UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

"Digital Automation Engineering" Master Degree
Course "Optimization Methods for Data-driven Engineering Processes"

Heuristics proposal for the Travel Salesman Problem with Time Windows

Simone Giovanardi
Leonardo Brighenti

Abstract

This report's aim is to present the Travel Salesman Problem with Time Window (TSPTW), explaining it and its constraints. An optimal solution of the problem is analyzed using PuLP, and then due possible heuristics are proposed. Both of them are based on sorting the customers in base of their closing time window, and then finding the shortest path to connect the customer at the top of this list and the successor. The second proposed heuristic implements the A* algorithm, in order to improve the first one. Finally, some datasets of different sizes are tested with these methods and results are reported and discussed.

1. Problem definition

The aim of the Traveling Salesman Problem with Time Windows is to find a minimum-cost path that visits each of a set of customers exactly once, starting and ending on a given depot, where each customer has a time window that must be respected. The cost is represented by the total time needed to visit all the customers and go back to the depot. Every customer must be visited before its due date and after the ready time, otherwise any other visiting times are considered infeasible; if the vehicle arrives before the ready time, it must wait.
The TSPTW can be on a graph G = (V , A), where V={1, 2 , . . . , n} is the set of customers, 0 is the depot, and A ={ $(i , j) : i, j \in V \cup \{0\}, i \neq j$ } is the set of arcs between customers. The parameter c_{ij} represents the cost of traveling from the customer *i* to the customer *j*, and it includes both the service time of customer *i* and the time needed to travel from *i* to *j*. The time window of a customer *i*, is denoted as $[a_i , b_i]$, where a_i is the ready time and b_i is the due time.
In case of customer *j* (or the depot in case *j*=0) is visited immediately after visiting customer *i*, the value of x_{ij} is equal to 1 and 0 otherwise. The customer *i* is visited at time β_i and M is taken as a big constant. The problem can be formulated through the objective function and constraint as an integer programming problem:

$$\text{Minimize} \sum_{i \in V \cup \{0\}} \sum_{j \in V \cup \{0\}} c_{ij} x_{ij} \qquad (0)$$

$$\sum_{i \in V \cup \{0\},\, i \neq j} x_{ij} = 1 \quad \forall j \in V \cup \{0\} \tag{1}$$

$$\sum_{j \in V \cup \{0\},\, j \neq i} x_{ij} = 1 \quad \forall i \in V \cup \{0\} \tag{2}$$

$$\beta_j \geq \beta_i + c_{ij} - M(1 - x_{ij}) \quad \forall i,j \in V \cup \{0\}, j \neq 0 \tag{3}$$

$$a_i \leq \beta_i \leq b_i \quad \forall i \in V \tag{4}$$

The objective function minimizes the total cost of the tour, Constraint (1) and (2) state that every customer and the depot must be visited exactly once. Constraint (3) ensures that the arrival time to a given customer cannot be smaller than the arrival time to the customer visited immediately before plus the cost of traveling between the two customers. Since all costs are positive constraints (3) avoid the formation of sub- cycles. Finally, constraint (4) imposes the time window.

2. Implemented model with PuLP

In order to have an optimal solution, the problem is solved firstly with Pulp, where the model is created with sense = 1, because of the minimization.

$$x_{ij} \in \{0,1\} \quad \forall i,j \in V \cup \{0\} \tag{5}$$

$$\beta_i \in \mathbb{R}_{\geq 0} \quad \forall i \in V \cup \{0\} \tag{6}$$

Variables are x_{ij} and β_i as in (5) and (6) respectively, where x_{ij} only has binary values and β_i is greater or equal than zero because of the definition of it; both include customers and depot. Objective function is expressed in (0), where summation is done using lpSum, and constraints are those already explained in the above section with formulas from (1) to (4). The function perf_counter( ) is used to measure the computing time of the code. PuLP is tested on different datasets, increasing in complexity, where it's a file written in the same way as the above one.

```
#N => number of customers
5

#[ai, bi] => Time windows
20 38
28 91
11 43
25 92
2 37

#T => time matrix
0 7 9 1 6 2
7 0 9 4 1 8
9 9 0 10 4 6
1 4 10 0 6 3
6 1 4 6 0 4
2 8 6 3 4 0
```

Then there are the number of customers, the time windows for each one, and the time matrix, which represent each c_{ij} cost between customers or between a customer and the depot; cost matrices are taken as symmetrical. Each file is created in a random way and PuLP is able to read it as a .txt file to receive all necessary information. Instances were tested with an increasing number of customers, in particular 5, 10, 15, 20, 30, 40, 50.
Finally a file is generated with some information, such as the instance name, the status of optimization, objective function value, variables value and computing time.

3. Implemented Heuristics

In this section a possible heuristic is proposed, and also an enhanced version of it, using the A* Algorithm.

3.1 Pure heuristic

3.1.1 General Idea of the Heuristic Algorithm

The heuristic algorithm presented aims to combine operational simplicity with a high level of efficiency within the defined time windows. Priority was given to the prevention of delays in order to eliminate the possibility of infeasibility. Therefore, a sequential route was defined to ensure that all customers visited. In order to deal with critical situations, waiting periods and a tolerance system were introduced to reconfigure the order of deliveries in the face of different time openings of customers.

It was decided not to deal with cost minimisation directly within this algorithm in order to preserve its simplicity and reduce computational complexity. A complementary algorithm was developed for cost optimization, which is detailed in the following chapters, therefore, this approach is referred to as 'pure heuristics'.

3.1.2 Explanation of the Heuristic Algorithm

The algorithm begins by organizing customers according to the next closing of their time windows, sorting them into a list, in order to establish a prioritized order of visits. It then proceeds with the systematic evaluation of the customers, calculating the direct cost for each starting node (the last customer in the tour) and target node (the customer in the priority list). Crucially, the expected arrival time is checked compared to the time window of the selected customer.
Scenarios in which arrival is earlier than the customer's time window are handled through the evaluation of waiting times, with the possibility of adjustment or, if necessary, reallocation of visits to other customers in order to avoid unnecessary delays. The process continues until visits to all listed customers are completed.

3.1.3 Pseudocode

SET a priori the status to 1 (success)
INITIALIZE a counter T, it used to select the customer in the dictionary
CALCULATE delta as an initial value based on the cost matrix
INITIALIZE beta to 0, it is a objective value/timer
CREATE a dictionary of customers to visit with their time windows
SORT customers based on the end of their time window
INITIALIZE the tour list with the starting node (depot)
START counting time

```
WHILE there are still customers to visit in the sorted dictionary
  SET UP temp: a temporary structure for cost and path
  CHOOSE the target customer based on T

  SAVE and PRINT:  information about problem status

  CALCULATE the direct cost from last element in tour list to the target customer
  UPDATE the temporary structure with direct cost and path

  COMPARE arrival time with his customer's time window
  IF arrival is within time window
    RESET T=0
    UPDATE beta with direct cost stored in temp
    REMOVE the visited customer inside dictionary with path stored in temp
    ADD the visited customer in tour list with path stored in temp
  ELSE IF arrival is early but acceptable with delta waiting
    RESET T=0
    CALCULATE waiting time, it is the time to wait the opening of time window
    UPDATE beta with direct cost stored in temp plus waiting time
    REMOVE the visited customer inside dictionary with path stored in temp
    ADD the visited customer in tour list with path stored in temp
  ELSE IF arrival time is too early
    INCREMENT T=T+1, for reiterate at next customer in customers dictionary
    IF T exceeds the number of unvisited customers
      INCREASE delta
      RESET T=0
  ELSE
    SET status to 0 (infeasible)
    END while loop

  SAVE and PRINT: log and tour status

Outside While loop:
ADD the cost of returning to depot to beta_H2
ADD the depot to the tour

END counting time
CALCULATE timing cost

SAVE the optimization results and log, into a txt file

IF status is successful
  PRINT the status
ELSE
  RAISE an infeasibility error, all data are already collected into txt file
```

3.2 Difference between the pure algorithm and A* enhanced heuristics

This section illustrates the differences related to the use of a basic heuristic in comparison to its enhanced version by the integration of the A* heuristic.
Both versions maintain the same basic algorithmic structure, differing only in the method for calculating the search path finding from the starting node to the target node, referred to as the starting and target nodes, respectively. The pair of nodes is predetermined by the main algorithm according to predetermined criteria.
The heuristic without A* proceeds from the starting node to the target directly, without evaluating intermediate nodes, and calculates the cost based only on the direct edge, without further cost analysis. In contrast, the heuristic including A* employs the A* search algorithm to define the least burdensome path, also taking into account indirect routes involving intermediate nodes, and decreasing the nodes to be visited, in this specific case only those with compatible opening hours have been evaluated.

3.3 Snippet of pseudocode of heuristics with A*

In this section, a pseudocode fragment relating to the implementation of the A* heuristic is presented. This snippet points out the changes introduced with respect to the standard heuristic, describing the method used to determine both the direct cost and the estimated cost with A* for the route between intermediate nodes.

```
CALCULATE the direct cost from last element in tour list to the target customer
FIND and CREATE a list of customers opened
SET big number for indirect cost

PERFORM A* search between the last element in tour and customer target, PASS even a list of
customers opened to define the passage accessible.
IF A* search is successful
   SET indirect cost to the cost result of A* search

DETERMINE whether the indirect path is better than the direct path
IF indirect cost is less than direct cost
   UPDATE temp with indirect cost and A* search path
   LOG 'The best path is A*, indirect path' with the A* search path and cost
ELSE IF indirect_cost is equal to direct_cost AND A* search path length is 2
   UPDATE temp with direct_cost and direct path (last node of tsp_H1_tour to target)
   LOG 'A* cost is equal to direct path cost' with direct path and cost
ELSE IF indirect_cost is equal to direct_cost AND A* search path length is greater than 2
   UPDATE temp with indirect_cost and A* search path
   LOG 'A* path has equal cost to direct path but passes through more nodes' with A* search path
and cost
ELSE
   UPDATE temp with direct cost and direct path
   LOG 'The best path is the direct path' with the direct path and cost
```

4. Results

Some final considerations are exposed in this section, in order to compare the three investigated methods (PuLP and the two heuristics), considering several aspects. The priority was focused on evaluating the differences between the pure heuristic and its improved version using A* algorithm, and where possible, a comparison with PuLP solution is performed. Here the two considered parameters are the total cost (objective value) and the computational time required to identify the optimal solution. Below, is presented a table where some cases are considered based on the number of customers (5, 10, 15, 20, 30, 40, 50).

Customers	Models	Computational time (s)	Objective value
5	PuLP	0.052	20
	with A*	0.013	37
	pure heuristic	0.02	37
10	PuLP	0.29	28
	with A*	0.161	46
	pure heuristic	0.014	60
15	PuLP	2.0	33
	with A*	0.031	68
	pure heuristic	0.027	79
20	PuLP	7.744	30
	with A*	0.03	64
	pure heuristic	infeasible	infeasible
30	PuLP	134.59	35
	with A*	0.051	87
	pure heuristic	infeasible	infeasible
40	PuLP	/	/
	with A*	0.048	92
	pure heuristic	infeasible	infeasible
50	PuLP	/	/
	with A*	infeasible	infeasible
	pure heuristic	infeasible	infeasible

Table 1. Computational time and objective value in different cases

Note that in the case of 40 and 50 customers, PuLP takes too long to find a solution (more than two hours with 40 customers).
It is possible to see that where all the 3 models give a solution, the pure heuristic is faster than the heuristic with A*, which in turn is faster than PuLP, except in the case of 5 customers. On the other hand, PuLP gives a better solution, in fact objective value is always lower, when calculated. The case with 50 customers doesn't provide a solution (because of the chosen time windows), and in any case an higher number of customers can't be evaluated, because 50 is the upper bound for the algorithm and the cost of edges is more than the closing time of the last customers.
Table 2, 3 and 4 give an idea of the differences in terms of percentages.

Customers	PuLP Computational Time Vs Pure Heuristic	PuLP Objective Value Vs Pure Heuristic
5	+162.9%	-45.9%
10	+1992.1%	-53.3%
15	+7281.9%	-58.2%
20	Not available	Not available
30	Not available	Not available

Table 2. PuLP Vs Pure Heuristic Comparison

Customers	PuLP Computational Time Vs A*	PuLP Objective Value Vs A*
5	+291.3%	-45.9%
10	+80.6%	-39.1%
15	+6372.9%	-51.5%
20	+26044%	-53.1%
30	+263085.7%	-59.8%

Table 3. PuLP Vs A* Comparison

Customers	A* Computational Time Vs Pure Heuristic	A* Objective Value Vs Pure Heuristic
5	-32.8%	-45.9%
10	+1058.2%	-53.3%
15	+14.0%	-58.2%

Table 4. A* Vs Pure Heuristic

Where a negative percentage of objective value with respect to another model is a better solution. A positive percentage respect to another model in terms of computational time means that it is slower. The integration of A* produced an improvement in the target value cost compared to the version without A*, highlighting the superiority of A* in identifying optimal paths that include intermediate nodes, consequently reducing the burden in terms of resources used. However, this advantage is contrasted with an increase in computation time, presumably due to the increased computational complexity that A* introduces.
Another parameter which allows to do some comparison between pure heuristic and its improved version with A* is the number of iteration cycles. The reduction in the number of iterations using A* heuristic was not significant, demonstrating how limited savings in iterations can still affect the objective cost. Table 5 provides a quantitative representation of this decrease. A detailed analysis measured how often the algorithm with the addition of the A* heuristic prefers indirect paths, typically for lower cost or more nodes visited. This preference was quantified as a percentage, providing an indication of the decision-making impact of A* on the total number of iterations of the heuristic.

Customers	A* used	Times in which Direct path used	Times in which A* is preferred instead pure heuristic	Reduced number of cycle using A* compared to not using it
5	0	8	0.0%	0.0%
10	1	18	5.26%	5.0%
15	6	11	35.29%	28.57%

Table 5. Comparison between pure heuristic and A* version using the number of iteration cycles

In conclusion, despite the increased computation time attributable to A* heuristics, the benefits in terms of reduced target value cost make A* heuristics preferable in scenarios where economic efficiency is preferred over speed of execution. These data underline the importance of choosing appropriate heuristic strategies based on the specific objectives and operational context. Using or not a certain model is established in base of the goal, for instance if it is necessary to have the lowest travel time possible, and the computational cost to obtain the solution is not a problem, PuLP is a good choice. Instead, if the priority is to have a possible solution in very short time, the better choice is the improved algorithm with A*, having a compromise in terms of travel time, because it will be greater than the optimal one. In conclusion the trade-off depends on the particular case at hand and the choice will be mainly affected by the number of customers; it's important to have in mind that sometimes a solution cannot be detected because for instance time windows can be very close together, or they can be narrowed, so the problem will be very complex.