Heuristic method for the TSPTW

Corentin D. David H.

January 2024

1 Introduction

Heuristics method are a way to approximate a solution for NP-hard problem, and it had been long proven that the Traveling Salesmen Problem (TSP) , which consist to find the Hamiltonian cycle of minimal weight in a complete graph without negative weight cycle is NP-Complete. With the added constraint of a time window, one can show that the TSP can be reduce to the TSP with Time window (by simply adding an unlimited time window) is also NP-Complete.

2 The TSP with Time Window problem

2.1 Parameters of the Problem

Given a oriented graph $G = (V, E)$ where V is a finite set and $E \subset V \times V$. We define a cycle of length $k \geq 2$ $C \subset E$ as

$$\{(u_i, v_i), i \in [0, k-1], \forall i \in [0, k-2], v_i = u_{i+1} \quad v_{k-1} = u_0\}$$

If $\#V = n \geq 0$ and $E = V \times V$, we call this graph complete and hence we will note $K_n = (V, V \times V)$.
We call a *good weight function* f

$$f : E \to \mathbb{R}_*^+, \quad (u, v) \to f(u, v)$$

a function that satisfies the following

1. f is symmetric: for all $(u, v) \in E, f(u, v) = f(v, u)$
2. f verifies the triangular inequality: for all $u, v, z \in V$ such that $(u, v), (v, z), (z, u) \in E, f(u, z) \leq f(u, v) + f(v, z)$
3. For all cycle $C \subset E$, we have $\sum_{(u,v) \in C} f(u, v) > 0$

Moreover, we call a service-time function s

$$s : V \to \mathbb{R}_*^{+\,2}, \quad v \to (a_v, b_v)$$

where $b > a$

2.2 The heuristic aspect

From now, the TSP with Time window will be the information of (K_n, f, s) where f is a *good weight function* and s an service-time function. Since the problem is NP-Complete, the usual method is, as stated before, to approximate the solution, which for our case, will be a list *sol* of distinct node obtained with a permutation of all the elements in V via a heuristic technique. For that, we have to define two key component, shared among the vast majority of heuristic techniques, which will be the subject of the next section :

2.2.1 Evaluation of a solution

The first key component is the ability to discriminate between solutions. For the TSP with Time Window,we need to take account of both the travelled distance and service rate. For the sake of simplicity, we will assume that the salesmen is travelling at speed of 1, meaning that if the distance between two nodes if x, the salesmen will take x to travel it. The *score* of a solution is calculate using the algorithm below and the heuristic is trying to find the solution the minimal score.

Algorithm 1: Evaluation of function

```
Data: (K_n, f, s), sol
Result: score
Initialization;
   distance=0,time=0, score=0, coefficient=100, n=#sol, count=0
for i ← 0 To n − 1 do
    distance ← f(sol[i], sol[i+1])+ distance
    time ← f(sol[i], sol[i+1])+ time
    a,b=v(sol[i+1])
    if time < a then
        time ← a
    if time > b then
        count ← count +1
distance ← f(sol[n], sol[0])+ distance
score ← distance+ count×coefficient
```

2.2.2 Choosing the next possible solution

The second key component is how to choose the next candidate with a certain randomness. Since the considered graph is complete, any permutation of all the nodes consisted of a Hamiltonian cycle. Hence we have a very simple algorithm to choose the candidate.

Algorithm 2: Neighborhood function

Data: *sol*
Result: *sol*
Initialization: $n = \#sol$
$i, j \leftarrow$ Random integer between 0 and $n-1$
Swap sol[i],sol[j]

3 Simulated Annealing

3.1 How it works

Simulated Annealing is one possible heuristic method,which takes inspiration from statistical physics. However, like many heuristic method, it runs the risk of getting stuck in a local minimum. To prevent this, we impose that, with a probability decreasing in time, that even if a solution is worse and the current best, we still take it in the hopes of escaping the local minima. The principle is illustrated in the following pseudo-code:

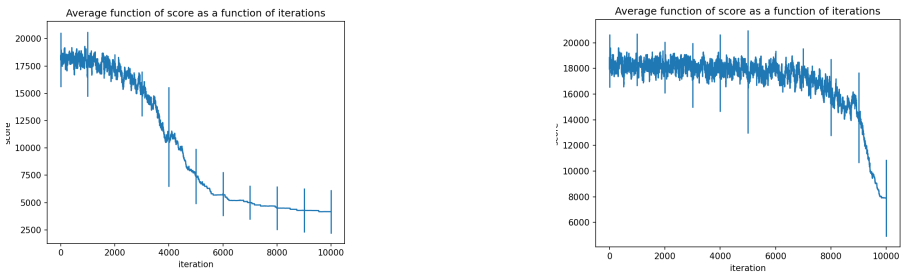
Algorithm 3: Simulated Annealing

Data: $(K_n, f, v), T_{init}, T_{end}$,neighbor,update
Result: sol,$f(sol)$
Initialization: sol=Random shuffle of nodes in K_n, $T = T_{init}$
while $T > T_{end}$ **do**
  next_sol=neighbor(sol)
  $x, y \leftarrow f(\text{next\_}sol), f(sol)$
  $sol \leftarrow$ next_sol **if** $x < y$ **then**
    $p \leftarrow 1$
  else
    $p \leftarrow \exp(-\frac{x-y}{T})$
  $r \leftarrow$ random number between 0 and 1
  if $r < p$ **then**
    $sol \leftarrow$ next_sol
  $T \leftarrow$ update(T)

3.2 Our results

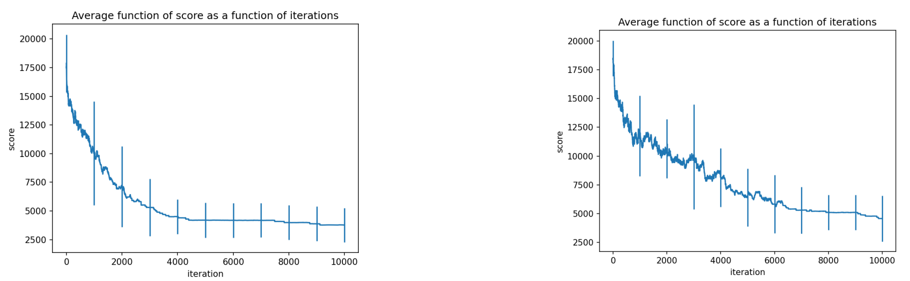
In this section, we will be showcasing the results of two different runs, one where the temperatures decays exponentially, one where it decays linearly. For the score function, we have chosen to put the violation coefficient at 1000, to prioritize minimizing the number of violation.

In our plot, we have plot the average of the score at each iterations over 10 runs, and the error bar is the amplitude times a coefficient, here chosen to be 0.5. We are also comparing the effect of the initial temperature on the heuristic.



(a) Exponential Decay (b) Linear Decay

Figure 1: With $T_{\text{init}} = 5000$ and $T_{\text{end}} = 10$



(a) Exponential Decay (b) Linear Decay

Figure 2: With $T_{\text{init}} = 500$ and $T_{\text{end}} = 10$

As we can see, there is a lot of randomness at the start, since the temperature is high. As expected, that randomness decreases faster with the exponential decay. However, with a lower starting temperature, the randomness is less present.

4 Analysis and Conclusion

Yet, we say that the score doesn't fall below 1000, meaning that there are still violations happening. It appears that Simulated annealing isn't really suited for this problem. Comparison with other heuristic method is to be conducted