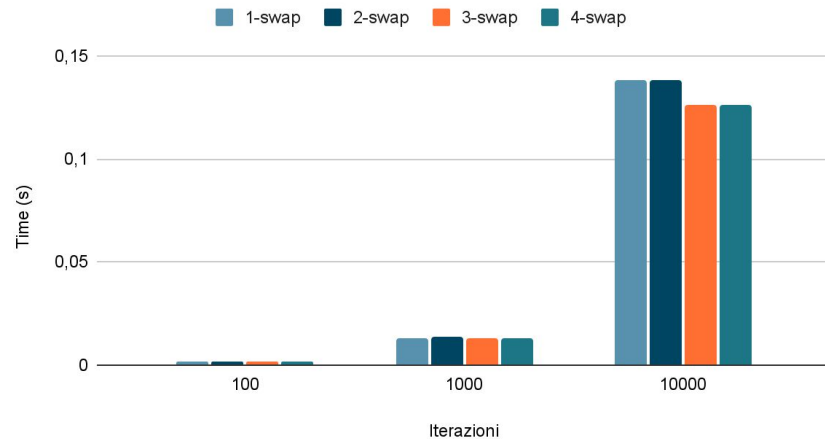
# Result - DepthLocalSearch

Confronto sulla base del numero  
di swap e delle iterazioni

Confronto costo



Confronto tempo





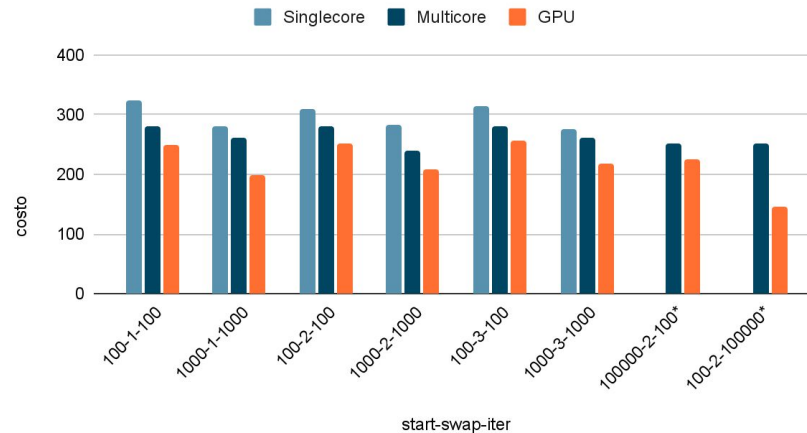
# Multistart algorithm

- L'algoritmo genera  $N_{\text{start}}$  soluzioni casuali, e da queste effettua  $N_{\text{sw}}$  swap per  $N_{\text{it}}$  iterazioni.
- Parametri:
  - ◆ Numero degli swap da applicare alla soluzione iniziale casuale;
  - ◆ Timeout;
  - ◆ Numero soluzioni iniziali;
  - ◆ Iterazioni da eseguire su ogni soluzione iniziale;
  - ◆ SingleCore, Multicore o GPU.

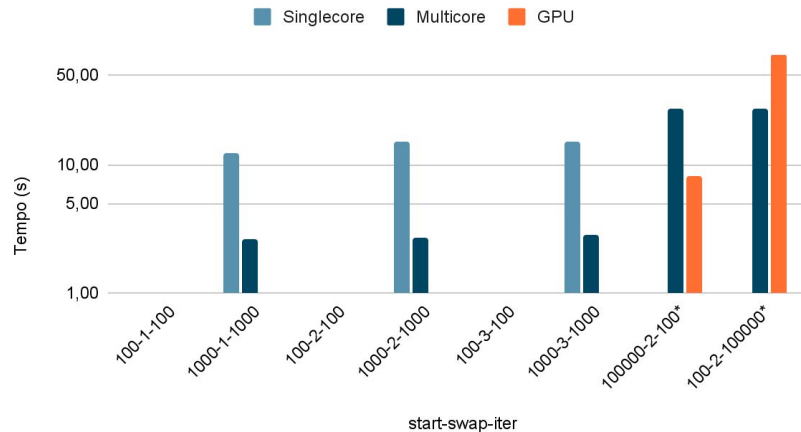
# Result - Multistart

Confronto sulla base del numero di start, swap e iterazioni

Confronto costi



Confronto tempi



\*I valori su singlecore non sono riportati per il tempo necessario al computamento



# Simulated Annealing algorithm

→ Partendo da una soluzione casuale ammissibile, l'algoritmo effettua uno swap con una probabilità  $P(sol)$  se la soluzione trovata è peggiorativa, altrimenti soluzioni migliori sono sempre accettate.

$$P(sol) = e^{-\frac{diff(sol, sol_{prec})}{temperature}}$$

→ Parametri:

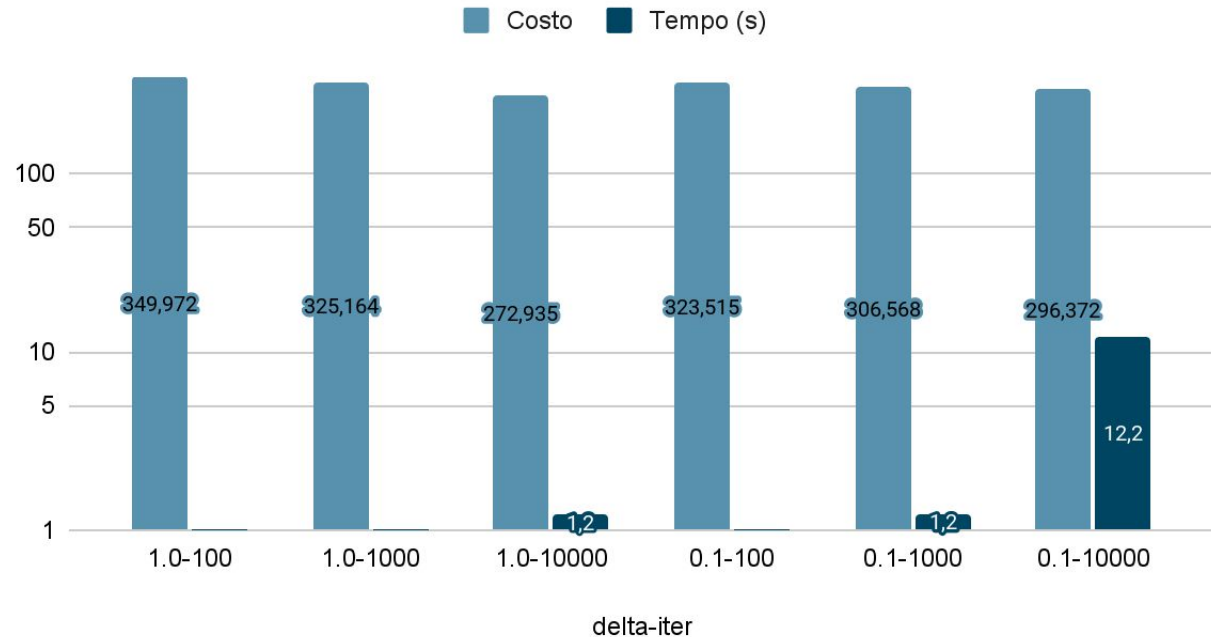
- ◆ Temperatura iniziale;
- ◆ Temperatura minima;
- ◆ Delta di decremento della temperatura;
- ◆ Iterazioni di discesa della temperatura;
- ◆ Timeout.



## Result - SimulatedAnnealing

Test eseguiti fissando una temperatura iniziale di 10 e minima di 0.1

Confronto Simulated Annealing





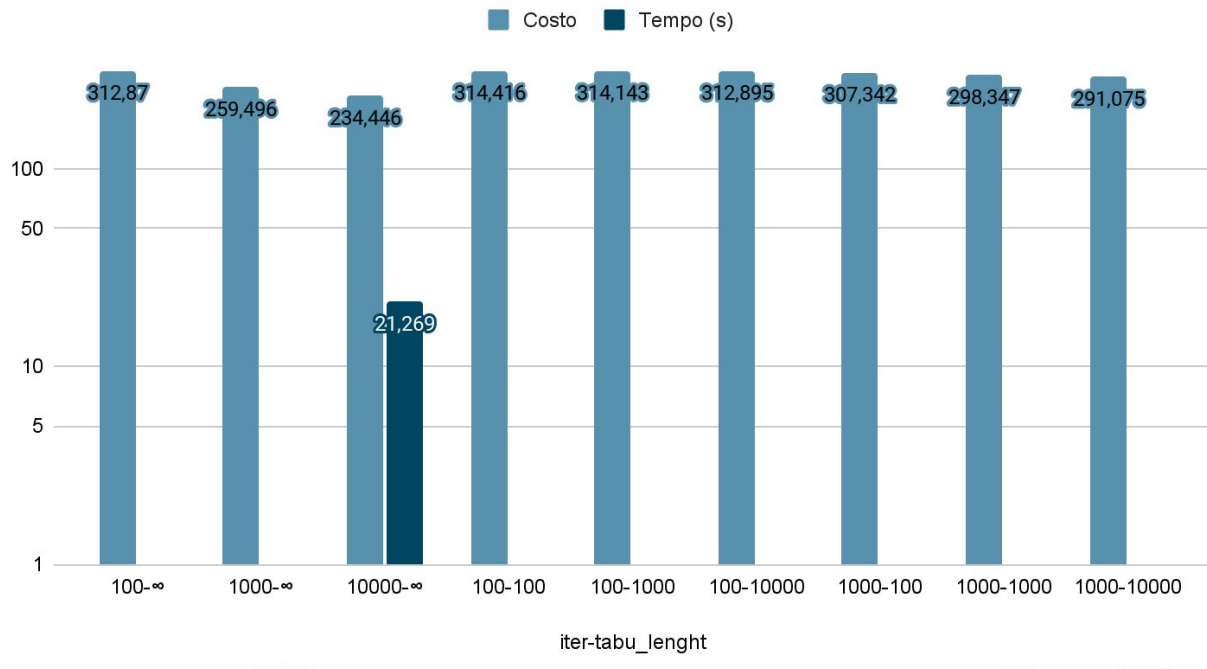
# TabuSearch algorithm

- Partendo da una soluzione iniziale tramite euristica costruttiva, l'algoritmo effettua  $N_{\text{swap}}$  per  $N_{\text{it}}$  iterazioni prendendo anche soluzioni che si discostano di un certo threshold per scappare da minimi locali. Soluzioni già esplorate vengono evitate tramite l'inserimento in una tabu list con lunghezza configurabile.
- Parametri
  - ◆ Iterazioni dell'algoritmo;
  - ◆ Massima differenza di costo per accettare una nuova soluzione;
  - ◆ Lunghezza della lista;
  - ◆ Timeout.

# Result - TabuSearch

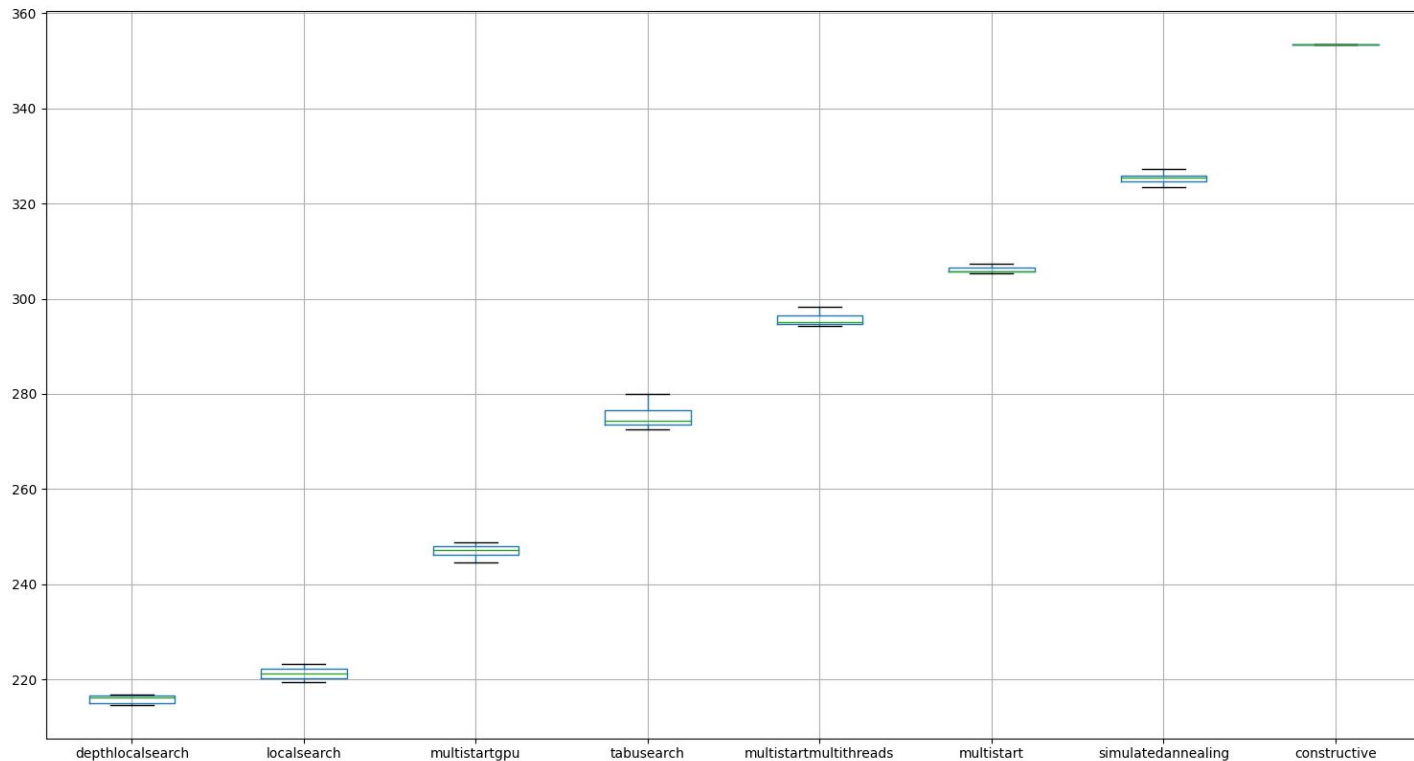
Settata la differenza di  
accettazione a 1

Confronto Tabu List



# Confronto Costo

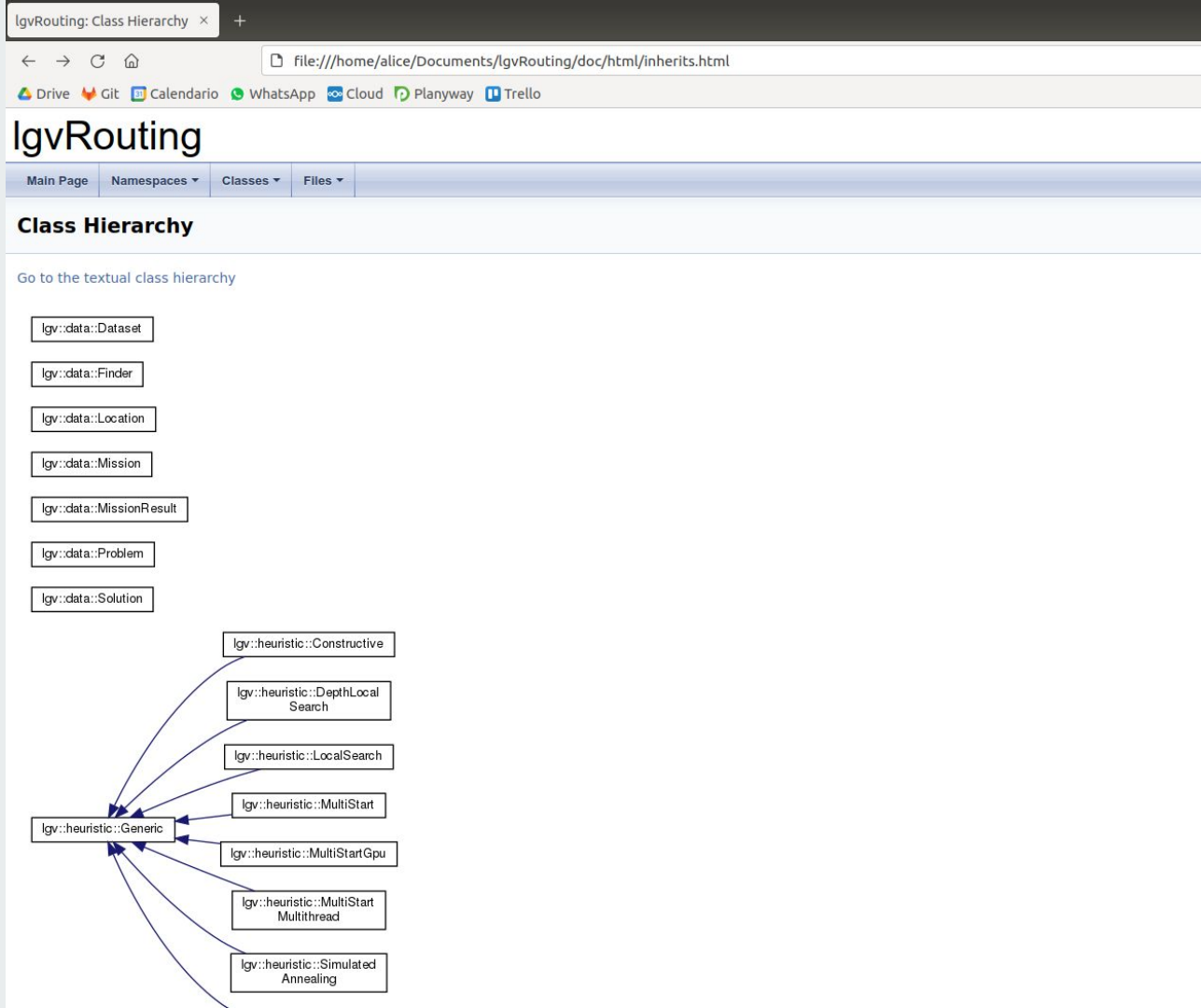
il confronto è  
stato  
effettuato  
mantenendo  
un tempo di  
esecuzione  
per tutti gli  
algoritmi di 1  
secondo



# Codice

- C++, CUDA
- Documentazione con Doxygen
- Test con catch2

<https://github.com/lucabart97/IgvRouting>



**Grazie dell'attenzione**

Luca Bartoli

[228618@studenti.unimore.it](mailto:228618@studenti.unimore.it)

Massimiliano Bosi

[205839@studenti.unimore.it](mailto:205839@studenti.unimore.it)