ISBN: 978-628-95207-8-1. ISSN: 2414-6390. Digital Object Identifier: 10.18687/LACCEI2024.1.1.885

A Bi-objective A* minimizing travel times and failure rates

Sergio Troncoso-Duque, B.S.*, Elizabeth Montero, PhD†, and Franco Menares, B.S.*
*Universidad Andres Bello, Viña del Mar, Chile, s.troncosoduque@uandresbello.edu, f.menaresfuentes@uandresbello.edu
†Universidad Técnica Federico Santa María, Valparaíso, Chile, elizabeth.montero@usm.cl

***Abstract*—Routing problems are present in much activities in modern cities. These problems not only aim to minimize transportation costs and times, they also consider objectives that depend on time itself. These issues tend to be highly complex, and current solution methods require a lot of effort or are able to only deliver sub-optimal solutions. This research proposes a bi-objective algorithm based on BOA* to tackle the bi-objective asymmetric TSP. TDBOARR is aimed to minimize both, traveling times and rejection rates within the urban logistics landscape of Santiago de Chile. We analyze the performance of our approach comparing their results with a gurobi implementation of the mathematical model. TDBOARR shows being efficient finding the Pareto front extreme solutions, but unable to find most of central solutions.**

***Index Terms*—traveling salesman problem, bi-objective A*, MILP, real-world problem instances.**

I. Introduction

Last-mile deliveries refer to the final process in a supply chain where the product is sent from a distribution center to the customer. This stage of the process is the most expensive, and full of situations that can cause failure when handling various uncontrollable variables that affect its normal course. A failure within a last-mile delivery could not only damage the product itself but also affect the customer experience.

A way to model this situation is through the Traveling Salesman Problem (TSP) or Vehicle Routing Problem (VRP).

This problem can become complicated as it approaches real-world. Moreover, when additional objectives as minimizing the failure or rejection rate of delivery, which usually varies during the hour of the day the route is executed. All these new variables increase the difficulty of a problem known to be NP-Hard. Given these factors, the possibility of finding an algorithm, not only capable of delivering a group of valid and optimal solutions, but also at a manageable speed is crucial for any type of intelligent system that acts on the last mile. These conditions change the nature of the problem to a bi-objective one.

Currently, exact methods tend to be slow in obtaining results when handling the huge combinatorial space of these types of problems in an naive manner or are based on integer linear programming (ILP), which requires modeling each variable of the problem beforehand. On the other hand, heuristic search has helped to generate a quick, simple, and accurate option for these types of problems, making them an attractive option for more complex problems that require greater sophistication.

Being Santiago the city of Chile with the highest percentage of this type of shipments makes it the perfect scenario to validate a new algorithm in a real environment. This work proposes a bi-objective algorithm based on BOA* that can solve instances of a bi-objective asymmetric TSP minimizing travel times and rejection rates. The approach is compared with a classic approach of an ILP model solved by Gurobi.

The problem studied is presented in Section II. Main studies related to the present work is summarized in Section III. Our approach is presented in Section IV. Experimental evaluation is presented in Section V. Conclusions and future work are presented in Section VII.

II. Problem statement

The Traveling Salesman Problem (TSP) is one of the most studied combinatorial optimization problems. TSP aims to find the shortest Hamiltonian cycle in a graph, meaning visiting each node once and returning to the original starting point. The bi-objective version of the problem adds a new value in each arc between each pair of nodes.

The bi-objective Traveling Salesman Problem (bTSP) can be defined by a graph $G = v_1, v_2, ..., v_n$ of cities and two values of cost $c1(v_i, v_j)$ and $c2(v_i, v_j)$ between each pair of cities (v_i, v_j). Given that the travel salesperson has to visit each city exactly once and return to the city of origin, the goal here is to obtain the minimal tour for both values. If $c_k(v_i, v_j) = c_k(v_j, v_i)$ for $i, j \in N$ we consider the problem a symmetric bTSP. Most multi-objective problems face conflicting objective functions, if a trade-off between objectives happens, delivering a set of tour solutions to each problem instances. Considering two solutions, $S1$ and $S2$ and two objectives functions being minimized, if $S_1(c_1) < S_2(c_1)$ and $S_1(c_2) > S_2(c_2)$, then the solution $S_1(c_1, c_2)$ and $S_2(c_1, c_2)$ are non-dominated. The set of all non dominated solutions are known as the Pareto frontier. Finding the Pareto frontier is NP-hard [1] therefore adds complexity to an already NP-hard problem [2].

The asymmetrical version of the problem considers the same graph G but with $c_k(v_i, v_j) \neq c_k(v_j, v_i)$. In this version of the problem it could be even possibly to not have a path from $c_k(v_i, v_j)$, but only a value for $c_k(v_j, v_i)$.

Fig. 1 shows an example for a bTSP where each city is connected to the other by an edge. The tuple with two values over each arc show the values c_1 and c_2 related to the two objective functions considered. The search tree in Fig. 2 shows all the possible tours for the instance, considering

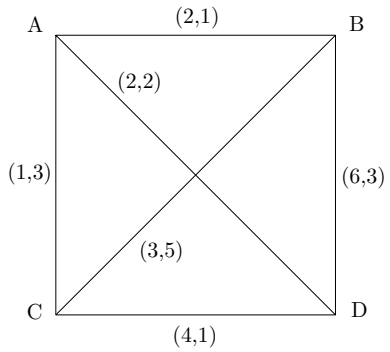


Fig. 1: An instance of bTSP of 4 cities with A acting as start city.

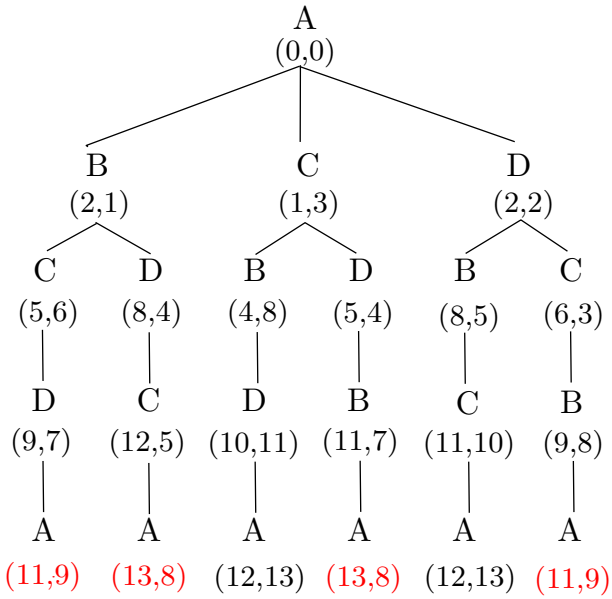


Fig. 2: Search tree for the instance shown in Fig. 1. Each node represents a sub-tour of the instances depicted in the tree with the letter of the last visited city. Below each node is a tuple with the pair of the cumulative values of each objective. In red are shown the set of solutions conforming the Pareto frontier.

the node A as the starting city. Three different solutions (in terms of objective functions were found), solutions $S_1(11, 9)$, $S_2(12, 13)$ and $S_3(13, 8)$. Here S_1 dominates S_2 because it has better c_1 and c_2, while S_1 and S_3 are non-dominated because S_1 has a better c_1 and S_3 has a better c_2. In red are shown the set of non-dominated solutions conforming the Pareto frontier.

Mathematical model of the problem being solved is presented below.

A. Mathematical model

Binary variables x_{ij}^m control the use of edges in their corresponding time-period, while real valued variables t_i control the time sequence of visits.

- Variables:

$$x_{ij}^m = \begin{cases} 1 \text{ if uses arc } (i,j) \in A \text{ in time-period } m \in M. \\ 0 \text{ otherwise.} \end{cases}$$

t_i : Time leaves the node $i \in N$.

- Objective functions: Function f_1 computes the total time, while function f_2 computes the total failure rate.

$$f_1 = Min \sum_{(i,j)\in A} \sum_{m\in M} t_{ij}^m * x_{ij}^m \quad (1)$$

$$f_2 = Min \sum_{(i,j)\in A} tf_i^m * x_{ij}^m \quad (2)$$

- Flow-balance constraints: Equations (3) and (4) control arriving and leaving the starting/ending node exactly once, while equations (5) and (6) control the same for each node in the problem.

$$\sum_{j\in N/(o,j)\in A} \sum_{m\in M} x_{oj}^m = 1 \quad (3)$$

$$\sum_{i\in N/(i,s)\in A} \sum_{m\in M} x_{is}^m = 1 \quad (4)$$

$$\sum_{i\in TN/(i,j)\in A} \sum_{m\in M} x_{ij}^m = 1, \forall j \in N \quad (5)$$

$$\sum_{j\in TN/(i,j)\in A} \sum_{m\in M} x_{ij}^m = 1, \forall i \in N \quad (6)$$

- Sequence constraints: Equations (7) and (8) control the visit times of nodes in the sequence of visits, while constrains (9) and (10) control the time-periods allocated to each visit.

$$t_j \geq t_i + t_{ij}^m - B * (1 - x_{ij}^m), \forall (i,j) \in A, m \in M \quad (7)$$

$$t_j \leq t_i + t_{ij}^m + B * (1 - x_{ij}^m), \forall (i,j) \in A, m \in M \quad (8)$$

$$t_i \geq TI_{ij}^m - B * (1 - x_{ij}^m), \forall (i,j) \in A, m \in M \quad (9)$$

$$t_i \leq TF_{ij}^m + B * (1 - x_{ij}^m), \forall (i,j) \in A, m \in M \quad (10)$$

- Domain constraints: Equations (11) and 12 establish the nature of the variables.

$$x_{ij}^m \in \{0, 1\}, \forall (i,j) \in A, m \in M \quad (11)$$

$$t_j \geq 0, \forall j \in N \quad (12)$$

III. Related work

Multi-objective problems have been studied in urban logistics since its origins [3] given its strong relationship with the urban transport systems. The asymmetric traveling salesman problem with time-dependent and rejection rates is a new problem in the field of trading. A similar variant is the traveling salesman problem (TSP with profits) defined in [4] introduces the gain as a new objective in addition to finding the shortest path. The more important difference here is that profits must be maximized while rejection rates must be minimized. Another related problem is the traveling salesperson with quotas (Quota TSP [5]) who introduces the value p_{min}, which generates a dependency on the utilities since a given quota must be fulfilled in the complete tour.

The classic version of the bi-objective traveling salesman problem is introduced in [6]. Here they define the possibility of expanding one or more criterion, either by minimizing or maximizing it, obtaining solutions from the Pareto set.

Meta-heuristics have been extensively used to solve bi-objective TSP problems. There are various sub-categories of meta-heuristics that allow obtaining high-quality results. [7] uses methods of genetic evolution to successfully solve the multi-target TSP problem. [8] uses the Ant Colony algorithm to emulate the behavior of ants' pheromones in order to find the best routes according to the objectives to conform a Pareto frontier. Another example is the use of Artificial Bee Colony algorithms to solve the multi-target TSP problem [9]. The algorithm emulates the behavior of bees when they search for pollen, leaving traces for other bees to mark the spots with the highest amount of pollen. These points in the algorithm are seen as cities and, it is executed for each objective function to form the Pareto frontier.

[10] developed a way to solve bi-objective problems with integer linear programming. Other popular solvers are those provided by Gurobi [11] and CPLEX algorithms, which use variations of the *branch-and-bound* algorithm for the optimization process. Some versions of this solver have been expanded to multi-criteria problems [12].

First-best methods for bi-objective problems have been used before with good results. [13] addresses the Steiner shortest path problem with results up to 200 cities. This version of the problem is a variant that mixes the Steiner tree problem with the shortest path problem. In this case it not only required a group of nodes, but also its minimum spanning tree. The problem is not directly a TSP problem but it is considered within the category of routing problems. Another algorithm that was used in a bi-objective shortest path environment was BOA* [14]. In [14] its performance wast tested on the map of the city of New York and compared to other first-best algorithms such as MOA* [15], NAMOA* [16] and BBDijkstra [17] among others.

In this work we study a heuristic search technique capable of solving the asymmetric traveling salesperson problem with time-dependent rejection rates and adapt the BOA* algorithm to consider rejection rates for delivery points and times travels based on Santiago de Chile's road grid. Here, we adapt a bi-objective version of the well known A* algorithm, BOA* [14], to consider the delivery rejections rates and time as both of the objectives to optimize.

IV. Proposed approach

BOA* follows the presentation of more modern descriptions of A* such as those in [18]. An important aspect of BOA* is its ability to compute cost-unique Pareto-optimal solutions to deliver one representative solution for all cost-identical solutions, minimizing the space of the Pareto-optimal set.

The $OPEN$ list of BOA*, like in A*, contains nodes that represent states. Each node x has a state $s(x)$, a g-value tuple $g(x_1, x_2)$, an f-value tuple $f(x_1, x_2)$, a h-value tuple $h(x_1, x_2)$ and a parent $parent(x)$. The algorithm saves the value $g_2^{min}(s)$ for each state s, which is the smallest g_2-value for the expanded node s. This can remove unnecessary nodes to be added to the $OPEN$ list and improve the efficiency in run-time, but also the size in memory the algorithm utilizes.

The algorithm takes as input a bi-objective search problem and a *consistent* heuristic function to compute the cost-unique Pareto-optimal solution set. A consistent heuristic is a heuristic function which always provides an estimate that is no greater than the actual cost of reaching the goal state. If, for every node in the search space, the estimated cost of reaching the goal state from that node is not greater than the current cost plus the estimated cost of reaching the goal state from the node's successor. An efficient heuristic is a heuristic function that helps the algorithm make *smart guesses* in order to find an optimal solution to the problem more quickly and using less computational resources. For a heuristic function to be called efficient it requires to be quick to compute, a good approximation of the true cost to reach the goal (as explain above) and designed specifically for the problem at hand.

Next, BOA* extracts a node x from the $OPEN$ list, following a lexicographical order, the smallest value for f of all nodes. The node is expanded if its vale of g_2 is bigger than $g_2^{min}(s(y))$ or if its f_2-value is bigger than $g_2^{min}(s_{goal})$. If the extracted node x is the goal state, then BOA* has found a non-dominated solution and adds the node x to the solution set $sols$. Otherwise, it calculates the corresponding x's children and adds them to the $OPEN$ list. Child nodes y whose g_2-value is at least $g_2^{min}(s(y))$ or its f_2-value is at least $g_2^{min}(s_{goal})$ are not added. The algorithm terminates when the $OPEN$ list is empty and returns the solution set.

A. Heuristics Functions

1) In-out Heuristic: For the objective c_1 (time) the algorithm use the *In-out* heuristic proposed in [19] to solve the TSP problem. The main objective of this heuristic function is to estimate the distance from a current node to the goal node based on the number of nodes that are *inner* and *outside* the shortest path from the current node to the goal node. This would give a good indication of how much is left in the tour. The main objective of this heuristic function is to provide a way for the algorithm to make an informed guess about which path will lead to the solution in the most efficient way.

The pseudo code for the implemented *In-out* heuristic function is shown in algorithm 1. To keep the heuristic in its highest performance, we pre-sorted the arcs according to the lower cost for each city. These values were stored in a 2-D matrix defined as s_edges. The main loop on the function iterates all the remain-to-be-visited cities and calculates the sum of the two shortest edges that are not connected to *inner* cities on the sub-tour. Inner cities are those already visited and neither the initial city nor the last city visited. Finally, the first and last city (of the sub-tour) shortest edges are added and the result is returned.

2) Proposed Heuristic: This implementation of BOA* for objective c_2 (rejection rate) uses a new proposed heuristic. In the problem of finding the fastest route from a series of location to try minimize the rejection rate of each delivery, the rejection rate calculated at arriving for each delivery points is affected by the time of arrival. In this case, a heuristic function that only considers the distance between two locations

Algorithm 1 In-out Heuristic

function $h_1(n, visited)$
  $sum \leftarrow initializedwith0$ $s\_edges \leftarrow$ sorted arcs by minor cost **for each** $n' \notin visited$ **do**
    if $n' \neq n$ **then**
      for each $i \notin inner(n')$ **do**
        $sum \leftarrow sum + s\_edges[i]$
  $sum \leftarrow sum + s\_edges[n] + s\_edges[initial\_city]$ **return** sum*0.5*

would not be sufficient to find the optimal solution. A heuristic function that considers temporal constraints can provide more accurate estimations of the time required to reach the goal state and lead to better solutions. The idea here is to create an estimator for the rejection rate that assumes a path for the missing nodes trying to estimate the rejection rate. The estimator uses the value c_1 of the current node to compute the initial time frame, which is the same value of g. Then it computes the estimated final frame with the sum of c_1 for the nodes not in the sub-tour in lexicographic order. Finally the heuristic value is the sum of the minimum values for each city n not in the sub-tour between the initial time frame i and the estimated final time frame j. As we choose the lowest rejection rate among both time frames we assure the estimation remains admissible. The heuristic function shown in algorithm 1 receives two parameters: n represents the next city to be visited and a boolean array with value 1 if the city was already visited. As in the regular in-out heuristic, the edge values are sorted to obtain the minimum rejection rates. Then, the lexicographic revisions explained above are run to calculate the approximated remaining time (stored in sum) and the approximated remaining rejection rate (return as the variable $sum2$).

Algorithm 2 Proposed Heuristic: Time-Dependent In-Out

function $h_1(n, visited)$
  $sum \leftarrow initializedwith0$ $frame_{init} \leftarrow getTimeFrame(g(n))$
  $s\_edges \leftarrow$ sorted arcs by minor cost **for each** $n' \notin visited$ **do**
    if $n' \neq n$ **then**
      for each $i \in inner(n')$ **do**
        $sum \leftarrow sum + s\_edges[i]$
  $frame_{max} \leftarrow getTimeFrame(sum)$ **for each** $n' \notin visited$ **do**
    $frame_{min} \leftarrow \infty$ **for** $j \leftarrow frame_{init}$ **to** $frame_{max}$ **do**
      if $frame_{min} < ratesLookupTable[j]$ **then**
        $frame_{min} \leftarrow ratesLookupTable[j]$
    $sum2 \leftarrow sum2 + frame_{min}$
  return $sum2$

B. Proposed Algorithm: TDBOARR

We introduce a new variant of BOA* to tackle the bi-objective nature of the problem being solved in this work. First, our time-dependent version introduces a new simplified version of the revisions in which we only focus on eliminating nodes based on the g_2^{min}. Second, to compute the rejection rate as a time-dependent variable, ***TDBOARR*** utilizes the function $rate\_lookup(g_1)$, implemented in algorithm 4 that takes the value of g_1, the accumulated time to move to the next city, and evaluates the rejection rate using the function $transform(g)$. This function is selected based on both the metric of g_1 and how the table is composed. For example, if the g_1 symbolizes accumulative time (in minutes) and the rejection rate table is divided on hours of the day, the function might be the truncated transforming from minutes to hours, so the returned value showcase the hour of day corresponding to a rejection rate. Finally, the variable t_0 is introduced as a form to change the initial time the paths start; initializing the value of g_1 without the rejection rate associated, so the remaining rejection rates are calculated with an offset value. The main changes to the original BOA* can be summarized as:

1) The $rate_lookup$ function to calculate the heuristic in a time-dependent fashion, and
2) The $prune$ function to optimize the construction of the Pareto frontier. For prune, the implementation in algorithm 5 shows the procedure that is executed in the main loop for the TDBOARR and cleans the $OPEN$ queue for lower values of f_2 than the previously obtained solution. This, because these nodes would generate dominated solutions if are expanded, so we can prune them to reduce space complexity, less memory space and faster solutions.

Algorithm 3 Time-dependent BOA* with Rejection Rate

function *TDBOARR*$(S, E, c, s_{start}, s_{goal}, t_0)$
  $sols \leftarrow \emptyset$ **for each** $s \in S$ **do**
    $g_2^{min} \leftarrow \infty$
  $x \leftarrow$ new node with $s(x) = s_{start}$ $g(x) \leftarrow (t_0, 0)$ parent(x) $\leftarrow$ null f(x) $\leftarrow (h1(s_{start}, h2(s_{start}))$ Initialize $OPEN$ and add x to it **while** $OPEN \neq \emptyset$ **do**
    Remove a node x from $OPEN$ with the lexicographically smallest f-value of all nodes in $OPEN$ **if** $g_2(x) \geq g_2^{min}(s(x))$ **then**
      continue
    $g_2^{min}(s(x)) \leftarrow g_2(x)$ **if** $s(x) = s_{goal}$ **then**
      Add x to $sols$ **continue**
    for each $t \in Succ(s(x))$ **do**
      $y \leftarrow$ new node with $s(y) = t$ $g(x) \leftarrow g(x) + c(s(x), t)$ $g(y) \leftarrow rate\_lookup(g(x))$ $f(y) \leftarrow g(y) + h(t)$ **if** $g_2(y) \geq g_2^{min}(t)$ **then**
        continue
      Add y to $OPEN$
  return $sols$

Algorithm 4 Look-up function for time-dependent heuristic

function *rate_lookup(g_1)*
  $rate\_idx \leftarrow transform(g)$ $g_2 \leftarrow lookup\_table[rate\_idx]$ **return** g_2

Algorithm 5 Prune function for queue optimization.

function *prune($OPEN$)*
  $last\_sol \leftarrow$ get last solution from $sols$ **for each** $n \in OPEN$ **do**
    if $f_2(n) \geq f_2(last\_sol)$ **then**
      Remove n from $OPEN$

V. Experimental results

For TDBOARR all the experiments were run in a AWS EC2 m5.large instance running g++8. The source code implementation can be found in a Github repository.[1]

The problem instances were generated based on the vehicle routing problem instances presented in [20]. Two types of problem instances were considered. The *a* instances that include 5 clients each one, and two *b* instances that include 10 clients each one. The geographical distribution of the clients in these problem instances is shown as red dots in figures 3 and 4. Moreover, for each of these problem instances, the interval of times considered in each one is shown in table I. Moreover, seven versions of each problem instances were considered by changing the initial operational time of the traveling salesman process. For this, we considered as starting times the instance initial time (0.0h) and 0.5, 1.0, 1.5, 2.0, 2.5 and 3.0 hours after the corresponding starting time.

TABLE I: Time intervals

Instance	#clients	planning horizon
a-Instance1	5	12:00:00 PM - 04:00:00 PM
a-Instance2		03:00:00 PM - 07:00:00 PM
a-Instance3		09:00:00 AM - 01:00:00 PM
a-Instance4		11:00:00 AM - 03:00:00 PM
a-Instance5		05:00:00 PM - 09:00:00 PM
a-Instance6		10:00:00 AM - 02:00:00 PM
a-Instance7		11:00:00 AM - 03:00:00 PM
a-Instance8		01:00:00 PM - 05:00:00 PM
a-Instance9		12:00:00 PM - 04:00:00 PM
a-Instance10		02:00:00 PM - 06:00:00 PM
b-Instance1	10	10:00:00 AM - 02:00:00 PM
b-Instance2		01:00:00 PM - 05:00:00 PM

VI. Results

The plots in figures 5 and 6 show the solutions found by the mathematical model (M) resolution using gurobi solver in yellow and the solutions found by the algorithm (A) TDBOARR in green for the *a* problem instances. In all cases a set of seven initial times were considered. These initial times are shown in different intensities in the color previously indicated. X axis show the normalized travel times of solutions and y axis show their normalized failure rates.

First, it is interesting to notice that the mathematical model was able to find at least two different solutions in 7 out of 10 problem instances. In instance *a-Instance1* when the starting time considered is 0.5 hour after the initial time, i.e. at 12:30:00 PM, it is possible to find three different solutions in the corresponding Pareto front. In instance *a-Instance4* there are two starting times that generate Pareto fronts with three solutions. Also, in problem instance *a-Instance9* there are four starting times that generate Pareto Fronts with three solutions. The existence of more solutions in the corresponding Pareto Fronts can be associated to the difficulty of the problem instance. In this case it is possible to observe that these three problem instances present different geographical distributions patterns, but all consider planning horizons that are very close to the noon, the time of the day with the highest levels of traffic congestion.

Moreover, from these results it is possible to observe that TDBOARR obtains equivalent solutions compared to those found by the solver applied to the mathematical model, but fails to obtain the other extreme solution of the Pareto front when it exists. In problem instances *a-Instance1*, *a-Instance2*, *a-Instance3*, *a-Instance4*, *a-Instance5*, *a-Instance9* and *a-Instance10* there is at least one starting time that generates a Pareto Front with two or more extreme solutions. One that minimizes the travels times and the other one that minimizes the failure rates. In some cases, the TDBOARR was able to find the solutions that minimize the travel times (*a-Instance1*, *a-Instance2*, *a-Instance3*, *a-Instance4*, *a-Instance5* and *a-Instance9*).

This behavior can be explained by analyzing the prune methods the algorithm implements, specifically the Pareto dominance revision for expanding nodes in the main search, the loop that iterates through $OPEN$ in algorithm 3. The method compares the $f2$ value of the selected node with the $g2_{min}$ value and prunes the node if the current $f2$ is greater than the later one. A detailed analysis shows that the heuristic value calculated for the second objective shown in algorithm 2 in some problem instances, like *a-Instance2* in Fig. 3b, overestimates the final rejection rate that the branch is going to obtain, making the value of $f2$ greater than it its supposed to be. With this, the selected node is discarded for expansion and the heuristic becomes not admissible. This might be triggered by the assumption that the lexicographical order to calculate the time remaining is not a good quantifier when handling time-dependent objectives as the original in-out heuristic values are based on the current time, unlike the proposed heuristic.

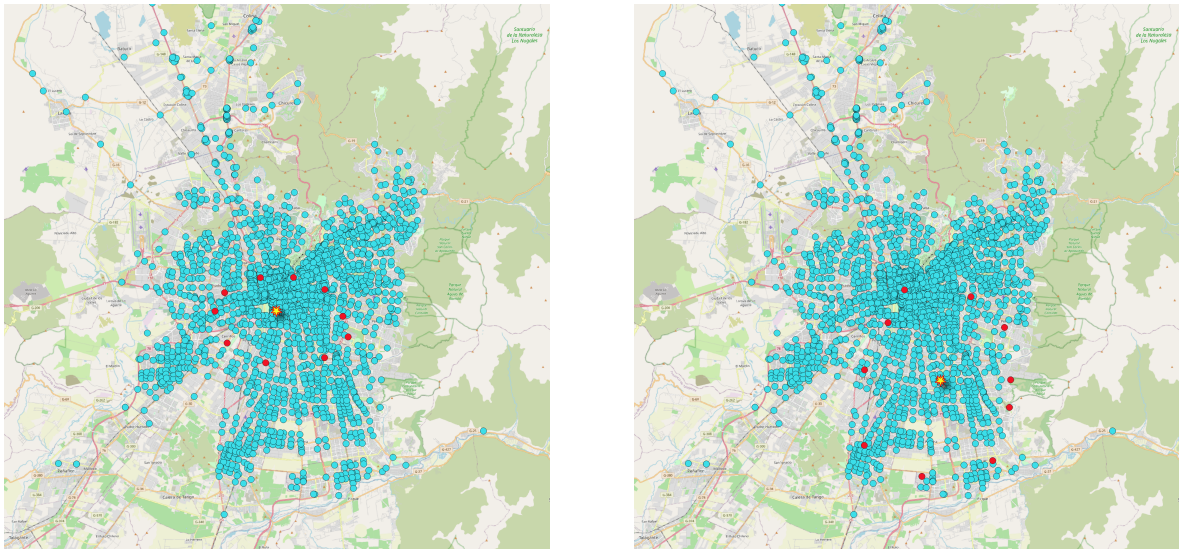
A general overview of the results shows that the Gurobi solver exhibits more consistency across all instances in obtaining solutions on the Pareto frontier, while TDBOARR's performance varies, often yielding only one result. In some instances, such as *a-Instance1*, TDBOARR appears to perform comparably to Gurobi, but in others, like a-Instance6, it is clearly outperformed. Since the objective is to minimize both

[1]Code Repository: https://github.com/stroncod/LastMileBOAstar

(a) *a-Instance1* (b) *a-Instance2* (c) *a-Instance3* (d) *a-Instance4* (e) *a-Instance5*

(f) *a-Instance6* (g) *a-Instance7* (h) *a-Instance8* (i) *a-Instance9* (j) *a-Instance10*

Fig. 3: Problem instances *a-Instance1 to a-Instance10*



(a) *b-Instance1* (b) *b-Instance2*

Fig. 4: Problem instances *b-Instance1 and b-Instance2*

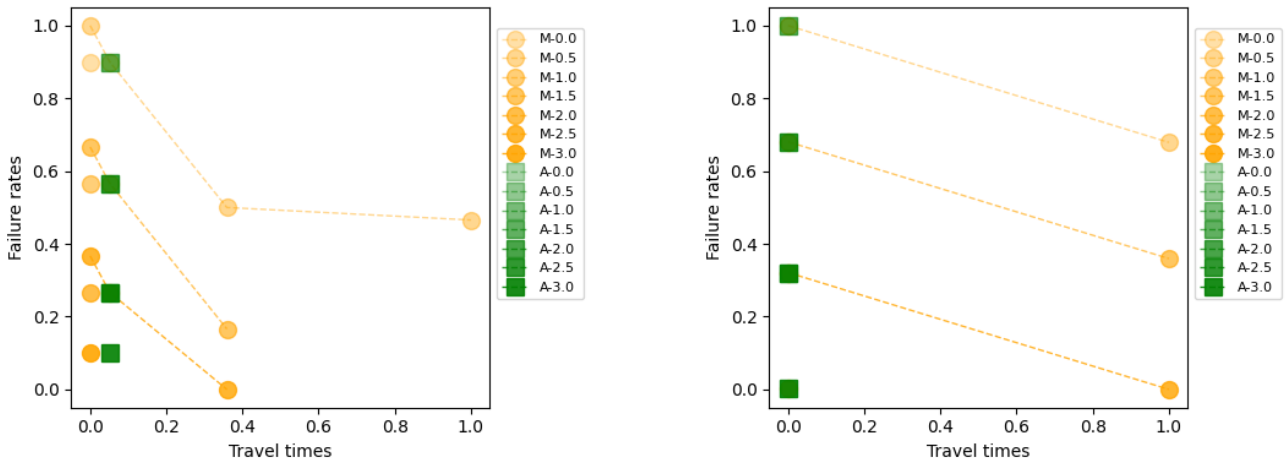
travel times and failure rates, solutions closest to the origin (the bottom-left corner) are considered the best. Gurobi seems to achieve this more effectively.

For *a-Instance1*, TDBOARR seems to match the first solutions found for the solver but the rest of the Pareto frontier is not achievable by the algorithm. Similar behavior is again shown in *a-Instance2*, *a-Instance3*, *a-Instance4* and *a-Instance5*, caused by the lacking of a good heuristic that guides the algorithm to be able to find the rest of the Pareto frontier, as the heuristic is vital to traverse in the search tree for other solutions that might arise in the frontier.

This behavior is explained by the composition of the heuristic. The lexicographical order used to compute the failure rate component does not adapt to the time domain nature of the problem, as can be seen in the results where no more than one solution is found on the Pareto frontier. Another problem is reflected, for example, in the instance *a-Instance6*, where both algorithms find the solution, but the values suffer an offset between the solver and TDBOARR. This is due to the floating nature of the values that can cause the values not to be exact between algorithms but very similar to each other; this behavior is observed for instances *a-Instance7*, *a-Instance-8*, *a-Instance9*, and *a-Instance10*.

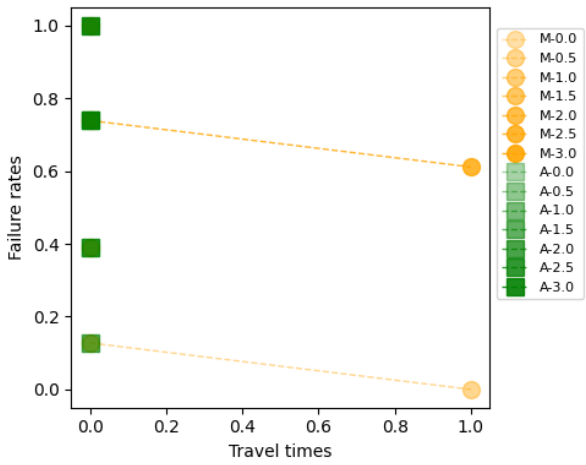
Last, in problem instances *a-Instance7* and *a-Instance8* most times the TDBOARR was able to find the optimal solution found by the mathematical model for most initial times considered.

Table II show the execution times of both, the solver using

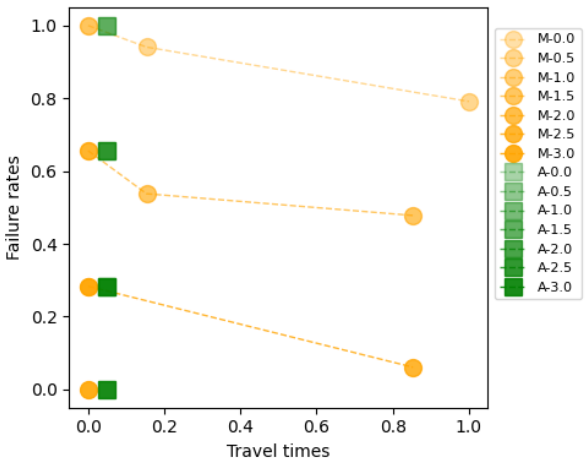


(a) *a-Instance1*

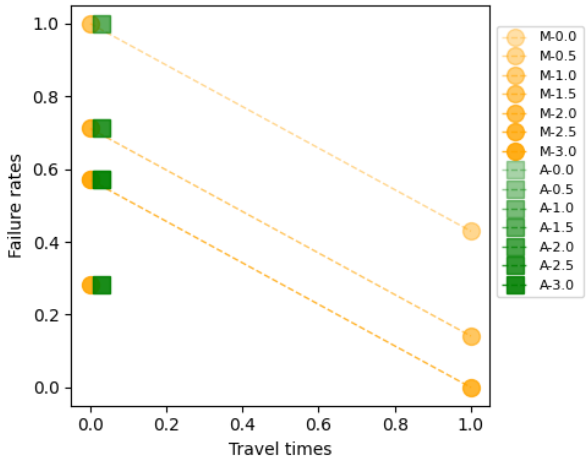
(b) *a-Instance2*



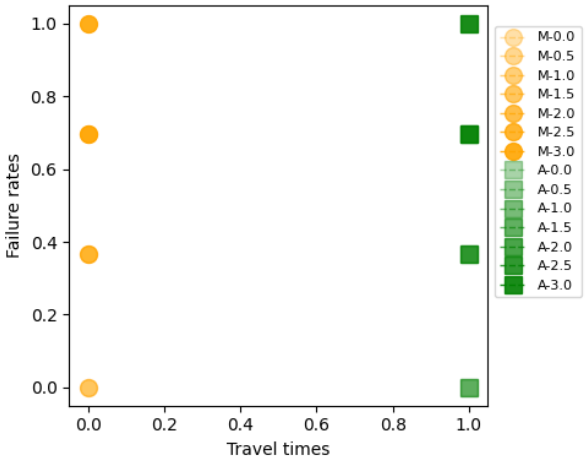
(c) *a-Instance3*



(d) *a-Instance4*



(e) *a-Instance5*

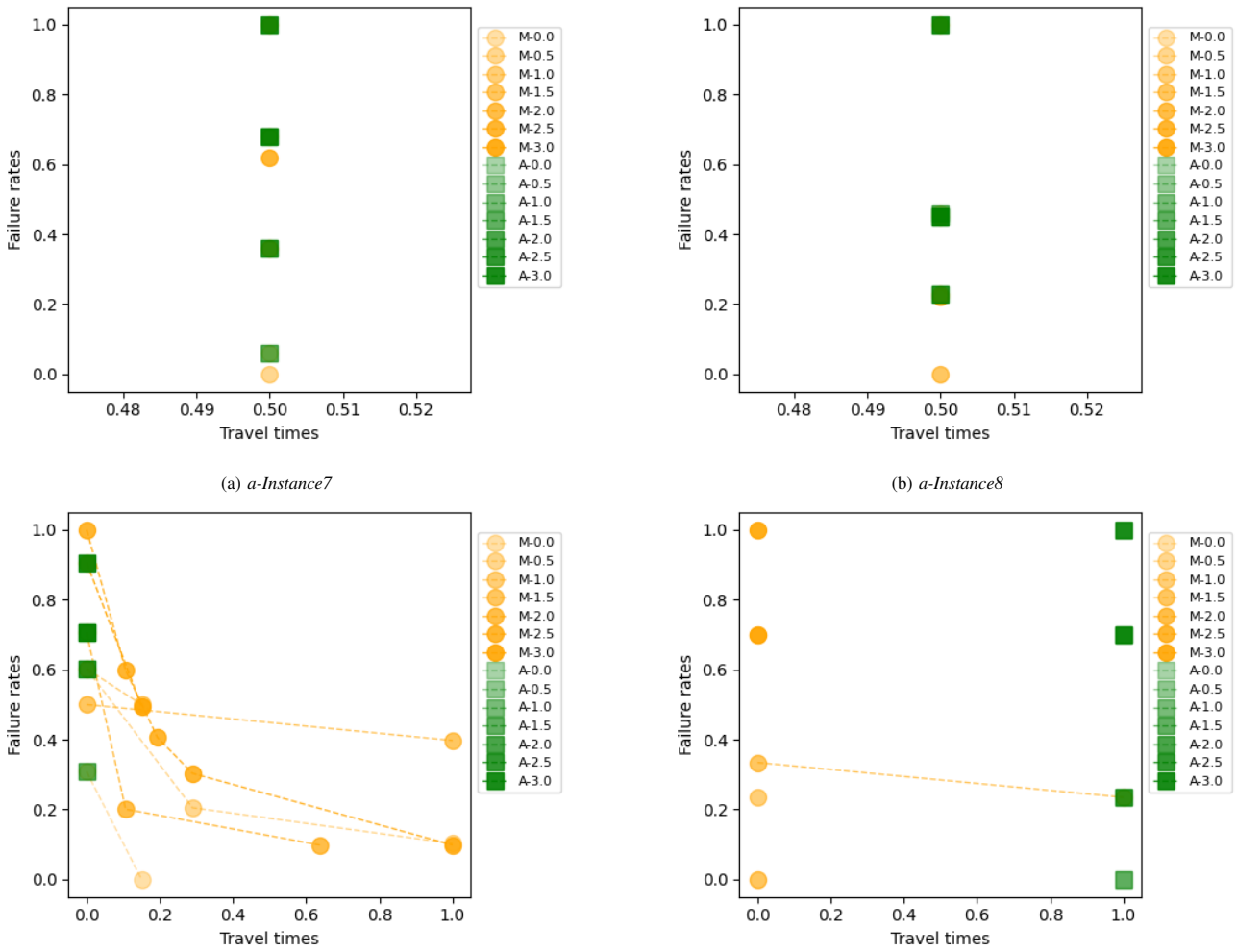


(f) *a-Instance6*

Fig. 5: Pareto fronts: *a-Instance1 to a-Instance6*

the mathematical model and the TDBOARR approach to find the solutions of each problem instance.

With respect to runtimes, TDBOARR is vastly superior as shown in the table II in which the new algorithm surpass the solver in all the testcases by a high margin. This is also shown in larger test cases in table III where the difference between



(a) *a-Instance7*

(b) *a-Instance8*

(c) *a-Instance9*

(d) *a-Instance10*

Fig. 6: Pareto fronts: *a-Instance7 to a-Instance10*

TABLE II: Execution times of *a* instances in seconds.

Instance	Gurobi	TDBOARR
a-Instance1	7.4	0.005
a-Instance2	6.6	0.006
a-Instance3	6.4	0.011
a-Instance4	8.4	0.010
a-Instance5	8.4	0.011
a-Instance6	8.3	0.005
a-Instance7	9.7	0.004
a-Instance8	9.4	0.004
a-Instance9	11.1	0.004
a-Instance10	8.9	0.002

TABLE III: Execution times of *b* instances in seconds.

Instance	Gurobi	TDBOARR
b-Instance1	22671.1	0.569
b-Instance2	22133.4	0.12

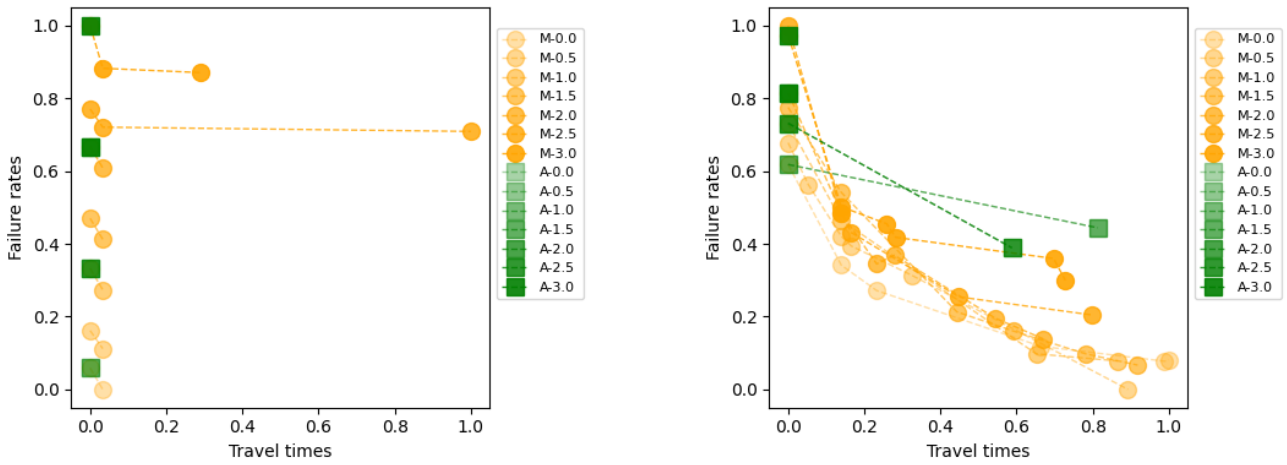
the two is even bigger as TDBOARR needs less than a second while the Gurobi solver needs around to twenty thousands.

Fig. 7 shows the Pareto Fronts obtained by the solver applied to the mathematical model and the TDBOARR approach proposed when solving the type b problem instances.

Finally, for the larger instances in Fig. 7. In instance *b-Instance1*, it can be seen that the solver does not find the full frontier at none of the start times, just like TDBOARR, and there is a small offset between the solutions, as previously commented. For the other case of instance *b-Instance2*, the Gurobi solver is able to find more intermediate solutions on the Pareto frontier versus TDBOARR, this because TDBOARR prunes these solutions to optimize the algorithm to be able to find both ends of the frontier more easily. This pruning can cause problems in finding other solutions, behavior that TDBOARR has demonstrated throughout the experiments.

VII. Conclusions and Future Work

We present TDBOARR as an alternative algorithm for time-domain bi-objective routing problems that uses the speed and simplicity of search techniques like BOA*. Our experimental



(a) *b-Instance1*

(b) *b-Instance2*

Fig. 7: Pareto fronts: *b-Instance1 and b-Instance2*

evaluation, uses small real-world scenarios in the Santiago de Chile street grid. We compared our proposed solution with a classical approach like ILP solvers. From this, we can observe that TDBOARR encounters some difficulties at the moment of finding the full Pareto frontier in time-domain objectives, while ILPs are a more robust but slower solution, being TDBOARR order of magnitude faster. These put TDBOARR as an alternative for this problems as an non-optimal but fast approach to solve time-domain bi-objective problems. Anyway it is important to have in mind that large problem instances should be tested in order to evaluate the real contribution of the approach to the real world problem.

As future work, we plan to extend TDBOARR with a more focused and proven time-domain heuristic that could modify the behavior that our present implementation shows and complete more instances. To test the limits of TDBOARR with larger problem instances to analyze if the fast performance of the algorithm grows with the size of the instances.

Aknowledgments

This work was funded by FONDECYT project 1230365.

References

[1] M. Ehrgott, "Approximation algorithms for combinatorial multicriteria optimization problems," *International Transactions in Operational Research*, vol. 7, no. 1, pp. 5–31, 2000.

[2] C. H. Papadimitriou, "The euclidean travelling salesman problem is np-complete," *Theoretical Computer Science*, vol. 4, no. 3, pp. 237 – 244, 1977.

[3] Y. P. Aneja and K. P. K. Nair, "Bicriteria transportation problem," *Management Science*, vol. 25, no. 1, pp. 73–78, 1979.

[4] D. Feillet, P. Dejax, and M. Gendreau, "Traveling salesman problems with profits," *Transportation science*, vol. 39, no. 2, pp. 188–205, 2005.

[5] K. Ilavarasi and K. S. Joseph, "Variants of travelling salesman problem: A survey," *International Conference on Information Communication and Embedded Systems (ICICES2014)*, pp. 1–7, 2014.

[6] C. C. Ribeiro and P. Hansen, *Essays and surveys in metaheuristics*. Kluwer Academic Publishers, 2002.

[7] Z. Yan, L. Zhang, L. Kang, and G. Lin, "A new moea for multi-objective tsp and its convergence property analysis," in *Evolutionary Multi-Criterion Optimization*, C. M. Fonseca, P. J. Fleming, E. Zitzler, L. Thiele, and K. Deb, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 342–354.

[8] Z. Zhang, C. Gao, Y. Lu, Y. Liu, and M. Liang, "Multi-objective ant colony optimization based on the physarum-inspired mathematical model for bi-objective traveling salesman problems," *PloS one*, vol. 11, no. 1, p. e0146709, 2016.

[9] I. Khan, M. K. Maiti, and K. Basuli, "Multi-objective traveling salesman problem: an abc approach," *Applied Intelligence*, vol. 50, no. 11, pp. 3942–3960, 2020.

[10] T. K. Ralphs, M. J. Saltzman, and M. M. Wiecek, "An improved algorithm for solving biobjective integer programs," *Annals of Operations Research*, vol. 147, no. 1, pp. 43–70, 2006.

[11] B. Bixby, "The gurobi optimizer," *Transp. Re-search Part B*, vol. 41, no. 2, pp. 159–178, 2007.

[12] M. R. Bussieck and S. Vigerske, "Minlp solver software," *Wiley encyclopedia of operations research and management science*, 2010.

[13] H. B. Ticha, N. Absi, D. Feillet, and A. Quilliot, "The steiner bi-objective shortest path problem," *EURO Journal on Computational Optimization*, p. 100004, 2021.

[14] C. H. Ulloa, W. Yeoh, J. A. Baier, H. Zhang, L. Suazo, and S. Koenig, "A simple and fast bi-objective search algorithm," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, 2020, pp. 143–151.

[15] B. S. Stewart and C. C. White, "Multiobjective a*," *J. ACM*, vol. 38, no. 4, p. 775–814, Oct. 1991.

[16] L. Mandow, J. P. De la Cruz *et al.*, "A new approach to multiobjective a* search," in *IJCAI*, vol. 8. Citeseer, 2005.

[17] A. Sedeño-noda and M. Colebrook, "A biobjective dijkstra algorithm," *European Journal of Operational Research*, vol. 276, no. 1, pp. 106–118, 2019.

[18] S. Stefan and S. Edelkamp, *Heuristic Search: Theory and Applications*. Morgan Kaufmann Publishers, 2012.

[19] I. Pohl, "The avoidance of (relative) catastrophe, heuristic competence, genuine dynamic weighting and computational issues in heuristic problem solving," in *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, ser. IJCAI'73. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1973, p. 12–17.

[20] F. Menares, E. Montero, G. Paredes-Belmar, and A. Bronfman, "A bi-objective time-dependent vehicle routing problem with delivery failure probabilities," *Computers Industrial Engineering*, vol. 185, p. 109601, 2023.