# Optimal Solutions for the Moving Target Vehicle Routing Problem via Branch-and-Price with Relaxed Continuity

**Anoop Bhat**[1], **Geordan Gutow**[2], **Zhongqiang Ren**[3] **Sivakumar Rathinam**[4] **Howie Choset**[1]
[1]Robotics Institute at Carnegie Mellon University, Pittsburgh, PA 15213
[2]Mechanical and Aerospace Engineering at Michigan Technological University, Houghton, MI 49931
[3]UM-SJTU Joint Institute and Department of Automation at Shanghai Jiao Tong University, Shanghai, China
[4]Department of Mechanical Engineering and Department of Computer Science and Engineering at Texas A&M University, College Station, TX 77843
agbhat@andrew.cmu.edu, gmgutow@mtu.edu, zhongqiang.ren@sjtu.edu.cn, srathinam@tamu.edu, choset@andrew.cmu.edu.

## Abstract

The Moving Target Vehicle Routing Problem (MT-VRP) seeks trajectories for several agents that intercept a set of moving targets, subject to speed, time window, and capacity constraints. We introduce an exact algorithm, Branch-and-Price with Relaxed Continuity (BPRC), for the MT-VRP. The main challenge in a branch-and-price approach for the MT-VRP is the pricing subproblem, which is complicated by moving targets and time-dependent travel costs between targets. Our key contribution is a new labeling algorithm that solves this subproblem by means of a novel dominance criterion tailored for problems with moving targets. Numerical results on instances with up to 25 targets show that our algorithm finds optimal solutions more than an order of magnitude faster than a baseline based on previous work, showing particular strength in scenarios with limited agent capacities.

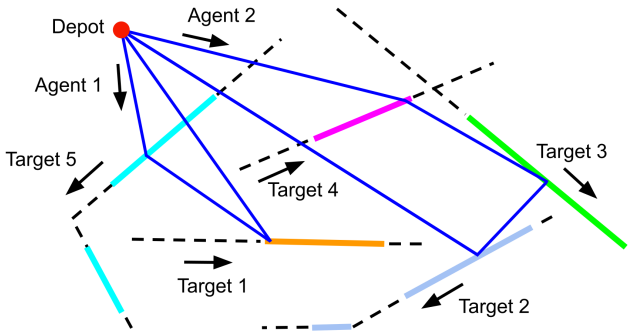


Figure 1: Targets move along piecewise-linear trajectories with time windows shown in bold lines. Two agents begin at the depot and collectively intercept all targets. Trajectories of agents are shown in blue.

## 1 Introduction

The Vehicle Routing Problem (VRP) is one of the most well-studied problems in logistics, with numerous applications (Toth and Vigo 2014; Archetti et al. 2025). The standard VRP considers a set of targets, each with a demand for goods, and a fleet of agents (often called vehicles), each with a capacity limit on the amount of goods it can deliver. Given the travel costs between every pair of targets, as well as between each target and the depot (the starting location of the agents), the VRP seeks a sequence of targets for each agent such that each target is visited by exactly one agent, the total demand serviced by each agent does not exceed its capacity, and the sum of the travel costs of the agents is minimized.

In this article, we consider a generalization of the VRP in which the targets are moving, and each target has one or more time windows in which it can be visited (Fig. 1). We call this generalization the Moving Target VRP (MT-VRP). We are motivated to solve this generalization due to its applications, including defense (Helvig, Robins, and Zelikovsky 2003; Smith 2021; Stieber and Fügenschuh 2022), shipping supplies to ships at sea (Brown et al. 2017), aerial fueling (Barnes et al. 2004), and recharging underwater vehicles monitoring the seafloor (Li et al. 2019).



The MT-VRP generalizes the Traveling Salesman Problem (TSP) and is therefore NP-hard (Hammar and Nilsson 1999; Helvig, Robins, and Zelikovsky 2003). Although previous work has investigated simplified versions of the MT-VRP without capacity constraints (Philip et al. 2025b; Stieber and Fügenschuh 2022), the capacitated version has not been considered and is the focus of this paper. Since state-of-the-art exact VRP solvers are based on the branch-and-price method (Costa, Contardo, and Desaulniers 2019), we present a branch-and-price approach for the MT-VRP.

A key aspect of the branch-and-price method involves alternately solving a *master problem* and a *pricing problem.* The master problem seeks to assign each agent a single sequence of target-time window pairings from a subset of all possible sequences. We refer to a target-window pairing as a *target-window*. The pricing problem attempts to add sequences to the subset of target-window sequences considered in the master problem. In particular, the pricing problem seeks one or more sequences with negative *reduced cost*, which is the cost of the sequence (i.e., the travel distance required to visit its targets), plus additional bias terms, discussed in Section 5.1. Solving the pricing problem efficiently requires a method for identifying when a sequence $\Gamma$ *dominates* another sequence $\Gamma'$ ending at the same target-window, meaning no extension of $\Gamma'$ can have a lower (i.e.

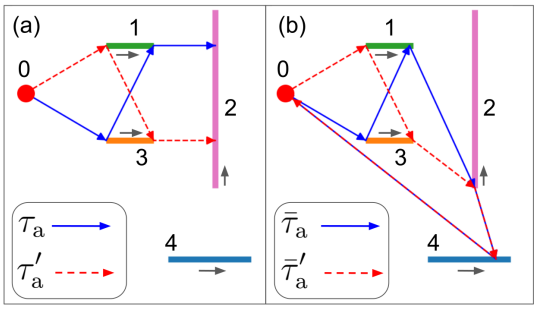


Figure 2: Example where existing VRP dominance checks fail for the MT-VRP. Since each target has one time window in this example, we represent a target-window using the index of the target. We treat the depot as a fictitious target 0. (a) $\tau_a$ is an optimal (i.e. minimum-distance) trajectory visiting the sequence of targets $\Gamma = (0, 3, 1, 2)$, and $\tau_a{}'$ is an optimal trajectory visiting the sequence $\Gamma' = (0, 1, 3, 2)$.[1] Conventional VRP dominance checks would conclude that $\Gamma$ dominates $\Gamma'$, since $\tau_a$ and $\tau_a{}'$ have the same cost.[2] (b) $\bar{\tau}_a$ optimally visits $\bar{\Gamma} = (0, 3, 1, 2, 4, 0)$ obtained by appending 4 and 0 to $\Gamma$, and $\bar{\tau}_a'$ optimally visits $\bar{\Gamma}' = (0, 1, 3, 2, 4, 0)$, obtained by appending 4 and 0 to $\Gamma'$. $\bar{\tau}_a$ travels more distance than $\bar{\tau}_a'$, so $\bar{\Gamma}$ is worse than $\bar{\Gamma}'$. Thus $\Gamma$ does not dominate $\Gamma'$.

more negative) reduced cost than the same extension of $\Gamma$.

Typical VRP dominance checks assume that when we append target-windows to $\Gamma$, the cost incurred by visiting these target-windows only depends on the final target-window in $\Gamma$, and that the same goes for $\Gamma'$. Under this assumption, appending the same sequence of target-windows to $\Gamma$ and $\Gamma'$ would result in the same increase in cost. However, as shown in Fig. 2, this assumption breaks when we have moving targets. In this case, appending the same sequence of target-windows to $\Gamma$ and $\Gamma'$ actually increases the cost of $\Gamma$ more than for $\Gamma'$. This is because the cost of the sequence obtained by appending target-windows to $\Gamma$ is computed using a continuous trajectory optimization problem that depends on the entire sequence of target-windows in $\Gamma$, not just its final target-window.

To address this challenge, we introduce a new labeling algorithm with a dominance check for the MT-VRP that compares an upper bound on the cost of $\Gamma$ and a lower-bound on the cost of $\Gamma'$, which defers the expense of solving a continuous optimization problem until necessary. The upper bound is efficiently calculated by constructing a feasible trajectory using sampled points from the target-windows, whereas the lower bound is computed via a trajectory planning problem with relaxed continuity constraints, inspired by (Philip et al. 2025a). We refer to our approach as Branch-and-Price with Relaxed Continuity (BPRC). We present numerical results that demonstrate in problems where agents have small capacities, BPRC achieves more than an order of magnitude speedup over a baseline method, which incorporates capacity constraints from (Desrochers et al. 1987) into a state-of-the-art multi-agent MT-TSP solver (Philip et al. 2025b). We also present ablation studies to further validate our approach.

## 2 Related Work

**VRP:** Approaches for solving VRP variants have utilized two main types of integer programming formulations: *compact* and *extended*. In compact formulations, the number of variables and constraints is polynomial in the number of targets (Desrochers et al. 1987; Bard, Kontoravdis, and Yu 2002). In contrast, extended formulations have a number of variables and constraints that is exponential (Ozbaygin et al. 2017; Desrochers, Desrosiers, and Solomon 1992; Costa, Contardo, and Desaulniers 2019). Although compact formulations are smaller optimization problems, state-of-the-art VRP solvers, based on branch-and-price, leverage extended formulations. This is because the extended formulations tend to have tighter relaxations, i.e., the optimal cost of an extended formulation's convex relaxation tends to be closer to the optimal cost of the non-relaxed problem. Our work employs an extended formulation of the MT-VRP, and our numerical results in Section 6 show that the extended formulation tends to be tighter than a compact one.

**Multi-Agent MT-TSP:** Prior work (Philip et al. 2025b) finds optimal solutions for the Multi-Agent MT-TSP without capacity constraints using a mixed-integer conic program (MICP), assuming that targets move along piecewise-linear trajectories. We make the same piecewise-linear assumption in our work. Our baseline is based on (Philip et al. 2025b). Note that, using the terminology from the previous section, the MICP from (Philip et al. 2025b) is compact.

**VRP with Floating Targets:** The VRP with Floating Targets (VRPFT) (Gambella, Naoum-Sawaya, and Ghaddar 2018) is a problem similar to the MT-VRP, but where the movements of the targets must also be planned, and the targets do not have time windows. (Gambella, Naoum-Sawaya, and Ghaddar 2018) applies branch-and-price to the VRPFT, though the pricing problem is solved using an off-the-shelf mixed-integer program solver. In other branch-and-price methods, it is common to instead use a labeling algorithm for the pricing problem (Costa, Contardo, and Desaulniers 2019). In contrast to (Gambella, Naoum-Sawaya, and Ghaddar 2018), our work uses a labeling algorithm, introducing a new dominance criterion for moving targets.

## 3 Problem Setup

We consider $n_{\text{agt}}$ agents and $n_{\text{tar}}$ targets moving in $\mathbb{R}^2$. All agents start at the same depot, with position $q_0 \in \mathbb{R}^2$. They also have an identical speed limit $v_{\max} \in \mathbb{R}_{\geq 0}$, as well as an identical capacity $d_{\max} \in \mathbb{R}_{\geq 0}$.

Each target $i$ has a demand $d_i \in \mathbb{R}_{\geq 0}$ that must be met by exactly one agent. It can be visited by any agent during one of its time windows in $\{[\underline{t}_{i,1}, \bar{t}_{i,1}], \ldots, [\underline{t}_{i,n_{\text{win}}(i)}, \bar{t}_{i,n_{\text{win}}(i)}]\}$, where $[\underline{t}_{i,j}, \bar{t}_{i,j}] \subseteq \mathbb{R}_{\geq 0}$ is the $j$th time window of target

[1] Agents in the MT-VRP can change their speed, which is why $\tau_a$ and $\tau_a{}'$ can travel the same distance, even though $\tau_a$ meets target 2 later than $\tau_a{}'$: $\tau_a$ can simply arrive at its interception position, then stop and wait for target 2.

[2] While dominance checks technically compare *reduced cost*, which adds bias terms to the cost, the bias terms for $\Gamma$ and $\Gamma'$ are equal in Fig. 2 (a), and the same goes for $\bar{\Gamma}$ and $\bar{\Gamma}'$ in Fig. 2 (b).

$i$, and $n_{\text{win}}(i)$ is the number of time windows of target $i$. Each target $i$ moves along a trajectory $\tau_i : \bigcup_{j=1}^{n_{\text{win}}(i)} [\underline{t}_{i,j}, \bar{t}_{i,j}] \to \mathbb{R}^2$ and has a constant velocity within a time window, but possibly different velocities in different time windows. We assume no target at any time moves faster than $v_{\max}$.

An agent's trajectory is *speed-admissible* if the agent's speed at all times along its trajectory satisfies the speed limit. The cost of an agent trajectory $\tau_{\text{a}}$, denoted as $c(\tau_{\text{a}})$, is its distance traveled, if $\tau_{\text{a}}$ is speed-admissible, and $\infty$ otherwise. An agent's trajectory $\tau_{\text{a}}$ *intercepts* target $i$ if there exists a time $t$ in one of target $i$'s time windows such that (1) $\tau_{\text{a}}(t) = \tau_i(t)$, and (2) $\tau_{\text{a}}$ *claims* target $i$ at time $t$. The notion of "claiming" is needed in scenarios where we plan a trajectory $\tau_{\text{a}}$ with the intent of intercepting target $i$, then $j$, but $\tau_{\text{a}}$ unintentionally satisfies $\tau_{\text{a}}(t) = \tau_k(t)$ for some target $k$ between the interceptions of $i$ and $j$.

The MT-VRP seeks a speed-admissible trajectory for each agent such that (i) each agent's trajectory begins and ends at the depot, (ii) every target is intercepted by exactly one agent, (iii) the sum of the demands of targets intercepted by each agent does not exceed $d_{\max}$, and (iv) the sum of the agents' trajectory costs is minimized.

## 4 Integer Linear Program (ILP) for MT-VRP

We formulate the MT-VRP on a graph $\mathcal{G}_{\text{tw}} = (\mathcal{V}_{\text{tw}}, \mathcal{E}_{\text{tw}})$ referred to as the *target-window-graph*. Each node in $\mathcal{V}_{\text{tw}}$ is a target paired with one of its time windows. We call such a pairing a *target-window*. The *target-windows* for a target $i \in \mathcal{V}_{\text{tw}}$ are represented as $\gamma_{i,j} = (i, [\underline{t}_{i,j}, \bar{t}_{i,j}])$ for $j \in \{1, \ldots, n_{\text{win}}(i)\}$. To simplify notation, we refer to the depot as a fictitious target 0 with demand $d_0 = 0$ and trajectory $\tau_0$ satisfying $\tau_0(t) = q_0$ for all $t \geq 0$. We also define a corresponding target-window $\gamma_{0,1} = \gamma_0 = (0, [0, \infty))$. An agent trajectory $\tau_{\text{a}}$ *intercepts* target-window $\gamma_{i,j}$ if $\tau_{\text{a}}$ intercepts target $i$ at a time $t \in [\underline{t}_{i,j}, \bar{t}_{i,j}]$.

$\mathcal{E}_{\text{tw}}$ contains an edge from target-window $\gamma_{i,j}$ to $\gamma_{i',j'}$ if and only if $i \neq i'$. This edge records the *latest feasible departure time* from $\gamma_{i,j}$ to $\gamma_{i',j'}$, denoted by $\text{LFDT}(\gamma_{i,j}, \gamma_{i',j'})$. This value specifies the maximum $t \in [\underline{t}_{i,j}, \bar{t}_{i,j}]$ for which there exists a speed-admissible trajectory that intercepts $\gamma_{i',j'}$ after intercepting $\gamma_{i,j}$ at time $t$.[3]

A *tour* in $\mathcal{G}_{\text{tw}}$ is a path beginning and ending with $\gamma_0$, visiting at most one target-window per non-fictitious target, such that the sum of demands of visited targets is no larger than $d_{\max}$. A *partial tour* has the same definition as a tour, but does not need to end with $\gamma_0$. $\Gamma[k]$ denotes the $k$th element of a partial tour $\Gamma$, and $\text{Len}(\Gamma)$ denotes the number of elements; we use the same notation for any *sequence* also. An agent trajectory $\tau_{\text{a}}$ *executes* a partial tour $\Gamma$ if $\tau_{\text{a}}$ intercepts the target-windows in $\Gamma$ in sequence. The optimal cost of a partial tour $\Gamma$, denoted as $c^*(\Gamma)$, is the minimum cost over all trajectories executing $\Gamma$. $c^*(\Gamma)$, and the associated trajectory, can be computed using the second-order cone program (SOCP) in Appendix A in the full paper (Bhat et al. 2026).[4]

[3](Philip et al. 2025a) describes how to compute LFDT.

[4]This SOCP can be obtained by modifying the mixed-integer SOCP from (Philip et al. 2025b), which considers the case where the sequence of target-windows is not fixed.

We now formulate an ILP where a solution to the MT-VRP is described by a set of tours, $\mathcal{F}_{\text{sol}}$. For a tour $\Gamma$, let $\alpha(i, \Gamma) = 1$ if $\Gamma$ contains a target-window $\gamma_{i,j}$ of target $i$ and $\alpha(i, \Gamma) = 0$ otherwise. Let the set of all possible tours for the agents be $\mathcal{S} = \{\Gamma_1, \Gamma_2, \ldots, \Gamma_{|\mathcal{S}|}\}$. For each $\Gamma_k \in \mathcal{S}$, define a binary variable $\theta_k$, which equals 1 if $\Gamma_k$ is chosen to be included in $\mathcal{F}_{\text{sol}}$ and 0 otherwise. Our ILP for the MT-VRP is as follows:

$$\min_{\{\theta_k\}_{\Gamma_k \in \mathcal{S}}} \quad \sum_{\Gamma_k \in \mathcal{S}} c^*(\Gamma_k)\theta_k \tag{1a}$$

$$\text{s.t.} \quad \sum_{\Gamma_k \in \mathcal{S}} \theta_k \leq n_{\text{agt}} \tag{1b}$$

$$\sum_{\Gamma_k \in \mathcal{S}} \alpha(i, \Gamma_k)\theta_k \geq 1 \ \ \forall i \in \{1, \ldots, n_{\text{tar}}\} \tag{1c}$$

$$\theta_k \in \{0, 1\} \ \ \forall k \in \{1, \ldots, |\mathcal{S}|\} \tag{1d}$$

(1a) minimizes the sum of costs of the selected tours. (1b) ensures that we select up to $n_{\text{agt}}$ tours. (1c) ensures that each target is visited by at least one tour. (1d) constrains $\theta_k$ to be binary. Capacity constraints are implicit in ILP (1), since a tour must fully satisfy the demands of all targets it visits.

A convex relaxation of ILP (1) consists of (1a)-(1c), but replaces (1d) with the constraint $\theta_k \geq 0$; as in (Feillet 2010), we remove the upper limit on $\theta_k$ in the relaxation. An optimal solution $\theta^* = (\theta_1^*, \theta_2^*, \ldots, \theta_{|\mathcal{S}|}^*)$ to ILP (1) is feasible for this relaxation, but not necessarily optimal: some fractional solution may be optimal instead. Thus, the branch-and-bound in Section 5 iteratively breaks ILP (1) into subproblems, each disallowing some set of edges $\mathcal{B}$ to eliminate fractional solutions. The aim is to generate some subproblem where $\theta^*$ is optimal for the relaxation. In particular, for a set $\mathcal{B} \subseteq \mathcal{E}_{\text{tw}}$, we define the subproblem ILP-$\mathcal{B}$ as ILP (1), with the constraint that $\theta_k = 0$ for each $\Gamma_k$ traversing some $e \in \mathcal{B}$. Let LP-$\mathcal{B}$ be the convex relaxation of ILP-$\mathcal{B}$.

## 5 Branch-and-Price with Relaxed Continuity

We first provide an overview of our approach and then focus on the pricing problem. At the start of BPRC, we generate an initial feasible solution $\mathcal{F}_{\text{inc}}$ with cost $c_{\text{inc}}$, using the method from Section 5.3. We call $\mathcal{F}_{\text{inc}}$ the *incumbent* and continually update it to be the best solution found so far. We also initialize a set of tours $\mathcal{F}$, setting it equal to $\mathcal{F}_{\text{inc}}$; we continually add tours to $\mathcal{F}$ throughout the algorithm.

BPRC then constructs a branch-and-bound tree, where a node is a set $\mathcal{B} \subseteq \mathcal{E}_{\text{tw}}$ of disallowed edges. We expand nodes from a stack in a depth-first fashion. When expanding a node $\mathcal{B}$, we solve LP-$\mathcal{B}$ (traditionally called the *master problem* in branch-and-price) using column generation. That is, when we initially solve LP-$\mathcal{B}$, we only optimize over $\theta_k$ variables for tours $\Gamma_k \in \mathcal{F}$, traversing no edge in $\mathcal{B}$, to keep the problem size tractable, assuming all other $\theta_k$ values are zero. This restriction of LP-$\mathcal{B}$ is known as the *restricted master problem* (RMP). After solving the RMP using an off-the-shelf LP solver, we solve a *pricing problem* to add tours to

$\mathcal{F}$ that may improve the optimal cost of the RMP. Section 5.2 describes our method of solving the pricing problem. We alternate between the RMP and the pricing problem until the pricing problem adds no tours to $\mathcal{F}$, implying we have an optimal solution to LP-$\mathcal{B}$.

Whenever we find a new best integer solution during column generation, we update $\mathcal{F}_{\text{inc}}$ and $c_{\text{inc}}$. Additionally, the first time we solve the RMP for branch-and-bound node $\mathcal{B}$, the LP solver may return infeasible. This means that every feasible MT-VRP solution that can be assembled from the tours in $\mathcal{F}$ traverses some edge in $\mathcal{B}$. If so, we attempt to use the method from Section 5.3 to produce a feasible MT-VRP solution $\mathcal{F}_{\text{new}}$ traversing no edge in $\mathcal{B}$, set $\mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{F}_{\text{new}}$, then solve the RMP again. If we do not find any $\mathcal{F}_{\text{new}}$, LP-$\mathcal{B}$ is infeasible, and we discard the branch-and-bound node $\mathcal{B}$.

When we find an optimal solution $\theta$ to LP-$\mathcal{B}$, if $\theta$ is integer, $\mathcal{B}$ has no successors. If $\theta$ is fractional, we apply the "conventional branching" rule from (Ozbaygin et al. 2017) as follows. First, we let the flow of an edge $e$ be the sum of $\theta_k$ values over all $\Gamma_k$ traversing $e$. We select an edge $e$ not incident to $\gamma_0$ with minimum flow, then generate two successors to $\mathcal{B}$, where in the first successor, $e$ is disallowed, and in the second successor, $e$ is mandated by disallowing other edges appropriately (see (Ozbaygin et al. 2017)). This successor generation procedure ensures that an optimal integer solution to ILP-$\mathcal{B}$ is feasible for one successor, but the fractional solution $\theta$ is feasible in neither. Each successor $\mathcal{B}'$ is given a lower bound equal to the cost of $\theta$ for LP-$\mathcal{B}$. $\mathcal{B}'$ is pruned upon expansion if its lower bound is larger than $c_{\text{inc}}$.

## 5.1 Pricing Problem

The pricing problem seeks a set of tours $\mathcal{F}_{\text{price}} = \{{}^1\Gamma, {}^2\Gamma, \ldots, {}^{n_{\text{found}}}\Gamma\}$ such that the RMP on $\mathcal{F} \cup \mathcal{F}_{\text{price}}$ has a smaller optimal cost than the RMP on $\mathcal{F}$. $n_{\text{found}}$ is the number of tours found when solving the pricing problem and is not a user-specified parameter. We find $\mathcal{F}_{\text{price}}$ by leveraging the optimal dual solution to the RMP, denoted as $(\lambda_0, \lambda_1, \ldots, \lambda_{n_{\text{tar}}})$, where $\lambda_0 \in \mathbb{R}_{\leq 0}$ is the dual variable corresponding to the constraint (1b), and $\lambda_i \in \mathbb{R}_{\geq 0}$ is the dual variable corresponding to the $i^{th}$ constraint in (1c). Following (Feillet 2010), $\mathcal{F}_{\text{price}}$ can reduce the optimal cost of the RMP only if $\mathcal{F}_{\text{price}}$ contains a tour $\Gamma$ with negative *reduced cost*, which is defined as follows:

$$c^*_{\text{red}}(\Gamma) = c^*(\Gamma) - \sum_{i=1}^{n_{\text{tar}}} \alpha(i, \Gamma)\lambda_i - \lambda_0. \tag{2}$$

Thus the pricing problem seeks tours with negative reduced cost.[5]

Note that since the dual variables $\lambda_i \geq 0$ are subtracted in the reduced cost (2), adding target-windows to a tour can decrease its reduced cost. This property is important in Section 5.2 for determining if a partial tour $\Gamma$ dominates another, $\Gamma'$. Specifically, our dominance criterion requires that any target-window appendable to $\Gamma'$ must also be appendable to $\Gamma$. This ensures that any decrease in reduced cost achievable by extending $\Gamma'$ is also achievable by extending $\Gamma$.

[5] A tour $\Gamma$ already in $\mathcal{F}$ must have $c^*_{\text{red}}(\Gamma) \geq 0$, because otherwise $\lambda_0, \lambda_1, \ldots, \lambda_{n_{\text{tar}}}$ would be dual infeasible for the RMP (Feillet 2010). However, in practice, due to floating-point arithmetic, we may compute a slightly negative $c^*_{\text{red}}(\Gamma)$. Thus, as in prior work (Kohl et al. 1999), we seek tours with $c^*_{\text{red}}(\Gamma) < -\epsilon$ to avoid repeatedly adding tours to $\mathcal{F}$; in our implementation, $\epsilon = 10^{-4}$.

## 5.2 BPRC Labeling Algorithm

We solve the pricing problem using a labeling algorithm based on (Boland, Dethridge, and Dumitrescu 2006), which forms the basis for state-of-the-art VRP pricing solvers. We first discuss the novel dominance criterion used in our labeling algorithm, then provide the algorithm itself.

For a partial tour $\Gamma$ to dominate another partial tour $\Gamma'$, two conditions must be met. First, any feasible extension of $\Gamma'$ (i.e., a sequence of target-windows that can be appended to it) must also be a feasible extension of $\Gamma$. Feasibility here means not violating capacity constraints or revisiting targets. Second, the minimum reduced cost over extensions of $\Gamma$ must be no greater than that from $\Gamma'$. *Our primary contribution is a novel and computationally efficient method for verifying this second condition for the MT-VRP.*

We first explain the core idea before providing formal definitions. Assume the feasibility condition is met and that both $\Gamma$ and $\Gamma'$ end at the same target-window, $\gamma_{i,j}$. For an arbitrary $t' \in [\underline{t}_{i,j}, \bar{t}_{i,j}]$, let ${\tau_{\text{a}}}'$ be an optimal trajectory that executes $\Gamma'$ and intercepts $\gamma_{i,j}$ at space-time point $(q', t')$, where $q' = \tau_i(t')$. Suppose there exists a trajectory $\tau_{\text{a}}$ that executes $\Gamma$ and intercepts $\gamma_{i,j}$ at its starting point $(q, t)$. Let $\lambda = \sum_{i=1}^{n_{\text{tar}}} \alpha(i, \Gamma)\lambda_i + \lambda_0$, and $\lambda'$ be defined similarly for $\Gamma'$. Clearly, $\Gamma$ dominates $\Gamma'$ if for all $t'$, there exists a corresponding $\tau_{\text{a}}$ satisfying

$$c(\tau_{\text{a}}) + \|q' - q\|_2 - \lambda \leq c({\tau_{\text{a}}}') - \lambda'. \tag{3}$$

This means that following $\tau_{\text{a}}$, then moving in a straight line from $q$ to $q'$, is no worse than following ${\tau_{\text{a}}}'$. Checking this condition for every $t'$ is intractable, and thus we derive a sufficient condition for (3) using easily computable bounds:

- An *upper bound* on $c(\tau_{\text{a}}) + \|q' - q\|_2$ is $c(\tau_{\text{a,ub}}) + \delta(\gamma_{i,j})$, where $\tau_{\text{a,ub}}$ is a trajectory that executes $\Gamma$ optimally under the constraint that $\tau_{\text{a,ub}}$ can only intercept target-windows at their start points, and $\delta(\gamma_{i,j})$ is the spatial length of $\gamma_{i,j}$. We call $\tau_{\text{a,ub}}$ an *upper-bounding trajectory* for $\Gamma$. An example is shown in Fig. 3 (a) and (c).
- A *lower bound* on $c({\tau_{\text{a}}}')$ is the cost of a trajectory $\tau'_{\text{a,lb}}$ that optimally executes $\Gamma'$, where $\tau'_{\text{a,lb}}$ is allowed to be discontinuous between its arrival and departure at any target-window. We call $\tau'_{\text{a,lb}}$ a *lower-bounding trajectory* for $\Gamma'$. An example is shown in Fig. 3 (b) and (c).

This yields the following sufficient condition for (3):

$$c(\tau_{\text{a,ub}}) + \delta(\gamma_{i,j}) - \lambda \leq c(\tau'_{\text{a,lb}}) - \lambda'. \tag{4}$$

We constrain $\tau_{\text{a,ub}}$ to intercept target-windows at their start-points because this enables efficient computation of $c(\tau_{\text{a,ub}})$, as described in the subsequent section on the labeling algorithm. Otherwise, computing $c(\tau_{\text{a,ub}})$ would require solving an SOCP; we show in Section 6 that doing so for the vast

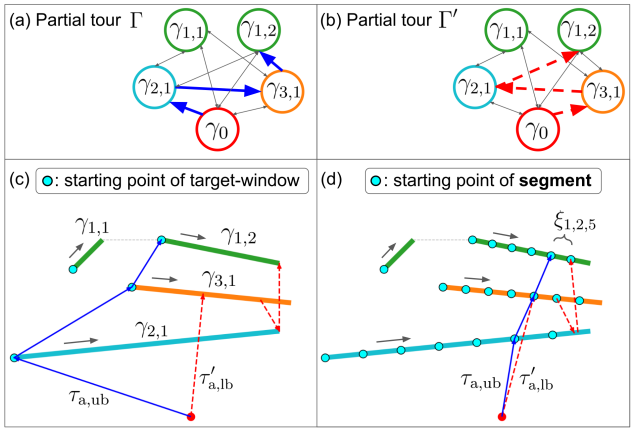


Figure 3: (a) and (b) show two partial tours $\Gamma$ and $\Gamma'$ ending with $\gamma_{1,2}$. We want to check if $\Gamma$ dominates $\Gamma'$. (c) We check dominance by comparing an upper-bounding trajectory $\tau_{\text{a,ub}}$ for $\Gamma$ and a lower-bounding trajectory $\tau'_{\text{a,lb}}$ for $\Gamma'$, as described in Section 5.2. (d) We strengthen our dominance check by dividing $\gamma_{1,2}$ into segments and applying the check from (c) for each segment; an example is shown for the 5th segment of $\gamma_{1,2}$, denoted as $\xi_{1,2,5}$.

number of partial tours in the pricing problem is computationally prohibitive. We relax continuity when computing $\tau'_{\text{a,lb}}$ similarly to avoid solving an SOCP.

Next, note that condition (4) can be weak if the time window of $\gamma_{i,j}$ is large, thus limiting its ability to prune partial tours. To strengthen (4), we divide each target-window into segments and apply (4) to each segment, as shown in Fig. 3 (d). The new labels in BPRC are designed to store this extra information–namely, the upper and lower cost bounds and the accumulated dual variables for each partial tour. We now formally define the labels and the algorithm.

**BPRC Labels:** We divide each target-window $\gamma_{i,j}$ into *segments* $\xi_{i,j,1}, \ldots, \xi_{i,j,n_{\text{seg}}(\gamma_{i,j})}$, where $\xi_{i,j,k} = (\gamma_{i,j}, [\underline{t}_{i,j,k}, \overline{t}_{i,j,k}])$ is the $k$th segment for $\gamma_{i,j}$, and $n_{\text{seg}}(\gamma_{i,j})$ is the number of segments for $\gamma_{i,j}$. The depot only gets one segment, i.e. $n_{\text{seg}}(\gamma_0) = 1$. For $\gamma_{i,j} \neq \gamma_0$, a user must first specify the number of segments per target, $n_{\text{seg,tar}}$. We then allocate segments to target-windows so that each target-window gets at least one segment, and target-windows with longer time windows get more segments. Appendix C (Bhat et al. 2026) gives the allocation formula.

Let $\delta(\xi_{i,j,k})$ be the spatial length of $\xi_{i,j,k}$, and let $s_{i,j,k} = (\tau_i(\underline{t}_{i,j,k}), \underline{t}_{i,j,k})$ be the starting point of $\xi_{i,j,k}$. We say an agent trajectory $\tau_{\text{a}}$ *intercepts* segment $\xi_{i,j,k}$ if $\tau_{\text{a}}$ intercepts target $i$ in the time window of $\xi_{i,j,k}$.

Within the labeling algorithm, we represent a partial tour $\Gamma$ as a *label*, denoted as $\text{Label}(\Gamma) = (\gamma_{i,j}, t, \sigma, \vec{b}, \vec{g}_{\text{ub}}, \vec{g}_{\text{lb}}, \lambda)$, whose elements we define below. Note that $\gamma_{i,j}, t, \sigma$, and $\vec{b}$ are elements typically stored in VRP labeling algorithms (Costa, Contardo, and Desaulniers 2019), while $\vec{g}_{\text{ub}}, \vec{g}_{\text{lb}}$ and $\lambda$ are specific to our method for the MT-VRP. In particular,

- $\gamma_{i,j}$ is the final target-window in $\Gamma$
- $t$ is the minimum time required to execute $\Gamma$
- $\sigma$ is the sum of demands of targets visited by $\Gamma$
- $\vec{b}$ is a binary vector with length $n_{\text{tar}}$, where $\vec{b}[i'] = 1$ if and only if $\Gamma$ visits target $i'$, $\sigma + d_{i'} > d_{\max}$, or $t > \max_{j' \in \{1,2,\ldots,n_{\text{win}}(i')\}} \text{LFDT}(\gamma_{i,j}, \gamma_{i',j'})$. The last two conditions mean target $i'$ cannot be intercepted after intercepting $\gamma_{i,j}$ at time $t$, due to capacity or speed constraints.[6]
- $\vec{g}_{\text{ub}}$ is a vector of length $n_{\text{seg}}(\gamma_{i,j})$, where $\vec{g}_{\text{ub}}[k]$ is the cost of an upper-bounding trajectory for $\Gamma$ intercepting $\xi_{i,j,k}$ at its start point.
- $\vec{g}_{\text{lb}}$ is a vector of length $n_{\text{seg}}(\gamma_{i,j})$, where $\vec{g}_{\text{lb}}[k]$ is the cost of a lower-bounding trajectory for $\Gamma$ intercepting $\xi_{i,j,k}$.
- $\lambda = \sum_{i=1}^{n_{\text{tar}}} \alpha(i, \Gamma)\lambda_i + \lambda_0$.

Each label $l$ also stores a backpointer to a parent label. By traversing backpointers recursively, we can obtain the partial tour $\Gamma$ corresponding to $l$, denoted as $\text{PartialTour}(l)$.

**BPRC Dominance Criteria:** Consider two labels $l = (\gamma_{i,j}, t, \sigma, \vec{b}, \vec{g}_{\text{ub}}, \vec{g}_{\text{lb}}, \lambda)$ and $l' = (\gamma_{i,j}, t', \sigma', \vec{b}', \vec{g}'_{\text{ub}}, \vec{g}'_{\text{lb}}, \lambda')$, both at target-window $\gamma_{i,j}$, and let $\Gamma = \text{PartialTour}(l)$ and $\Gamma' = \text{PartialTour}(l')$. We now define dominance so that if $l$ dominates $l'$, $\text{PartialTour}(l)$ dominates $\text{PartialTour}(l')$.

- General case when $\gamma_{i,j} \neq \gamma_0$: $l$ dominates $l'$ if all the following conditions hold:

$$\sigma \leq \sigma' \tag{5}$$

$$\vec{b}[i'] \leq \vec{b}'[i'], \ \ \forall i' \in \{1, \ldots, n_{\text{tar}}\} \tag{6}$$

$$\vec{g}_{\text{ub}}[k] + \delta(\xi_{i,j,k}) - \lambda \leq \vec{g}'_{\text{lb}}[k] - \lambda', \ \forall k \in \{1, \ldots, n_{\text{seg}}(\gamma_{i,j})\} \tag{7}$$

- Special case when $\gamma_{i,j} = \gamma_0$: $l$ dominates $l'$ if $c^*_{\text{red}}(\Gamma) \leq c^*_{\text{red}}(\Gamma')$.

(5)-(6) ensure that any target-windows appendable to $\Gamma$ are appendable to $\Gamma'$, and (7) is a generalization of condition (4).

**BPRC Labeling Algorithm:** The algorithm (Alg. 1) first initializes a priority queue, where a label $l = (\gamma, t, \sigma, \vec{b}, \vec{g}_{\text{ub}}, \vec{g}_{\text{lb}}, \lambda)$ has higher priority than label $l' = (\gamma', t', \sigma', \vec{b}', \vec{g}'_{\text{ub}}, \vec{g}'_{\text{lb}}, \lambda')$ if $(t, \sigma, \min(\vec{g}_{\text{lb}}) - \lambda)$ is lexicographically smaller than $(t', \sigma', \min(\vec{g}'_{\text{lb}}) - \lambda')$. The priority queue is initialized with the label whose partial tour contains only the depot. Alg. 1 then initializes a set $\mathfrak{D}[\gamma]$ of nondominated labels at each target-window $\gamma$ (Lines 2-3). Alg. 1's main loop performs a best-first search. Each iteration pops the highest-priority label $l = (\gamma_{i,j}, t, \sigma, \vec{b}, \vec{g}_{\text{ub}}, \vec{g}_{\text{lb}}, \lambda)$ from the priority queue, then iterates over the *successor target-windows* of $l$, i.e. each target-window $\gamma_{i',j'}$ satisfying

1. $(\gamma_{i,j}, \gamma_{i',j'}) \in \mathcal{E}_{\text{tw}} \setminus \mathcal{B}$, preventing traversal of edges in $\mathcal{B}$
2. $t \leq \text{LFDT}(\gamma_{i,j}, \gamma_{i',j'})$, so intercepting $\gamma_{i',j'}$ after intercepting $\gamma_{i,j}$ at time $t$ does not violate the speed limit
3. $\sigma + d_{i'} \leq d_{\max}$, enforcing capacity constraints

[6] Including both visitation and reachability status of targets in a label is a well-known strategy for strengthening dominance checks in VRP variants, introduced by (Feillet et al. 2004).

4. If $i' \neq 0$, then $\vec{b}[i'] = 0$, ensuring no targets are revisited

For each successor target-window $\gamma_{i',j'}$, we create a successor label $l' = (\gamma_{i',j'}, t', \sigma', \vec{b}', \vec{g}'_{\text{ub}}, \vec{g}'_{\text{lb}}, \lambda')$ as follows:

- $t'$ is the *earliest feasible arrival time* from space-time point $(\tau_i(t), t)$ to $\gamma_{i',j'}$, denoted as $\text{EFAT}(\tau_i(t), t, \gamma_{i',j'})$, which is the minimum time at which a speed-admissible trajectory can intercept $\gamma_{i',j'}$ after departing $(\tau_i(t), t)$. (Philip et al. 2025a) describes how to compute $t'$.
- $\sigma' = \sigma + d_{i'}$
- $\vec{b}'$ is identical to $\vec{b}$, except for the following modifications. First, if $i' \neq 0$, we set $\vec{b}'[i'] = 1$. Then, we set $\vec{b}'[i''] = 1$ for all targets $i''$ such that $\sigma' + d_{i''} > d_{\max}$ or $t' > \max_{j'' \in \{1,2,\ldots,n_{\text{win}}(i'')\}} \text{LFDT}(\gamma_{i',j'}, \gamma_{i''j''})$.
- $\lambda' = \lambda$, if $i' = 0$, and $\lambda' = \lambda + \lambda_{i'}$ otherwise
- For each $k' \in \{1, 2, \ldots, n_{\text{seg}}(\gamma_{i',j'})\}$, we compute $\vec{g}'_{\text{ub}}[k']$ via Bellman's optimality principle:

$$\vec{g}'_{\text{ub}}[k'] = \min_{k \in \{1,2,\ldots,n_{\text{seg}}(\gamma_{i,j})\}} \vec{g}_{\text{ub}}[k] + c_{\text{start}}(s_{i,j,k}, s_{i',j',k'}) \quad (8)$$

  where $c_{\text{start}}(s_{i,j,k}, s_{i',j',k'})$ is the Euclidean distance between the positions of $s_{i,j,k}$ and $s_{i',j',k'}$, if a speed-admissible trajectory exists from $s_{i,j,k}$ to $s_{i',j',k'}$, and $\infty$ otherwise. We compute $c_{\text{start}}$ for all pairs $(s_{i,j,k}, s_{i',j',k'})$ with $i \neq i'$ at the beginning of the BPRC algorithm.
- To compute each $\vec{g}'_{\text{lb}}[k']$, we first check if $t' > \bar{t}_{i',j',k'}$; if so, $\vec{g}'_{\text{lb}}[k'] = \infty$, since it is impossible for an agent to intercept $\xi_{i',j',k'}$ by executing PartialTour$(l')$. If $t' \leq \bar{t}_{i',j',k'}$, we apply Bellman's principle, similarly to (8):

$$\vec{g}'_{\text{lb}}[k'] = \min_{k \in \{1,2,\ldots,n_{\text{seg}}(\gamma_{i,j})\}} \vec{g}_{\text{lb}}[k] + c_{\text{seg}}(\xi_{i,j,k}, \xi_{i',j',k'}) \quad (9)$$

  where $c_{\text{seg}}(\xi_{i,j,k}, \xi_{i',j',k'})$ is the minimum distance an agent must travel between an interception of $\xi_{i,j,k}$ and an interception of $\xi_{i',j',k'}$, where $\xi_{i',j',k'}$ is intercepted after $\xi_{i,j,k}$. We compute $c_{\text{seg}}$ for all pairs $(\xi_{i,j,k}, \xi_{i',j',k'})$ with $i \neq i'$ at the beginning of the BPRC algorithm, using the SOCP in Appendix B (Bhat et al. 2026).

After generating the successor label $l'$, if $\gamma_{i',j'} = \gamma_0$ (i.e. $l'$ represents a tour $\Gamma'$), we perform additional pruning checks (Lines 9-10). First, if any label $l''$ at $\gamma_0$ satisfies $c^*_{\text{red}}(\text{PartialTour}(l'')) \leq \vec{g}'_{\text{lb}}[1] - \lambda'$, we prune $l'$. This is because $\vec{g}'_{\text{lb}}[1] - \lambda'$ is a lower bound on $c^*_{\text{red}}(\Gamma')$, and if this lower bound is not smaller than the reduced cost of a previously found tour, then $c^*_{\text{red}}(\Gamma')$ cannot be smaller either. Next, if $\vec{g}'_{\text{lb}}[1] - \lambda' \geq -\epsilon$, we prune $l'$, since this means $c^*_{\text{red}}(\Gamma')$ is no smaller than $-\epsilon$. If both of these checks fail, we compute $c^*_{\text{red}}(\Gamma')$ via an SOCP and prune $l'$ if $c^*_{\text{red}}(\Gamma') \geq -\epsilon$. We perform the first two checks before the third to avoid the computational expense of solving an SOCP, if possible.

Next, if $l'$ has not yet been pruned, we check if $l'$ is dominated by any existing labels at $\gamma_{i',j'}$, and prune $l'$ if so (Line 11). If $l'$ is not dominated, we prune labels stored at $\gamma_{i',j'}$ dominated by $l'$ and add $l'$ to $\mathfrak{D}[\gamma_{i',j'}]$ (Line 13). Then, if $\gamma_{i',j'} \neq \gamma_0$, we push $l'$ onto the priority queue, and if $\gamma_{i',j'} = \gamma_0$, we extract the tour corresponding to $l'$ and add it to the set $\mathcal{F}_{\text{price}}$ returned upon termination (Lines 14-15).

To speed up BPRC, we apply a heuristic from (Ozbaygin et al. 2017), where each $\mathfrak{D}[\gamma_{i,j}]$ only stores the label at $\gamma_{i,j}$ with the smallest $\min(\vec{g}_{\text{lb}}) - \lambda$ so far. This modification may prevent finding tours with negative reduced cost. Thus, if this modified Alg. 1 returns $\emptyset$, we run the unmodified Alg. 1.

Algorithm 1: SolvePricingProblem($\lambda, \mathcal{B}$)

1: PQUEUE = $[(\gamma_0, 0, 0, \underbrace{(0, 0, \ldots, 0)}_{n_{\text{tar}} \text{ times}}, (0), (0), \lambda_0)]$
2: $\mathfrak{D}$ = dict()
3: **for** $\gamma \in \mathcal{V}_{\text{tw}}$ **do** $\mathfrak{D}[\gamma] = \{\}$
4: $\mathcal{F}_{\text{price}} = \emptyset$
5: **while** PQUEUE is not empty **do**
6: $\quad l$ = PQUEUE.pop()
7: $\quad$ **for** each $\gamma_{i',j'} \in$ SuccessorTargetWindows$(l, \mathcal{B})$ **do**
8: $\quad\quad l' = (\gamma_{i',j'}, t', \sigma', \vec{b}', \vec{g}'_{\text{ub}}, \vec{g}'_{\text{lb}}, \lambda') =$ SuccessorLabel$(l, \gamma_{i',j'})$
9: $\quad\quad$ **if** $\gamma_{i',j'} = \gamma_0$ AND CheckDepotPrune$(l')$ **then**
10: $\quad\quad\quad$ continue
11: $\quad\quad$ **if** any $l'' \in \mathfrak{D}[\gamma_{i',j'}]$ dominates $l'$ **then** continue
12: $\quad\quad$ Delete all $l'' \in \mathfrak{D}[\gamma_{i',j'}]$ dominated by $l'$
13: $\quad\quad \mathfrak{D}[\gamma_{i',j'}]$.add$(l')$
14: $\quad\quad$ **if** $\gamma_{i',j'} \neq \gamma_0$ **then** PQUEUE.push$(l')$
15: $\quad\quad$ **else** $\mathcal{F}_{\text{price}} = \mathcal{F}_{\text{price}} \cup \{\text{PartialTour}(l')\}$
16: return $\mathcal{F}_{\text{price}}$

### 5.3 Generating Additional Feasible Solutions

To generate the initial incumbent and additional feasible solutions when the RMP is infeasible, we extend the feasible solution generation method from (Bhat et al. 2024), which applies to the single-agent MT-TSP, to handle multiple agents and capacity constraints. In particular, (Bhat et al. 2024) expands partial tours from a stack in depth-first fashion. We instead expand sets of partial tours from a stack, where each set contains up to one partial tour per agent. Appendix D (Bhat et al. 2026) fully describes the algorithm.

### 5.4 Theoretical Analysis

Branch-and-price finds an optimal solution to an ILP of type (1) if the dominance checks are correct, and the feasible solution generation method (Section 5.3) is complete.[7] We prove our dominance checks' correctness here; proofs of intermediate lemmas and completeness are in Appendix E (Bhat et al. 2026).

**Theorem 1.** *Let* $l = (\gamma_{i,j}, t, \sigma, \vec{b}, \vec{g}_{\text{ub}}, \vec{g}_{\text{lb}}, \lambda)$ *and* $l' = (\gamma_{i,j}, t', \sigma', \vec{b}', \vec{g}'_{\text{ub}}, \vec{g}'_{\text{lb}}, \lambda')$. *If* (5)-(7) *hold, then* $\Gamma = \text{PartialTour}(l)$ *dominates* $\Gamma' = \text{PartialTour}(l')$.

*Proof.* Let $T'$ be a least-reduced-cost tour beginning with $\Gamma'$, and let $\lambda_{\text{post}} = \sum_{i=1}^{n_{\text{tar}}} \alpha(i, T')\lambda_i - \lambda_0 - \lambda'$. Let $\tau_{\text{a}}{}'$ be a least-cost, speed-admissible trajectory executing $T'$. Let $\tau'_{\text{a,pre}}$ be

[7] Complete algorithms generate a feasible solution if one exists and report infeasible in finite time otherwise (Choset et al. 2005).

the portion of $\tau_a{}'$ ending at $\gamma_{i,j}$, and let $\tau'_{a,post}$ be the remaining portion. In Lemma 5 in Appendix E (Bhat et al. 2026), we prove that (7) and the speed-admissibility of $\tau'_{a,pre}$ ensure that there is a trajectory $\tau_{a,pre}$ executing $\Gamma$ satisfying

$$c(\tau_{a,pre}) - \lambda \leq c(\tau'_{a,pre}) - \lambda'. \tag{10}$$

Let $\tau_a$ be the concatenation of $\tau_{a,pre}$ and $\tau'_{a,post}$. Adding $c(\tau'_{a,post})$ to both sides of (10), we have

$$c(\tau_a) - \lambda \leq c(\tau'_a) - \lambda'. \tag{11}$$

Let $T'_{post}$ be the portion of $T'$ from $T'[\text{Len}(\Gamma') + 1]$ onward, and let $\sigma'_{post}$ be its depleted capacity. Let $T$ be the concatenation of $\Gamma$ with $T'_{post}$. Since $T'$ is a tour, $T'_{post}$ does not revisit targets visited by $\Gamma'$. Combining this with (6), $T'_{post}$ does not revisit targets visited by $\Gamma$ either, and thus $T$ does not revisit targets. Also, since $T'$ is a tour, its depleted capacity $\sigma' + \sigma'_{post}$ is no larger than $d_{max}$; combining this with (5), the depleted capacity of $T$, i.e. $\sigma + \sigma'_{post}$, is no larger than $d_{max}$. Thus, since $T$ revisits no targets, satisfies capacity constraints, and begins and ends at $\gamma_0$, $T$ is a tour.

Also, $\tau_a$ executes $T$, so $c^*(T) \leq c(\tau_a)$. This means

$$\begin{aligned} c^*_{red}(T) = c^*(T) - \lambda - \lambda_{post} &\leq c(\tau_a) - \lambda - \lambda_{post} && (12)\\ &\leq c(\tau_a{}') - \lambda' - \lambda_{post} && (13)\\ &= c^*_{red}(T') && (14) \end{aligned}$$

where (13) follows from (11). Thus we have a tour $T$ beginning with $\Gamma$ with $c^*_{red}(T) \leq c^*_{red}(T')$, so $\Gamma$ dominates $\Gamma'$. □

## 6 Numerical Results

We ran experiments using an Intel i9-9820X 3.3GHz CPU with 128 GB RAM, with 10 cores and hyperthreading disabled. Our baseline, called *Compact MICP*, is the MICP (Philip et al. 2025b), with additional decision variables and constraints based on (Desrochers et al. 1987) to enforce capacity constraints. In particular, we implemented (17)-(29) from (Philip et al. 2025b), then added an additional decision variable $\sigma_{i,j} \in [0, d_{max} - d_i]$ for each target-window $\gamma_{i,j}$, equal to the depleted capacity immediately before visiting $\gamma_{i,j}$, if $\gamma_{i,j}$ is visited. We then added a constraint

$$\sigma_{i,j} + d_i \leq \sigma_{i',j'} + (d_{max} + d_i)(1 - y_e) \; \forall e = (\gamma_{i,j}, \gamma_{i',j'}) \in \mathcal{E}_{tw} \tag{15}$$

where $y_e$ is a binary decision variable from (Philip et al. 2025b) which equals 1 if edge $e$ is traversed and 0 otherwise. We also implemented two ablations of BPRC, called *BPRC-ablate-dominance* and *BPRC-ablate-bounds*. BPRC-ablate-dominance is meant to show the benefit of our dominance checks at target-windows $\gamma \neq \gamma_0$ by removing these checks. In particular, BPRC-ablate-dominance does not store non-dominated labels at any $\gamma \neq \gamma_0$ and does not store $\vec{g}_{ub}$ in its labels.[8] BPRC-ablate-bounds is meant to show the benefits of our upper and lower-bounding methods for computing $\vec{g}_{ub}$ and $\vec{g}_{lb}$, respectively. In particular, BPRC-ablate-bounds computes $\vec{g}_{ub}$ and $\vec{g}_{lb}$ exactly using SOCPs. We solved Compact MICP and all SOCPs with Gurobi 12.03.

[8] BPRC-ablate-dominance still stores $\vec{g}_{lb}$ to break ties on the priority queue and prune labels at the depot.

We generated problem instances by extending the instance generation method from (Bhat et al. 2024) to multiple agents. In all instances, each target had two time windows and a demand of 1. Then we varied the number of targets, the capacity, the number of agents, and the time window lengths in Experiments 1 to 4. In these experiments, we set the number of segments per target, i.e. $n_{seg,tar}$, to 32 for BPRC, 4 for BPRC-ablate-bounds, and 8 for BPRC-ablate-dominance.[9] We varied $n_{seg,tar}$ in Experiment 5. We limited each planner's computation time to 10 min per instance. We also set a 100 GiB memory limit, which was only relevant for BPRC-ablate-dominance; the maximum measured memory usages for Compact MICP, BPRC-ablate-bounds, and BPRC were 1.41 GiB, 0.345 GiB, and 1.28 GiB, respectively. When we give runtime statistics (e.g. median) for BPRC-ablate-dominance, we only take statistics over instances where it did not reach the memory limit.

**Experiment 1 - Varying the Number of Targets:** We varied $n_{tar}$ from 5 to 25. Each instance had 3 agents with capacity $\lceil n_{tar}/n_{agt} \rceil$, i.e. this experiment considers severely limited capacity. Experiment 2 considers larger capacities. The sum of time window lengths per target was 50. The results are in Fig. 4 (a). For 15 to 20 targets, BPRC's median runtime is more than an order of magnitude smaller than Compact MICP's. BPRC has smaller runtime than the ablations as well, indicating that our dominance checks, and our cheap methods of evaluating them, are beneficial. For 25 targets, BPRC reaches the time limit in 5 instances and thus has similar median runtime to the baseline and ablations. For 25 targets, pricing was the bottleneck, taking more than 80% of the runtime in 8 of the 10 instances.

**Experiment 2 - Varying the Capacity:** We varied the capacity from 7 to 22, fixing the number of targets to 20, the number of agents to 3, and the sum of time window lengths per target to 50. The results are in Fig. 4 (b). BPRC has smaller median runtimes than Compact MICP for small capacities, and slightly larger median runtimes for large capacities. The ablations perform poorly for all capacities.

As stated in Section 2, one reason that branch-and-price is favored over compact formulations for VRPs is that the relaxation of branch-and-price's integer program tends to be tighter than the relaxations of compact formulations. To validate that this holds in the MT-VRP, for each instance in Experiment 2, for both BPRC and Compact MICP, we computed a lower bound $c_{LB}$ on the optimal cost by solving the convex relaxation of the method's integer or mixed-integer program.[10] We then computed the gap of the optimal cost from each method's lower bound, defined as $\%\text{Gap} = (c_{opt} - c_{LB})/c_{LB} * 100\%$. As shown in Fig. 4 (c), BPRC's

[9] We tuned $n_{seg,tar}$ for BPRC to achieve the best median runtime over 10 instances with 20 targets, separate from the instances in the presented experiments. The ablations' median runtimes were equal to the time limit for each tested $n_{seg,tar}$ for these tuning instances, so we instead tuned the ablations on 10 instances with 15 targets.

[10] For BPRC, $c_{LB}$ is the optimal cost of LP-$\mathcal{B}$ at the root branch-and-bound node, i.e. computing it does not require enumerating $\mathcal{S}$.

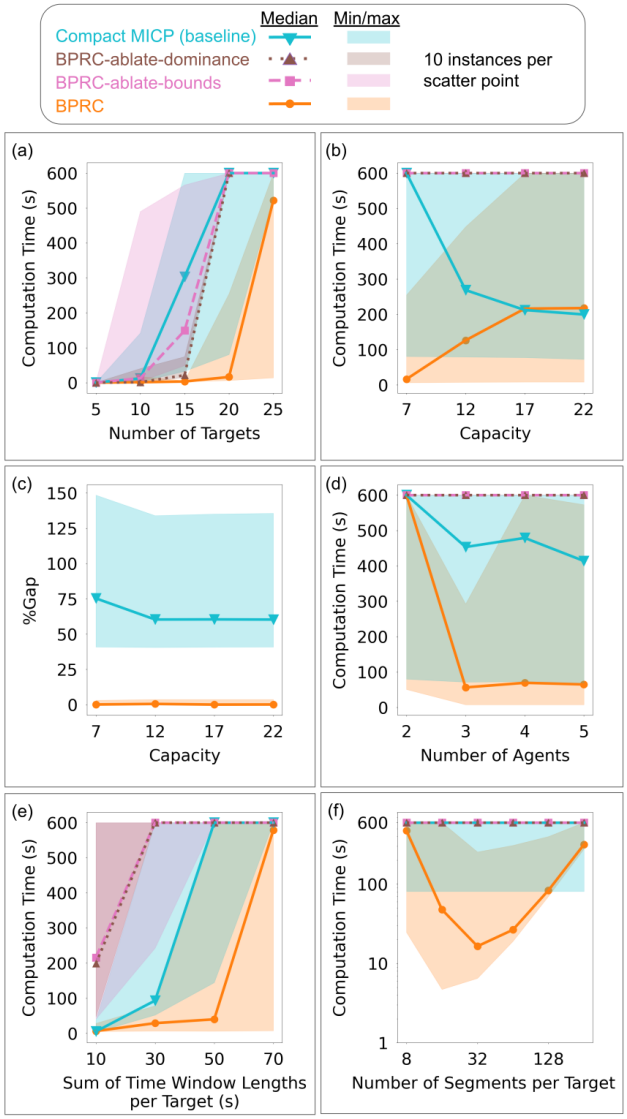


Figure 4: (a) Varying the number of targets. The gap in computation time between BPRC and the baseline and ablations widens as we increase the number of targets up to 20. The gap shrinks at 25 targets because BPRC reaches the time limit in 5 instances. We do not show a computation time for BPRC-ablate-dominance for 25 targets because it reached the memory limit in every instance. (b) Varying the capacity. BPRC shows advantage for small capacities. (c) %Gap (as defined in Experiment 2) of each method's relaxation solution. BPRC's relaxation has a smaller %Gap, and is thus tighter, in every instance. (d) Varying the number of agents. BPRC shows advantage when the number of agents is 3 or larger. (e) Varying the time window lengths. BPRC's performance advantage widens as we increase time window length up to 50, then shrinks because BPRC reaches the time limit in many instances. (f) Varying the number of segments in BPRC. Horizontal and vertical axes are log-scaled. MICP's min, median, and max are shown as flat lines without scatter points because the quantity being varied is specific to BPRC. We only show MICP's computation time to indicate that BPRC has smaller median computation time for a wide range of parameter values.

relaxation is empirically tighter than Compact MICP's.[11]

**Experiment 3 - Varying the Number of Agents:** We varied $n_{\text{agt}}$ from 2 to 5. Each instance had 20 targets, capacity 10, and sum of time window lengths per target equal to 50. Fig. 4 (d) shows the results. BPRC's median computation time drops steeply as we increase $n_{\text{agt}}$ from 2 to 3, then remains roughly constant as we increase $n_{\text{agt}}$ from 3 to 5. For 3 to 5 agents, BPRC's median computation time is significantly lower than the baseline's and the ablations'.

The large runtime for 2 agents appears to be because the agent limit (1b) is directly affecting the optimal cost when $n_{\text{agt}} = 2$, but not when $n_{\text{agt}} > 3$. That is, in 4 of the 5 instances where some method found the optimum for $n_{\text{agt}} = 2$, increasing $n_{\text{agt}}$ to 3 actually resulted in a different optimal solution with a smaller cost. However, increasing $n_{\text{agt}}$ from 3 to 4 did not change the optimal solution in any instance. Increasing $n_{\text{agt}}$ can only decrease the optimal cost if $\lambda_0$ is strictly negative, which is what we often observed in the 2-agent instances: the median $\lambda_0$ value was -52, and the largest was -47231. Meanwhile, among all instances with $n_{\text{agt}} > 2$, the median $\lambda_0$ was 0, and the largest was only -889. Prior work (Afsar, Afsar, and Palacios 2021) has observed that large $\lambda_0$ values can slow the convergence of branch-and-price, which is what we observed here for $n_{\text{agt}} = 2$.

**Experiment 4 - Varying the Time Window Lengths:** We varied the sum of time window lengths per target from 10 to 70. Each instance had 20 targets and 3 agents with capacity 7. The results are in Fig. 4 (e). All methods' runtimes tend to increase with time window length. BPRC's median runtime is smaller than Compact MICP's for lengths 30 and 50, and smaller than the ablations' for lengths 10, 30, and 50.

**Experiment 5 - Varying the Number of Segments:** We varied $n_{\text{seg,tar}}$ from 8 to 256, on a log-scale, i.e. we tested values 8, 16, 32, etc. Each instance had 20 targets, 3 agents with capacity 7, and sum of time window lengths per target equal to 50. The results are in Fig. 4 (f). BPRC's median runtime is smaller than the other methods' minimum runtime for $n_{\text{seg,tar}}$ from 16 to 64 (inclusive). Thus, BPRC shows advantage even when we change $n_{\text{seg,tar}}$ by a factor of 4.

BPRC's runtime is large for large $n_{\text{seg,tar}}$ because computing distances between all pairs of segments becomes expensive. BPRC's runtime is large for small $n_{\text{seg,tar}}$ because dominance checks become weaker, causing more retained labels per target-window, making pricing expensive.

## 7 Conclusion

In this paper, we introduced BPRC, a new algorithm to find the optimum for the MT-VRP. We demonstrated that it finds the optimum with less computation time than a baseline, particularly when the agents have small capacities. As mentioned in Section 6, pricing is a bottleneck in BPRC, and thus a direction for future work is to employ techniques such as the *ng*-path relaxation (Baldacci, Mingozzi, and Roberti 2011) to reduce time spent in the pricing problem.

[11] We only take statistics over instances where at least one method found the optimum, i.e. 8 instances for capacity 17 and 9 instances for capacity 22.

## Acknowledgements

This material is partially based on work supported by the National Science Foundation (NSF) under Grant No. 2120219 and 2120529. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect views of the NSF.

## References

Afsar, H. M.; Afsar, S.; and Palacios, J. J. 2021. Vehicle routing problem with zone-based pricing. *Transportation research part E: logistics and transportation review*, 152: 102383.

Archetti, C.; Coelho, L.; Speranza, M.; and Vansteenwegen, P. 2025. Beyond fifty years of vehicle routing: Insights into the history and the future. *European Journal of Operational Research*.

Baldacci, R.; Mingozzi, A.; and Roberti, R. 2011. New route relaxation and pricing strategies for the vehicle routing problem. *Operations research*, 59(5): 1269–1283.

Bard, J. F.; Kontoravdis, G.; and Yu, G. 2002. A branch-and-cut procedure for the vehicle routing problem with time windows. *Transportation Science*, 36(2): 250–269.

Barnes, J. W.; Wiley, V. D.; Moore, J. T.; and Ryer, D. M. 2004. Solving the aerial fleet refueling problem using group theoretic tabu search. *Mathematical and computer modelling*, 39(6-8): 617–640.

Bhat, A.; Gutow, G.; Ren, Z.; Rathinam, S.; and Choset, H. 2026. Optimal Solutions for the Moving Target Vehicle Routing Problem via Branch-and-Price with Relaxed Continuity. arXiv:2603.00663.

Bhat, A.; Gutow, G.; Vundurthy, B.; Ren, Z.; Rathinam, S.; and Choset, H. 2024. A Complete Algorithm for a Moving Target Traveling Salesman Problem with Obstacles. In *International Workshop on the Algorithmic Foundations of Robotics*. Springer.

Boland, N.; Dethridge, J.; and Dumitrescu, I. 2006. Accelerated label setting algorithms for the elementary resource constrained shortest path problem. *Operations Research Letters*, 34(1): 58–68.

Brown, G. G.; DeGrange, W. C.; Price, W. L.; and Rowe, A. A. 2017. Scheduling combat logistics force replenishments at sea for the US Navy. *Naval Research Logistics (NRL)*, 64(8): 677–693.

Choset, H.; Lynch, K. M.; Hutchinson, S.; Kantor, G. A.; and Burgard, W. 2005. *Principles of robot motion: theory, algorithms, and implementations*. MIT press.

Costa, L.; Contardo, C.; and Desaulniers, G. 2019. Exact branch-price-and-cut algorithms for vehicle routing. *Transportation Science*, 53(4): 946–985.

Desrochers, M.; Desrosiers, J.; and Solomon, M. 1992. A new optimization algorithm for the vehicle routing problem with time windows. *Operations research*, 40(2): 342–354.

Desrochers, M.; Lenstra, J. K.; Savelsbergh, M. W.; and Soumis, F. 1987. Vehicle routing with time windows: optimization and approximation. Technical report, CWI. Department of Operations Research and System Theory [BS] Vancouver, BC ....

Feillet, D. 2010. A tutorial on column generation and branch-and-price for vehicle routing problems. *4or*, 8(4): 407–424.

Feillet, D.; Dejax, P.; Gendreau, M.; and Gueguen, C. 2004. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks: An International Journal*, 44(3): 216–229.

Gambella, C.; Naoum-Sawaya, J.; and Ghaddar, B. 2018. The vehicle routing problem with floating targets: Formulation and solution approaches. *INFORMS Journal on Computing*, 30(3): 554–569.

Hammar, M.; and Nilsson, B. J. 1999. Approximation results for kinetic variants of TSP. In *Automata, Languages and Programming: 26th International Colloquium, ICALP'99 Prague, Czech Republic, July 11–15, 1999 Proceedings 26*, 392–401. Springer.

Helvig, C. S.; Robins, G.; and Zelikovsky, A. 2003. The moving-target traveling salesman problem. *Journal of Algorithms*, 49(1): 153–174.

Kohl, N.; Desrosiers, J.; Madsen, O. B.; Solomon, M. M.; and Soumis, F. 1999. 2-path cuts for the vehicle routing problem with time windows. *Transportation science*, 33(1): 101–116.

Li, B.; Page, B. R.; Hoffman, J.; Moridian, B.; and Mahmoudian, N. 2019. Rendezvous Planning for Multiple AUVs With Mobile Charging Stations in Dynamic Currents. *IEEE Robotics and Automation Letters*, 4(2): 1653–1660.

Ozbaygin, G.; Karasan, O. E.; Savelsbergh, M.; and Yaman, H. 2017. A branch-and-price algorithm for the vehicle routing problem with roaming delivery locations. *Transportation Research Part B: Methodological*, 100: 115–137.

Philip, A. G.; Ren, Z.; Rathinam, S.; and Choset, H. 2025a. C$^*$: A New Bounding Approach for the Moving-Target Traveling Salesman Problem. *IEEE Transactions on Robotics*, 41: 4663–4678.

Philip, A. G.; Ren, Z.; Rathinam, S.; and Choset, H. 2025b. A Mixed-Integer Conic Program for the Multi-Agent Moving-Target Traveling Salesman Problem. In *2025 IEEE 21st International Conference on Automation Science and Engineering (CASE)*, 492–498.

Smith, C. D. 2021. *Assessment of genetic algorithm based assignment strategies for unmanned systems using the multiple traveling salesman problem with moving targets*. University of Missouri-Kansas City.

Stieber, A.; and Fügenschuh, A. 2022. Dealing with time in the multiple traveling salespersons problem with moving targets. *Central European Journal of Operations Research*, 30(3): 991–1017.

Toth, P.; and Vigo, D. 2014. *Vehicle routing: problems, methods, and applications*. SIAM.