Ai for TSP Competition

The Margaridinhas

July 13, 2021

Fig. 1: From left: Justine Pepin, Caroline Leboeuf, Warley Almeida, Carl Perreault-Lafleur, Federico Bobbio.

1 Introduction

As part of the AI for TSP Competition, we are interested in solving a variant of the traveling salesman problem. Namely, we are making use of Artificial Intelligence (AI) and Operations Research (OR) methods for solving the time-dependent orienteering problem with stochastic weights and time windows (TD-OPSWTW) using surrogate based algorithms. Here, the objective is to maximize the total rewards collected at each node of a tour in a network, where the actual travel time between nodes is stochastic. Furthermore, penalties are given if the time windows associated to each node are not respected, and if total travel time of the tour exceeds a pre-defined threshold.

The team *Margaridinhas* is a group of 5 Masters and PhD students from Université de Montréal. During the competition process, we made several attempts to solve the stated problem using different methods which were most familiar to us given our shared background in Operations Research. As such, we have started with a surrogate algorithm based on a mixed-integer programming model, which increases the problem size as cuts of impossible nodes or sub-tours are added as constraints to the solver. The complete description of this algorithm is available in section 2. We have also developed a genetic algorithm, as described in section 3, which enabled to seek for possible node swaps from the surrogate model solution to increase the total reward of the tour. And finally, an alternative approach using dynamic programming concepts was used to compare the results with our surrogate model. The dynamic approach showed very good results in the validation phase, but results were not conclusive compared to the previous two methods in the test phase. The description of this algorithm is available in appendix 6 as a reference. Finally we also present a preliminary exploration of the Nearest Neighborhood methodology and its effectiveness on the problem; this topic is treated in the last part of the Appendix 6.2.

2 Surrogate-based tracker approach

Here is the complete description of the surrogate-based *tracker approach*.

2.1 Concept definitions

This section presents the technical terms used throughout this section:

- Simulation means one call to the black-box simulator with a route (namely, one call to the `check_solution` function of the `env.py` file);
- Solution means an output of the surrogate model (namely, a value for each decision variable after the optimization of the surrogate model);
- Route means a sequence of nodes starting at and going back to the depot;
- A solution represents a route, and a route may be represented as a solution. Thus, the terms route and solution may be used interchangeably.

2.2 Surrogate model

This section presents the surrogate model used in the tracker approach.

2.2.1 Data

The surrogate model takes into consideration the following deterministic data:

- $\mathcal{N}$ - set of nodes in the network;
- N - number of nodes in the network;
- T - maximum duration of the route;
- r_i - reward retrievable at node i;
- o_i - opening of time window at node i;
- c_i - closing of time window at node i;
- t_{ij} - worst travel time of arc (i, j).

The surrogate model takes into consideration the following stochastic data:

- p_{ij} - penalty associated with arc (i, j).

The stochastic data is estimated with increasing accuracy as previous simulations accumulate.

2.2.2 Decision

The surrogate model takes in consideration the following decision variables:

- x_{ij} - 1 if arc (i, j) is in the route, 0 otherwise;
- t_i - step of the route in which node i is visited.

Let $\mathcal{T}$ be the space of the decision variables, and s be the length of the route.

The tracker approach solves the following surrogate model $\mathcal{M}(s)$:

$$\mathcal{M}(s): \quad \max_{(x,t)\in\mathcal{T}} \quad \sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}}(r_j + p_{ij})\cdot x_{ij} \tag{1a}$$

$$\text{subject to:} \quad \sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}} x_{ij} \leq s \tag{1b}$$

$$\sum_{k\in\mathcal{N}} x_{1k} = 1 \tag{1c}$$

$$\sum_{k\in\mathcal{N}} x_{k1} = 1 \tag{1d}$$

$$\sum_{k\in\mathcal{N}} x_{ki} = \sum_{k\in\mathcal{N}} x_{ik} \qquad \forall i \in \mathcal{N} \tag{1e}$$

$$\sum_{k\in\mathcal{N}} x_{ik} \leq 1 \qquad \forall i \in \mathcal{N} \tag{1f}$$

$$\sum_{k\in\mathcal{N}} x_{ki} \leq 1 \qquad \forall i \in \mathcal{N} \tag{1g}$$

$$t_j - t_i \geq 1 - N\cdot(1 - x_{ij}) \qquad \forall i \in \mathcal{N}, \forall j \in \mathcal{N}/\{1\} \tag{1h}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in \mathcal{N}, \forall j \in \mathcal{N} \tag{1i}$$

$$t_i \geq 0 \qquad \forall i \in \mathcal{N} \tag{1j}$$

The objective function (1a) maximizes the deterministic reward plus the estimated penalty of the route based on previous simulations. Constraint (1b) restricts the number of arcs a route can have to s, which is a parameter of the tracker approach. Constraints (1c)-(1h) define together how a valid route must look like. Constraint (1c) guarantees that the route starts at the depot, whereas constraint (1d) guarantees that the route ends at the depot. Constraints (1e) preserve the flow at each node. Constraints (1g) require that the route can arrive at each node at most once, whereas constraints (1f) require that the route can depart from each node at most once. Constraints (1h) guarantee that the model does not output more than one route, i.e., one route that starts at and goes back to the depot with one or more routes between other nodes of the network.

Note that the surrogate model $\mathcal{M}(s)$ does not explicitly handle time windows at nodes and the maximum duration of the route. The idea is to acquire this information:

1. in the objective function by well estimating penalty p_{ij} (section 2.2.3);
2. in the constraints by adding cuts for each solution (section 2.2.4).

2.2.3 Estimation

Let $\mathcal{H}$ be the set of pairs route-penalty (r, p) already observed, where r is a route and p is the penalty associated with route r in a particular implementation. In other words, set $\mathcal{H}$ stores part of the information from previous simulations. It is possible to estimate the value of the penalty p_{ij} per arc (i, j) by taking the average penalty equally distributed among arcs inside a route of routes with arc (i, j). More specifically, it is possible to calculate penalty p_{ij} using the following formula, where $occurrences((i, j), \mathcal{H})$ yields the number of routes in

set $\mathcal{H}$ containing arc (i, j), $belongs((i, j), r)$ yields 1 if arc (i, j) is part of route r and 0 otherwise, and $size(r)$ yields the number of arcs in route r:

$$p_{ij} = \frac{1}{occurrences((i,j),\mathcal{H})} \cdot \sum_{(r,p)\in\mathcal{H}} \frac{belongs((i,j),r) \cdot p}{size(r)}$$

2.2.4 Cuts

In this section we expose the cuts that we employ in our tracker approach.

Cuts for infeasible solutions. Imagine that a solution x^k, found by the tracker approach, represents route r^k $[1, 2, 3, 1]$, and that route r^k is deemed infeasible. It follows by intuition that all routes starting with $[1, 2, 3]$ are likely to be infeasible as well. If infeasibility comes from violating a time window at a certain node, all routes starting with $[1, 2, 3]$ will still likely violate those time windows. If infeasibility comes from violating the maximum duration of the route, adding an extra stop to another node before returning to depot is also likely to violate the maximum duration of the route as the triangle inequality holds. Thus, constraint $x_{12} + x_{23} \leq 1$ could be added to the surrogate model $\mathcal{M}(s)$ to forbid the search for solutions that start with $[1, 2, 3]$.

More generally, let $\mathcal{A}(x^k)$ be the set of arcs in the route represented by solution x^k minus the return to the depot. In the example, $\mathcal{A}(x^k)$ would be $\{(1, 2), (2, 3)\}$. The following cut for an infeasible solution could be added to the surrogate model $\mathcal{M}(s)$ without huge harm for an infeasible solution x^k:

$$\sum_{(i,j)\in\mathcal{A}(x^k)} x_{ij} \leq |\mathcal{A}(x^k)| - 1$$

Cuts for feasible solutions. Imagine that a solution x^k, found by the tracker approach, represents route r^k $[1, 2, 3, 1]$, and that route r^k is deemed feasible. The tracker approach will run a standard number of simulations to identify what is the actual score (reward plus penalties) of route r^k and consider whether route r^k should replace the best route in the current iteration. However, nothing prevents solution x^k from appearing in the next iterations after the adjustment of the penalties in the objective function. In this sense, it is possible to add a cut for a feasible solution to the surrogate model $\mathcal{M}(s)$ to prevent it from considering solution x^k again in the next iterations.

More generally, let $\mathcal{B}(x^k)$ be the set of arcs in the route represented by solution x^k. Note that here the return to the depot is important. In the example, $\mathcal{B}(x^k)$ would be $\{(1, 2), (2, 3), (3, 1)\}$. The following cut for a feasible solution could be added to the surrogate model $\mathcal{M}(s)$ for a feasible solution x^k:

$$\sum_{(i,j)\in\mathcal{B}(x^k)} x_{ij} \leq |\mathcal{B}(x^k)| - 1$$

Impossible nodes. Apparently, some nodes cannot be part of attractive routes. An example would be a node with a time window that opens after the maximum duration of the route. These nodes form an impossible set $\mathcal{I}$. A node i is part of the impossible set $\mathcal{I}$ if a simple route that visits only node i has an average score (rewards plus penalties) smaller than zero. This set of impossibles

nodes is generated at the beginning of the algorithm and these nodes are ignored in the surrogate model $\mathcal{M}(s)$ to decrease the size of the search space.

Risky arcs. There are some arcs that seem to be too risky to be part of attractive routes. More specifically, if it is not possible to reach node j before its closing time from node i while departing at the earliest time possible and considering the worst travel time (i.e., $o_i + t_{ij} > c_j$), then arc (i, j) is considered too risky to be part of a route. This set of risky arcs is generated at the beginning of the algorithm and these arcs are also ignored in the surrogate model $\mathcal{M}(s)$ to decrease the size of the search space.

2.3 Tracker approach

This section presents the components of the surrogate-based tracker approach.

The tracker approach takes into consideration the following parameters:

- K - maximum number of iterations;
- M - number of simulations per iteration;
- f - feasibility percentage threshold to consider a route as feasible or not (e.g., $f = 0.9$ means that from M simulations during a certain iteration k at least 90% of them must certify route r^k as feasible);
- g - gap threshold when considering the upper bound for a route size s.

The tracker approach has the following steps:

1. Eliminate impossible nodes and risky arcs;
2. Set the initial value for the route size $s \leftarrow 2$;
3. Initiate the iteration counter $k \leftarrow 0$;
4. Run the surrogate model $\mathcal{M}(s)$ and obtain a solution x^k;
5. Identify the route r^k represented by solution x^k;
6. Simulate route r^k M times and assess the output;
7. If route r^k is feasible at least f percent of the time, store route r^k and add a cut for the feasible solution x^k. If not, add a cut for the infeasible solution x^k;
8. Calculate the upper bound u^k for size s (namely, solve the surrogate model $\mathcal{M}(s)$ without considering estimated penalties in the objective function);
9. If the gap between the best score and the upper bound u^k at iteration k is less than g, allow larger routes by setting $s \leftarrow s + 1$;
10. Update the historical observation set $\mathcal{H}$ with route r^k represented by solution x^k and the average penalty p^k over M simulations;
11. Update the penalties p_{ij} according to the updated set $\mathcal{H}$;
12. Return to step 4 until $k > K$ or the best score meets the upper bound;
13. Output the feasible route r^k with the best score.

2.4 Improvement step

The tracker approach generates different solutions according to the parameters. In this sense, a genetic algorithm has been written to work with the different solutions outputted by the tracker approach and come up with nicer solutions. A genetic algorithm is used as a random local search policy, and would be very difficult to exploit in stand-alone, as we run it for only a few generations on the warmed-up solutions of the surrogate steps to see if there exists a better solution in the neighborhood. More information about it in Section 3.

2.5 Track qualification

The tracker approach qualifies for the surrogate-based track because it obeys the definition in the problem statement: *Given one instance, previously tried routes, and the reward for those routes, the goal is to learn a model that can predict the reward for a new route.* Given the historical observation set $\mathcal{H}$, the surrogate model $\mathcal{M}(s)$ is progressively updated to predict the reward for a new route of length s. *Then an optimizer finds the route that gives the best reward according to that model, and that route is evaluated, giving a new data point.* The mixed-integer programming solver, CPLEX, solves the current version of the surrogate model $\mathcal{M}(s)$ and outputs a solution that represents a route. The related route is simulated by the tracker approach M times to determine what is the actual score (reward plus penalties) of the route. *Then the model is updated, and this iterative procedure continues for a fixed number of steps. Over time, the model becomes more accurate, giving better and better routes.* The historical observation set $\mathcal{H}$ grows over time and the surrogate model $\mathcal{M}(s)$ yields progressively better solutions due to a better estimation of the penalties and the addition of constraints for visited feasible and infeasible solutions.

3 Genetic algorithm

We used as an improvement step a very small and standard genetic algorithm (GA). The idea is to imitate natural selection by creating a population of solutions and observing its evolution over the course of multiple generations.

3.1 Outer loop and stopping criterion

After the initialisation step, the GA iterates for `generation_num` generations. This parameter serves as a stopping criterion. Each generation comprises the following steps : evaluation, selection, reproduction and mutation.

3.2 Initialisation

Based on solutions found by the tracker approach, the GA constitutes a population of `population_num` solutions. If the tracker has not found enough solutions to fill the entire population, the GA completes it by generating random solutions (random permutations of nodes). If a dictionary of a previous GA run exists, the initialisation step can use it to extract warmed solutions to fill the initial population. Generally, the less random solutions there is, the quicker the algorithm will be to converge, but only a few make a big difference already.

3.3 Evaluation

All solutions in the population are tested `monte_carlo` times with the problem environment function `check_solution(solution)` to get a good idea of the mean objective value obtainable with a solution. The values are stored in a dictionary for further analysis.

3.4 Selection

The solutions are compared and classified according to a selection operator. The best `parents_num` solutions are kept for reproduction, in order to create the next generation of solutions that will be evaluated. The selection operator we used in our algorithm was homemade, but very classic in its taste : it select from a tournament a fraction of the future parents according to their last scores, and completes the other fraction with solutions from the best ancestors (from all previous generations).

3.5 Reproduction

The parents are crossed to create `population_num` new solutions for the next generation. We use the Non-Wrapping Ordered Crossover reproduction operator from Cicirello [2006], as it preserve the order of the nodes, which is a good thing in our case.

3.6 Mutation

With probability `mutation_proba`, the solutions in the population can incur a mutation (which is simply a permutation between two random nodes that are not the first 1). This helps to keep a good variety of solutions in the population.

3.7 Storage of solutions

After the evaluation step, the solutions are stored in a dictionary operated by the `mean_dico.py` file. The dictionary is dumped in a file after every generation to keep a trace of found and tested solutions. When restarting the GA after a first run, it is a good idea to pass the most recent dictionary name to the `dico_filename` parameter. The initialisation step will use it to grab warmed solutions, helping the GA to keep improving the solutions.

3.8 GA Results on test phase

The tracker approach generated 14 solutions with scores ranging from 9.83 to 11.28 that were all used for the initialisation step of the GA. After running `generation_num=5` of the GA with `population_num=25600` and `parents_num=200` with `monte_carlo=100`, we had the `20210707-125553-env-65-6537855_pop-25600-200_gen-4.json` dictionary, that contains a huge number of 11.32 solutions (which we believe is the optimal solution for the Track 1 test instance). The first 11.32 showed up after the fourth reproduction step.

3.9 GA in stand-alone

During the validation phase, the GA was tested in stand-alone. It did not perform very well with a few generations starting from only random solutions, landing best solutions scoring about 4 instead of the expected 8. However, as soon as the warm start was used, the GA started to output very good solutions.

4 Why we can take the last weekend off

As described in Section 2.2.4, we were able to ignore many nodes that we marked as impossible nodes. Every impossible node n_i is defined as having a negative objective for the tour $[1, n_i, 1]$, mostly because of a late opening time o_i, forcing the tour to arrive later than time T. By summing the rewards of the remaining nodes (in green), we were able to obtain a lower bound on the value of the optimal tour. Those lower bounds were of value 8.52 and 11.32 for the validation and test instance respectively. Using the mathematical programming and genetic algorithm approaches, we were able to find solutions that achieve those values. From that point, the only thing that could improve our solution was to try to incorporate an impossible node to our solution. But this sounded difficult from the definition of the impossible nodes. We managed to find some nodes n_j such that the tour $[1, n_i, n_j, 1]$ was faster than $[1, n_i, 1]$ for some impossible nodes n_i, but due to the stochasticity of the problem, we always obtained a small degree of infeasibility, which made it sub-optimal to add any impossible node in the final solutions. We deduce with a very high degree of confidence that the lower bound was also the upper bound of the maximization problem, and that we had a solution reaching that bound. This thesis was further enforced by our analysis of 554 **different** optimal solutions (which could be found in the folder "many11_32" and in the cvs file "collect_test_solutions". The first finding that we achieved by analyzing these solutions (all of which would provide a reward of 11.32) is that each one of them visits 35 nodes, and these 35 nodes are the same in all solutions with some swaps. In particular, as it can be seen in Figure 4, there are 5 main clusters of nodes that swap the visiting position among each other. Moreover, all the solutions started visiting the sequence of nodes [32,45,55] and visited as a last node 48 (before returning to the node depot 1).

5 Conclusion

To conclude, we believe the combination of the surrogate model and the genetic algorithm serves best as the optimal method for the TSP problem with time windows and stochastic travel time. Results from the test phase showed that we were able to reach the maximum upper bound for numerous solutions, therefore it would be interesting to compare the other teams methods with ours when the maximum reward is hard to reach or when the instance is very large.

6 Appendix

6.1 Dynamic Programming Approach

Here, our approach is to compute the actual travel time between nodes, and the expected rewards when departing at node i at time t to reach node j. Based on

this information, we select the next best node to visit according to a ratio of the expected reward to the travel time to reach node j from node i at time t. Time t is updated at each visited node using the *check_performance* function, with the current solution returning to the depot at node 1. The current time is then modified to subtract the estimated time to travel from the last added node to the depot. In order to reduce the computing complexity, we have removed all nodes j^* which on average yield negative rewards using the solution $[1, j^*, 1]$. Therefore, we will refer to the number of nodes n^* as the final number of nodes to use, which on average was observed to be $30-40\%$ smaller than the original network.

To retrieve the actual travel time between node i and node j, we made use of a modified instance. This modified instance had the following changes:

- Set the distance between node 1 to i to 0 : $d_{1i} = 0$
- Set the distance between node j to 1 to 0: $d_{j1} = 0$
- Set the window opening time of node i to 0: $o_i = 0$
- Set the window opening time of node j to 0: $o_j = 0$

Using the modified instance, we have used the *check_performance* function for solution $[1, i, j, 1]$ $\forall$ i, j and stored the tour time for each edge. Given the stochasticity of the travel time, we have stored a number of 100 time simulations in a matrix named TT of dimension $100 \times n^* \times n^*$, where $TT[i, j, k]$ is the k-th simulation of the travel time between node i and node j.

To compute the average reward matrix E, we made the following changes to the original instance:

- Set the distance between node 1 to i to 0 : $d_{1i} = 0$
- Set the window opening time of node i to t: $o_i = t$

We then used the solution $[1, i, j, 1]$ $\forall$ i, j as an input for the *check_performance* function with the modified instance, and stored the average objective in matrix E. Therefore, input $E[i, j, t]$ refers to the average reward of visiting node j when leaving node i at time t.

After computing matrix E and TT, we made a tour starting at node 1 (solution $[1, 1]$) and adding the next best node to visit according to the following formula:

$$j = \underset{j}{argmax} \frac{E[i, j, t]}{max(t + median(TT[i, j, :]), o_j) - t} \ ,$$

where the denominator $max(t + median(TT[i, j, :]), o_j) - t$ refers to the median travel time between node i and j including the waiting time until the time window opening. Once the current time exceeds the maximum tour time or once every possible reward is negative, we return to the depot and stop the visit.

This approach showed very good results for the validation phase, but very poor results in the test phase. The main problem with this method is that it is a one-step look ahead approach, and therefore acts as a greedy algorithm at each

step without accounting for the next nodes to visit afterwards. Furthermore, we were able to prove that this method severely overfitted the validation set, which explains why it performed so poorly at the test phase. An alternative two-steps look ahead was also developped, but showed even worst results than the initial one-step look ahead.

6.2 Nearest Neighborhood Approach

At the beginning of our investigation we also explored the possibility to apply the Nearest Neighborhood method (NNeigh). It performed with very poor (negative) outcome. We further pruned away nodes that were not compatible with time windows and the result was not still satisfiable.
To conclude, we even modified the algorithm in the following way: It would explore as a first node the furthest feasible node, and then it would proceed through the standard NNeigh method. Also this variation scored poorly.

References

Vincent Cicirello. Non-wrapping order crossover: An order preserving crossover operator that respects absolute position. volume 2, pages 1125–1132, 07 2006. doi: 10.1145/1143997.1144177.

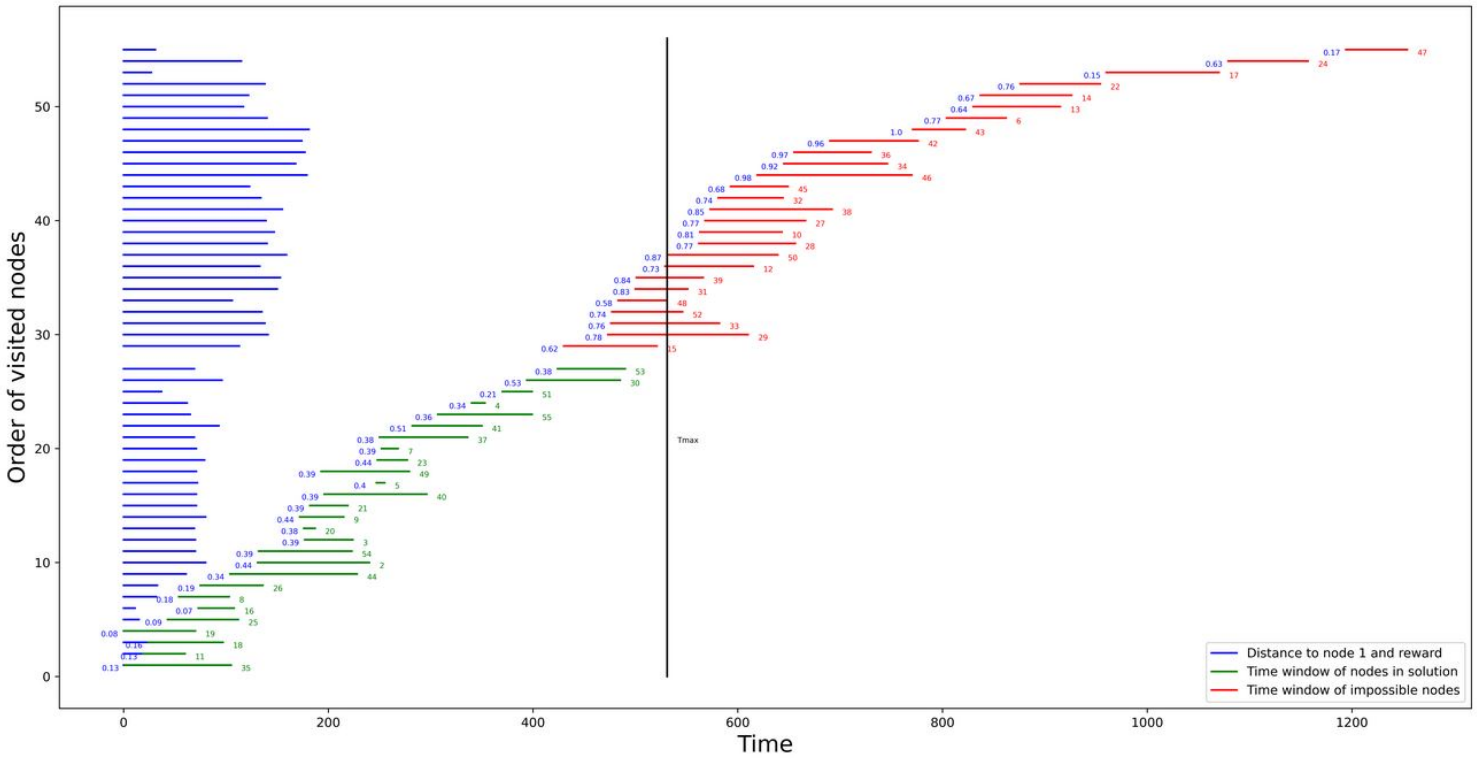

Fig. 2: (a) Best solution found on validation instance.

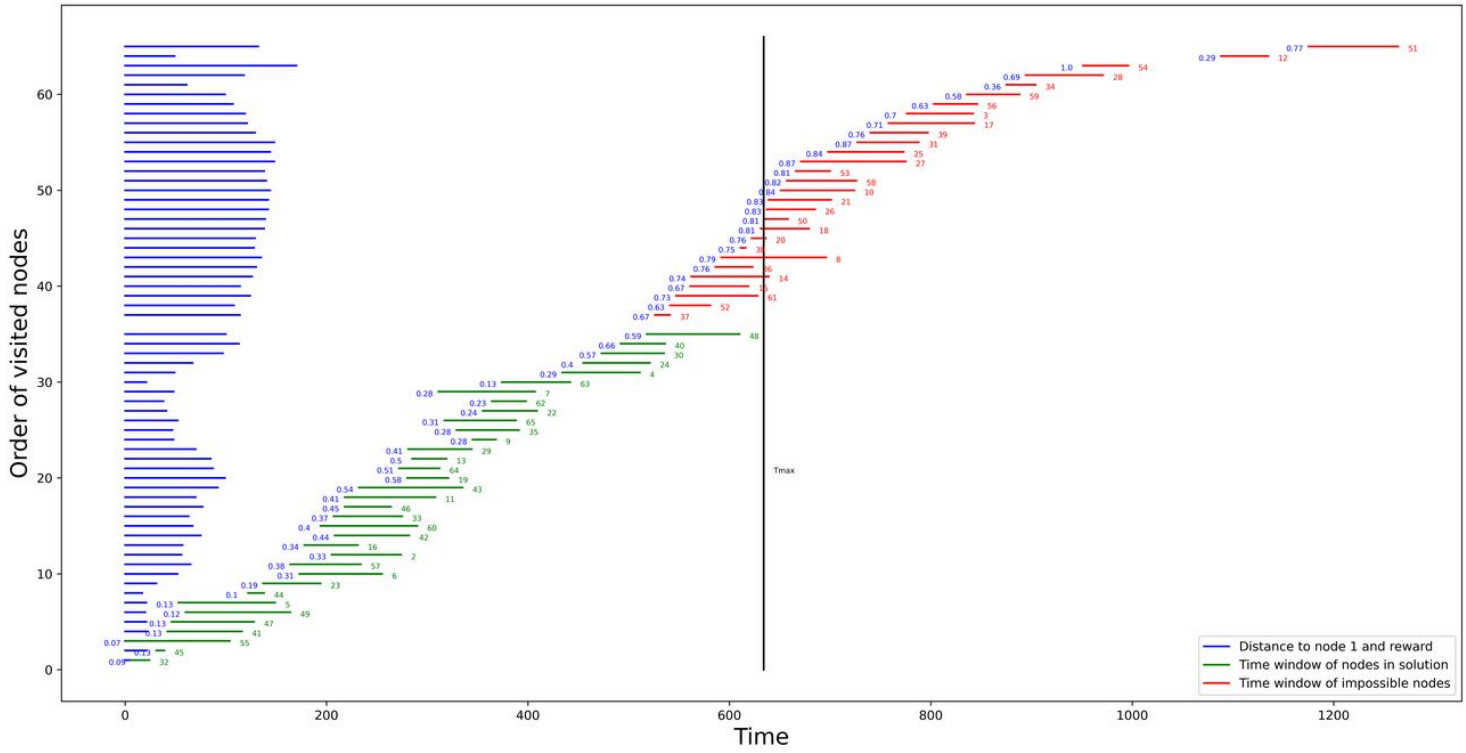

Fig. 3: (b) Best solution found on test instance.

Numbers and positions	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
32	1																																		
45		1																																	
55			1																																
47				0.319	0.123	0.094	0.464																												
5				0.002	0.289	0.433	0.276																												
41				0.188	0.525	0.287																													
49				0.491	0.063	0.186	0.26																												
44								1																											
23									1																										
6										0.958	0.038		0.002	0.002																					
57										0.042	0.827	0.132																							
2											0.002	0.819	0.161	0.018																				
16											0.132	0.045	0.819	0.004																				
42														0.834	0.043	0.023	0.016	0.083																
6											0.002	0.004	0.014	0.103	0.749	0.125	0.002	0.002																
33													0.004	0.04	0.204	0.713		0.04																
46															0.002	0.063	0.924	0.011																
11															0.002	0.076	0.058	0.865																
43																			0.874	0.02	0.067	0.04												
19																			0.034	0.857	0.034	0.074												
64																			0.02	0.101	0.845	0.034												
13																			0.072	0.022	0.054	0.852												
29																							1												
35																								0.251	0.202	0.359	0.081	0.099	0.007						
65																								0.307	0.217	0.152	0.202	0.088	0.034						
9																								0.188	0.473	0.222	0.099	0.018							
22																									0.04	0.141	0.475	0.209	0.135						
62																										0.029	0.034	0.542	0.395						
7																								0.255	0.069	0.097	0.108	0.043	0.428						
63																														1					
4																															1				
24																																1			
3																																	0.948	0.052	
4																																	0.052	0.948	
48																																			1

Fig. 4: Distribution of the 35 visited nodes from 554 different optimal solutions. Each row is a node, each column is the average visiting position in the path.