# TSP Partition Heuristics

Sebastian Roberto Opriscan

Algoritmi e Modelli per l'Ottimizzazione Discreta

Università degli Studi di Roma, Tor Vergata
Corso di laurea magistrale in Ingegneria Informatica

22 May 2024

## Outline

## Abstract

This project aims to explore the efficiency of an heuristic for the TSP. Such heuristic is based on the differentiation between the *Connection Constraints*, that say that every center has to be connected to another one, and the *Subtour Elimination Constraints (SEC)* :

By solving a relaxed version of the problem, including only the *Connection Constraints*, the partitions obtained from the nodes will create a new TSP instance whose solution (that can be obtained with this same process, making the heuristic recursive) will give suggestions on how to connect such partitions to obtain a fully connected solution for the problem.

## The baseline heuristic

As we already know, the TSP problem is expressed in terms of two types of constraints :

- The *Connection Constraints*, indicating that every node should be connected to another one :

$$\sum_{\substack{i=1\\ i\neq j}}^{n} x_{ij} = 1, \quad j = 1, ..., n$$

$$\sum_{\substack{j=1\\ j\neq i}}^{n} x_{ij} = 1, \quad i = 1, ..., n$$

- The *Subtour Elimination Constraints (SEC)*, avoiding the formation of subtours

$$\sum_{i\notin S}\sum_{j\in S} x_{ij} \geq 1, \quad \forall S \subset C, S \neq \emptyset$$

Where $C$ is the set of nodes and $|C| = n$

## The baseline heuristic

Furthermore, we know that, of these constraints, the *SEC* are the ones that make the problem really hard to tackle with.

The proposed heuristic (in all its declinations) aims to solve the problem only with the *Connection Constraints*, and then create a TSP instance on the obtained partitions, that, when solved, will prescribe how to interconnect such partitions (which partition should be connected with what).

Note that the TSP instance of the partitions can be solved with the same approach, too, thus leading to a recursive algorithm.

As the base step for this algorithm, it has been decided to use a threshold such that instances with size lesser than that threshold will be solved with all the constraints.

## The baseline heuristic - Code

```
procedure TSPPH(instance, dist_deriv, reconstruct, solver, threshold)
    if instance.size $\leq$ threshold then
        solver(instance, NOT_RELAXED)
        return
    end if
    solver(instance, RELAXED)
    if instance has not got subtours then
        return instance
    end if
    subInstance $\leftarrow$ dist_deriv(instance)
    TSPPH(subInstance, dist_deriv, reconstruct, solver, threshold)
    reconstruct(instance, subInstance)
    return instance
```

## The baseline heuristic

By looking at this description, we can come to understand why the heuristic can have many declinations, because two questions are left open :

- How to deduce the distances for the TSP instances of the partitions?
- How to actually interconnect two partitions?

For the first point, the distances chosen for the generated instances are evaluated from those between the nodes of the partitions (that is, for every couple of partitions $(S, S')$, the distance is a statistic of all the distances between the nodes $(i, j)$, with $i \in S, j \in S'$) , in our case the minimum, maximum and average distances.

## Reconstruction heuristics

For the creation of a fully connected TSP solution given a solved partitions' instance two heuristics are proposed:

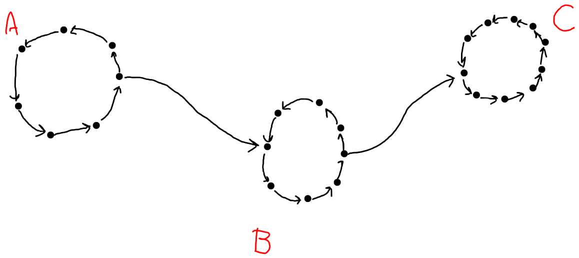
- Minimum reconstruction
- Saving reconstruction

Before exploring such methods it seems important to notice how, being this an heuristic, the focus is shifted more in making something that is fast and reasonable than something that can be proven to be optimal but slow.

## Reconstruction heuristics - Minimum Reconstruction (1)

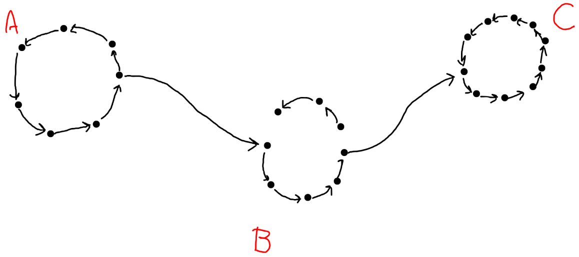
The minimum reconstruction method works in the following way:

Let us consider three partitions, $A, B$ and $C$, and suppose that the partitions' instance tells us that $A \to B \to C$. As a first step, the lower cost arcs between $A$ and $B$ and between $B$ and $C$ are activated.



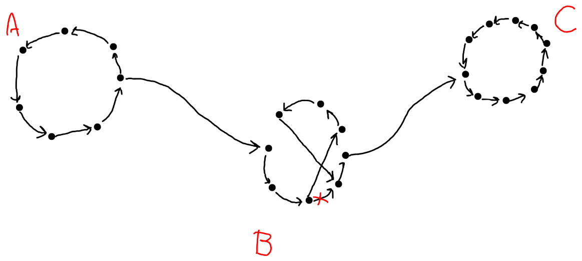
# Reconstruction heuristics - Minimum Reconstruction (2)

Let us call the node in $B$ that *"receives"* the connection from $A$ the *entrance node*, whilst the node in $B$ that connects to the one in $C$ the *exit node*; As a second step, the entering arc of the *entrance node* in $B$ and the exiting arc of the *exit node* in $B$ are disabled:



## Reconstruction heuristics - Minimum Reconstruction (3)

After the last step, the nodes in $B$ are divided in two groups, one connected with the other partitions, another isolated. One of the arcs in the connected group is deactivated and the free nodes are then connected with the *extrema* of the isolated partition. This choice is made in a way to minimize the augmented cost:

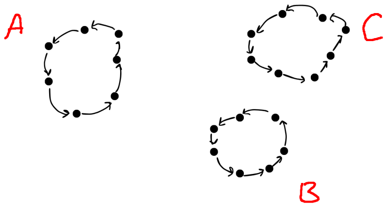


The same process is repeated for every partition.

## Reconstruction heuristics - Saving Reconstruction (1)

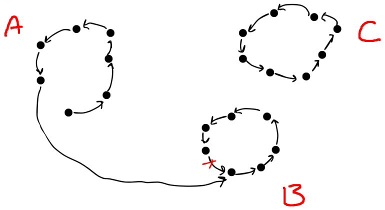
The saving reconstruction method works the following way instead :

Let us consider, once again, our partitions $A, B$ and $C$, but now the partitions' instance, which has size 3, tells us they are connected this way : $A \rightarrow B \rightarrow C \rightarrow A$



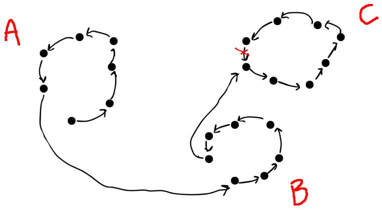
## Reconstruction heuristics - Saving Reconstruction (2)

Starting from an edge of a partition, such edge is removed, and then the exit node is connected to the next partition prescribed by the partitions' instance with the lowest cost edge: this will lead to deactivate the edge entering the *"landing"* node in the reached partition :

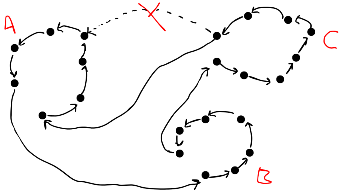


# Reconstruction heuristics - Saving Reconstruction (3)

Then a node will remain without an outgoing connection: this node will be connected to the next partition using the same criterion, until the starting partition is reached again.



## Reconstruction heuristics - Saving Reconstruction (4)



Note : when closing the circle, we are forced to use the edge that connects to the hole left open by the algorithm's starting point.

## Reconstruction heuristics - Saving Reconstruction (5)

How to choose the starting edge? Being that all the choices, selected an edge, are made greedly besides the last one, the forced choice might lead to high costs. For this reason the edges are ranked based on the following score:

$$score(edge) = cost_{best\_exit\_edge} + cost_{worst\_entrance\_edge} - cost_{edge}$$

Where $cost_{best\_exit\_edge}$ is the cost of the edge chosen to connect to the next partition and $cost_{worst\_entrance\_edge}$ is the cost of ending up using the most expensive edge to close the reconstruction. The chosen edge will be the one with the lowest score.

## Analysis of the performance - Environment

Regarding the execution environment for the performance analysis, we have the following:

- CPU clock : 2.6 GHz
- RAM : 8 GB
- Swap space : 6 GB on SSD
- OS : Ubuntu 23.10
- No particular background service
- Timer precision : nanoseconds
- Used solver : HiGHS

## Analysis of the performance - Configuration

To avoid measurement bias, two versions of the algorithm have been created: A normal one and a sampled one, that captures all the structures used in the recursion steps and times spent in the three main phases (solving, derivating, reconstructing), for each of these steps.

The unsampled version has been used to collect the general execution times for the algorithm on the various instances, while the sampled version to collect times specific to the various phases along the recursion steps and the solutions.

Regarding the setting of the threshold, the algorithm has been monitored with thresholds 3, 10 and 18, and every analysed instance has been measured twice for every threshold, one time with the sampled version, the other with unsampled one.

## Analysis of the performance - Data

The instances used for the measurements are the TSP and ATSP problems taken from the *TSPLIB* of the Heidelberg University:

- `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`

## Analysis of the performance - Expectations (1)

What execution cost law, function of the size, should be expected from the algorithm?

The process is recursive and, looking at the code, the derivation and reconstruction functions don't have costs that are easy to predict, but, looking at the data, the algorithm ends up, in its steps, creating partitions as small as possible (mostly of two nodes), being the costs positive.

Assuming this might help us find a rule of thumb for the process Derivation: for every couple of partitions, all edges between those two are checked. Assuming $\frac{size}{2}$ partitions of size 2, we have

$$(\frac{size}{2})^2 * 2 * 2 = size^2$$

## Analysis of the performance - Expectations (2)

For the min reconstruction, considering that the derivation fuction creates metadata for the reconstruction phase, for every partition we must: find the best exit node and the best entrance, then scan the partition nodes to find the rejoining edge so we have :

$$(\frac{size}{2}) * 2 = size$$

for the saving the cost is similar because once found the scoreboard, with cost size, it's just needed to follow the instructions. So the cost is linear ($O(size)$).
The only problem left is the cost of the solver, that being an MIP one is not very much clear.
With hints from the master theorem, having an $\Omega(size^2)$ or more, we'll try a cubic guess.