# Metaeuristica Semi-Greedy applicata a una istanza di problema TSP

Filippo Vajana

November 29, 2019

## 1 Semi-Greedy TSP

Introduzione problema TSP ed approccio Semi-Greedy

## 2 Pseudocodice

**Algorithm 1**

```
1: procedure MAIN
2:     data ← ReadData()
3:     dm ← ParseData(data)
4:     runs ← 10
5:     results ← ∅

6:     for i = 0; i ≤ runs do
7:         (circuit, cost) ← Run(dm)
8:         results ← results ∪ {(circuit, cost)}

9:     Save(results)
```

**Algorithm 2**

```
1: procedure RUN(dm)
2:     rowsCount ← Sqrt(|dm|)
3:     circuitList ← ∅

   ▷ Seleziono nodo iniziale
4:     startNode ← RandomUniform(rowsCount)
5:     circuitList.AddFirst(startNode)

   ▷ Inizializzo frontiera
6:     frontierList ← [0, ..., rowsCount]
7:     frontierList ← frontierList.Remove(startNode)

   ▷ Eseguo metaeuristica
8:     for i = 0; i ≤ rowsCount do
9:         (cNode, fNode) ← SelectNextNode(dm, circuitList, frontierList)
10:        circuitList ← circuitList.AddAfter(cNode, fNode)
11:        frontierList ← frontierList.Remove(fNode)

12:    return (circuitList, Cost(dm, circuitList))
```

**Algorithm 3**

1: **procedure** FILTERFRONTIER($distances, currentNode, frontier$)
2: $filteredFrontier \leftarrow [|frontier|]$
3: $costs \leftarrow [|filteredFrontier|]$
4: $costsSum \leftarrow 0$

▷ **Calcolo costo nodi frontiera**
5: **for** $i = 0; i \leq |filteredFrontier|$ **do**
6: $costs[i] \leftarrow distances[currentNode, i]$
7: $costsSum \leftarrow costsSum + costs[i]$

▷ **Riscalo costi in** $[0, 1]$
8: $minCost \leftarrow \min(costs)$
9: $maxCost \leftarrow \max(costs)$
10: **for** $i = 0; i \leq |costs|$ **do**
11: $costs[i] \leftarrow \frac{costs[i]-minCost}{maxCost-minCost}$

▷ **Filtro nodi nella frontiera**
12: $k \leftarrow$ `RandomUniform`$([0, 1])$
13: **for** $i = 0; i \leq |costs|$ **do**
14: **if** $k \geq costs[i]$ **then**
15: $filteredFrontier \leftarrow filteredFrontier - frontier[i]$

16: **return** $filteredFrontier$

**Algorithm 4**

1: **procedure** SELECTNEXTNODE($dm, circuit, frontier$)
2: $circuitNodeId \leftarrow 0$
3: $frontierNodewId \leftarrow 0$
4: $minimumAddCost \leftarrow \infty$

▷ **Seleziona il nodo da aggiungere al circuito**
5: **for** $circuitNode \in circuit$ **do**
6: $nextNodeId \leftarrow 0$
7: **if** $circuitNode = circuit.Last()$ **then**
8: $nextNodeId \leftarrow circuit.First()$
9: **else**
10: $nextNodeId \leftarrow circuit.Next(circuitNode)$

▷ **Filtro frontiera**
11: $filteredFrontier \leftarrow$ `FilterFrontier`$(dm, circuitNodeId, frontier)$

▷ **Seleziono miglior nodo da aggiungere**
12: **for** $ext \in filteredFrontier$ **do**
13: $extAddCost \leftarrow dm[circuitNode, ext] + dm[ext, nextNodeId]$
14: $extAddCost \leftarrow extAddCost - dm[circuitNode, nextNodeId]$
15: **if** $extAddCost \leq minimumAddCost$ **then**
16: $minimumAddCost \leftarrow extAddCost$
17: $circuitNodeId \leftarrow circuitNode$
18: $frontierNodeId \leftarrow ext$

19: **return** $(circuitNodeId, frontierNodeId)$

# 3 Risultati

**Dati.** Il dataset utilizzato fa parte di **TSPLIB**[1], una libreria che raccoglie numerosi esempi di istanze di TSP e di problemi collegati. Più precisamente l'istanza sulla quale è stato provato l'algoritmo descritto in precedenza è la **bayg29**[2]. In essa sono presenti 29 nodi rappresentanti altrettante città della Baviera mentre i costi sono rappresentativi delle distanze geografiche[3] tra di esse.

**Soluzioni.** Per ogni sessione di benchmarking vengono registrati come soluzione i circuiti calcolati ed i relativi costi. Di seguito viene mostrata una rappresentazione grafica rispettivamente del miglior circuito e del peggiore. Secondo la letteratura corrente il costo ottimo per questa particolare istanza di TSP è pari a **1610**.

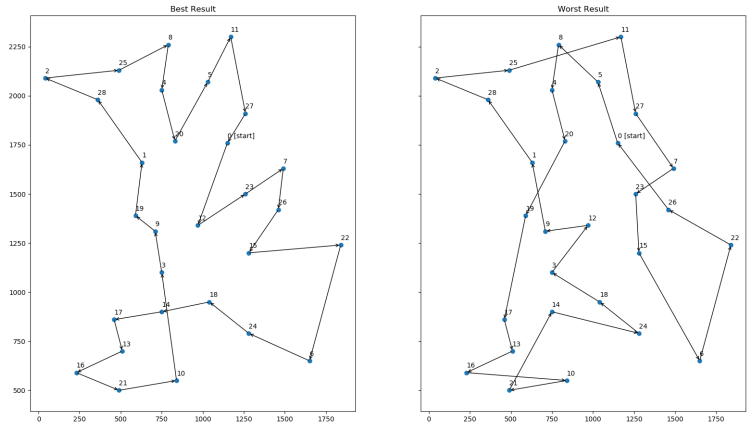


Figure 1: La soluzione di sinistra ha costo totale pari a **1698**, la soluzione di destra invece ha costo **2070**

[1] `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`
[2] `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/tsp/bayg29.tsp.gz`
[3] `https://en.wikipedia.org/wiki/Geographical_distance`