informs
http://pubsonline.informs.org/journal/ijoc

INFORMS JOURNAL ON COMPUTING
Vol. 00, No. 0, Xxxxx 0000, pp. 000–000
ISSN 0899-1499, EISSN 1526-5528

Submitted to *Informs Journal on Computing*

An Exact Framework for Solving the Space-Time Dependent TSP

Isaac Rudich
Polytechnique Montréal, Montreal, Canada, isaac.rudich@polymtl.ca

Manuel López-Ibáñez
University of Manchester, Manchester, UK, manuel.lopez-ibanez@manchester.ac.uk

Michael Römer
Universität Bielefeld, Bielefeld, Germany, michael.roemer@uni-bielefeld.de

Quentin Cappart
Polytechnique Montréal, Montreal, Canada and UCLouvain, Louvain-la-Neuve, Belgium, quentin.cappart@polymtl.ca

Louis-Martin Rousseau
Polytechnique Montréal, Montreal, Canada, louis-martin.rousseau@polymtl.ca

Abstract. Many real-world scenarios involve solving bi-level optimization problems in which there is an outer discrete optimization problem, and an inner problem involving expensive or black-box computation. This arises in space-time dependent variants of the Traveling Salesman Problem, such as when planning space missions that visit multiple astronomical objects. Planning these missions presents significant challenges due to the constant relative motion of the objects involved. There is an outer combinatorial problem of finding the optimal order to visit the objects and an inner optimization problem that requires finding the optimal departure time and trajectory to travel between each pair of objects. The constant motion of the objects complicates the inner problem, making it computationally expensive. This paper introduces a novel framework utilizing decision diagrams (DDs) and a DD-based branch-and-bound technique, Peel-and-Bound, to achieve exact solutions for such bi-level optimization problems, assuming sufficient inner problem optimizer quality. The framework leverages problem-specific knowledge to expedite search processes and minimize the number of expensive evaluations required. As a case study, we apply this framework to the Asteroid Routing Problem, a benchmark problem in global trajectory optimization. Experimental results demonstrate the framework's scalability and ability to generate robust heuristic solutions for tested instances. Many of these solutions are exact, contingent on the assumed quality of the inner problem's optimizer.

Key words: Decision Diagrams, Spacecraft Trajectory Optimization, Dynamic Programming, Combinatorial Optimization, Sequencing

MSC2000 subject classification: 90-08

1. Introduction

In various real-world scenarios, we must determine optimal sequences for visiting locations (or scheduling jobs), when the associated travel costs (or setup times) are defined by an expensive black-box function. These scenarios are in a class of bi-level optimization problems. The *outer* problem involves finding an optimal permutation, while the *inner* problem evaluates the cost of this permutation, often through an expensive black-box function. This makes finding exact solutions challenging.

Instances of this problem include time-dependent routing problems (Gendreau et al. 2015), when travel-costs are computed on-demand (Ehmke et al. 2016), or require computing optimal vehicle speeds (Andersson et al. 2015). Other examples are planning missions in space (Shirazi et al. 2018), touring Jupiter's Galilean moons (Izzo et al. 2013), global optimization for multiple gravity assist trajectories (Abdelkhalik and Gad 2012, Ceriotti and Vasile 2010, Vasile and De Pascale 2006), active space debris removal (Izzo et al. 2015), and spacecraft formation control (Lee et al. 2007). These problems are often solved using bi-level heuristics that combine numerical optimization methods with tree search methods like Beam-Search (Petropoulos et al. 2014), Lazy Race Tree Search (Izzo et al. 2013), Monte-Carlo Tree Search (Hennes and Izzo 2015) and Beam-ACO (Simões et al. 2017). Other approaches treat the inner problem as an expensive black-box function (Zaefferer et al. 2014, Irurozki and López-Ibáñez 2021, Santucci and Baioletti 2022, Chicano et al. 2023). To our knowledge, no exact method exists for finding optimal solutions to these problems. Optimal solutions to global trajectory optimization problems are crucial due to their significant economic costs (hundreds of millions of dollars), and long mission horizons (decades).

We propose a framework for finding exact solutions by representing the outer problem using decision diagrams (DDs) (Bergman et al. 2016, Cire and van Hoeve 2013) and solving it with a DD-based branch-and-bound technique called Peel-and-Bound (Rudich et al. 2023) combined with DD-based search techniques (Gillard 2022). This approach treats the inner problem as an expensive black-box function evaluated on-demand and uses problem-specific knowledge to reduce evaluations.

We consider the Asteroid Routing Problem (ARP) (López-Ibáñez et al. 2022), inspired by the 11$^{\text{th}}$ Global Trajectory Optimization Competition (https://gtoc11.nudt.edu.cn). The ARP is a space-time dependent variant of the TSP, where a spacecraft from Earth visits a set of asteroids. The objective is to find the asteroid permutation that minimizes a weighted sum of fuel consumption and total travel time. Traveling between asteroids involves computing optimal departure and travel times. This requires solving a black-box function using a deterministic optimizer that always returns the same objective function value for a given permutation. This makes the ARP a simplified benchmark for the class of problems described, allowing researchers to focus on the *outer* problem of optimizing the asteroid permutation. Brute force search is the only known method to identify an optimal solution to the ARP, but is just feasible for a very small number of asteroids. All current approaches are heuristic and time-consuming (López-Ibáñez et al. 2022, Chicano et al. 2023).

This paper presents the first exact approach for solving the ARP, proving optimality for the outer problem, assuming that the inner problem is solved deterministically with a level of performance that does not affect the outer problem's optimality. To achieve this, we propose an extension of Peel-and-Bound that operates with DDs in which arc values are not exact but form bounds. The bounds are obtained by relaxing the inner problem, and the overall approach is designed such that the number of calls to the black box is kept as small as possible. Our experimental results demonstrate that our approach can quickly generate strong heuristic solutions to the ARP. We provide exact and new best-known solutions for several ARP instances.

2. Problem Description

This section outlines and explains the ARP. First, we define the general black-box time-dependent routing problem, then we show how the ARP serves as an example.

2.1. Black-Box Time-Dependent Routing Problems

We consider routing problems that seek an optimal permutation π of n locations $A = a_1, \ldots, a_n$ minimizing a function $f\colon S_n \to \mathbb{R}$, where S_n denotes all permutations of A. Specifically, $f(\pi)$ is given by the black box function $\sum_{i=2}^{n} \mathcal{B}(\pi_{i-1}, \pi_i, t_i)$, with each travel time t_i from π_{i-1} to π_i depending on the visitation time of π_i. When a closed-form for $\mathcal{B}$ is available, the problem reduces to the time-dependent TSP (TD-TSP); otherwise, if evaluating $\mathcal{B}$ requires solving an optimization problem, we call it a *black-box time-dependent routing problem*, of which the ARP (defined next) is an example.

2.2. The Asteroid Routing Problem

In the ARP (López-Ibáñez et al. 2022), we are given a set of n asteroids $A = \{a_1, \ldots, a_n\}$ orbiting around the sun that a spacecraft launched from Earth (a_0) must visit. The spacecraft does not need to return to Earth. For simplicity, the ARP treats the spacecraft transfer from Earth to the first-visited asteroid as any other transfer between asteroids, i.e., Earth has no escape velocity.

Given an arrival time (epoch) η at asteroid a, any feasible transfer to another asteroid a' is characterized by a wait time τ bounded by $[0, \tau_{\text{max}}]$ and a travel time t bounded by $[1, t_{\text{max}}]$ determining the arrival time at a' is $\eta + \tau + t$. The lower bound of τ is 0 because the spaceship can depart immediately without waiting. The lower bound of t is 1 because a transfer will always require a nonzero time (at least 1 day in the ARP). The upper bounds τ_{max} and t_{max} are set to 730 days in the original definition of the ARP (López-Ibáñez et al. 2022). The travel cost z from a to a' depends on η, τ and t. The following black box function $\mathcal{B}$ returns a tuple (τ, t, z) with the cost-minimizing wait and transfer times τ and t along the the cost z:

$$\mathcal{B}(a, a', \eta)\colon A \times A \times \mathbb{R}^+ \to [0, \tau_{\text{max}}] \times [1, t_{\text{max}}] \times \mathbb{R}^+ \tag{1}$$

An example transfer is depicted in Online Supplement A Fig. EC.1. Given a solution $\pi = (\pi_0 = a_0, \pi_1, \ldots, \pi_n)$, where the subsequence $(\pi_1, \ldots, \pi_n)$ is a permutation of the asteroids in A that indicates the order in which they will be visited, the objective function of the ARP is:

$$\text{Minimize} \quad f(\pi) = \sum_{i=1}^{n} z_i \qquad \text{where} \quad (\tau_i, t_i, z_i) = \mathcal{B}(\pi_{i-1}, \pi_i, \eta_i) \quad \forall i \in \{1, \ldots, n\} \tag{2}$$

Besides, $\eta_i = \eta_{i-1} + \tau_{i-1} + t_{i-1}$ is the arrival time at π_{i-1} and η_1 is a given mission start time, which can be simplified to be zero. While transfer costs depend on the absolute time, we can normalize the mission start time to zero. This fixed reference point allows us to measure all subsequent transfer costs relative to the start time without affecting the overall formulation. Online Supplement A Fig. EC.2 visualizes a solution.

The black-box function $\mathcal{B}$ depends not only on the two asteroids being visited but also on the arrival time η_i. Thus the ARP is an example of the class of time-dependent problems discussed in the introduction.

2.3. Trajectory Optimization

In the ARP, and in many trajectory optimization problems, the spacecraft only uses *impulsive* maneuvers. An impulsive maneuver changes the spacecraft's velocity in a negligible amount of time, simplifying the calculation of the maneuver's effect on the spacecraft's orbit. As a result, the maneuver's effect can be modeled as an instant change in the spacecraft's velocity vector $\Delta\vec{v}$, without accounting for the dynamics of the propulsion system or the influence of external forces during the thrust application.

Lambert's Problem (Izzo 2015) calculates an orbit that connects two points in space-time departing at time $\eta + \tau$ and arriving at time $\eta + \tau + t$. Using the solution to Lambert's Problem, a transfer between the orbits of two asteroids a and a' can be achieved with two impulsive maneuvers. Initially, at time (epoch) η, the spacecraft follows the same orbit around the sun as asteroid a. *Waiting* in this orbit does not consume any fuel but changes its relative distance to other asteroids. The first impulse $\Delta\vec{v}_1$ happens τ days after the current time η (epoch), and moves the spacecraft from its current orbit to an orbit that will intercept the target asteroid a' after traveling for t days. When the spacecraft intercepts a', the second impulse $\Delta\vec{v}_2$ modifies its orbit to match the orbit of a', so that the spacecraft follows the same orbit as a' without consuming any additional fuel. Usually, the total magnitude of velocity change (the Euclidean L2 norm) is used as a surrogate for fuel consumption/energy costs: $\Delta V = \|\Delta\vec{v}_1\|_2 + \|\Delta\vec{v}_2\|_2$ where $(\Delta\vec{v}_1, \Delta\vec{v}_2) = \text{Lambert}(a, a', \eta + \tau, t)$.

We still need to decide τ, how long the spacecraft should wait at a, and t, the transfer time to a'. Waiting changes the relative distance to other asteroids, which may reduce the fuel or time needed to reach the next asteroid. The optimal values of τ and t depend on which objectives are optimized.

In trajectory optimization, there are many possible objectives. Two typical objectives are the minimization of fuel (energy) consumption and the total mission time ($\tau + t$). These two objectives are often in conflict. The ARP aggregates the two objectives as shown below:

$$\begin{aligned} z = f_{\text{inner}}(a, a', \eta, \tau, t) &= \Delta V + \frac{2\,\text{km/s}}{30\,\text{days}} \cdot (\tau + t) \\ \text{s.t.} \quad & \tau \in [0, \tau_{\max}],\ t \in [1, t_{\max}] \\ \text{where} \quad & \Delta V = \|\Delta\vec{v}_1\|_2 + \|\Delta\vec{v}_2\|_2 \quad \text{and} \quad (\Delta\vec{v}_1, \Delta\vec{v}_2) = \text{Lambert}(a, a', \eta + \tau, t) \end{aligned} \tag{3}$$

where the trade-off constant $\frac{2\,\text{km/s}}{30\,\text{days}}$ was chosen by López-Ibáñez et al. (2022). The numerator represents the increase of ΔV worth paying for a 30 day reduction in mission time. This trade-off may change from one mission to another and our approach does not depend on a specific value. The minimization of f_{inner} becomes an *inner* optimization problem whose solution gives the values τ, t and z returned by $\mathcal{B}$ (Eq. 2).

In the ARP, this inner problem is commonly solved using Sequential Least Squares Programming (SLSQP) (Kraft 1988), a deterministic optimizer. SLSQP iteratively calculates the position and velocity of the asteroids and then solves Lambert's Problem to evaluate the objective. Even if time is discretized into days, this inner optimization would be too slow to allow us to calculate the cost of every optimal trajectory for each pair of asteroids at every point in time.

A mission with n asteroids may have up to $n(\tau_{\max} + t_{\max})$ possible values of η. In a mission where $n = 10$, calculating every possible trajectory from just one asteroid to one other asteroid for every possible departure time when time is discretized into days would take about 10 minutes using the inner optimizer in our implementation. Thus, calculating the full cost matrix for every ordered pair of asteroids ($n(n-1)$ pairs) at every possible departure day would take about 15 hours. Furthermore, departure time is continuous, not discrete, so this cost matrix would not even fully represent the problem, although it would certainly provide information that is useful for solving it.

3. Decision Diagrams and Peel-and-Bound

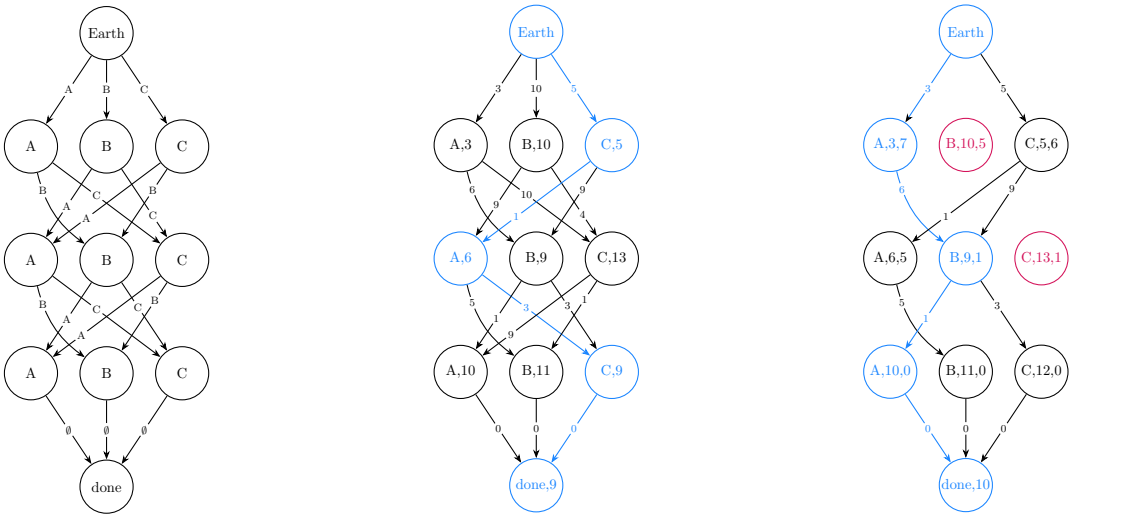
This section briefly reviews DDs for combinatorial optimization (Bergman et al. 2016) and the Peel-and-Bound algorithm (Rudich et al. 2022, 2023, Rudich 2024). Using the TD-TSP as a running example, we set the stage for extending these techniques to black-box time-dependent routing problems, such as the ARP.

3.1. Relaxed Decision Diagrams

A DD is a directed layered graph that encodes solutions to an optimization problem as paths from the root (denoted here as *Earth*) to the terminal (denoted here as *done*). A DD is *relaxed* if it encodes every feasible solution, but also encodes infeasible solutions (Cire and van Hoeve 2013). It is usually obtained by imposing a maximum width ω that is obtained merging non-equivalent nodes on each layer. Figure 1a displays a valid relaxed DD for a TD-TSP with asteroids $\{A, B, C\}$. Each arc is labeled with the decision about which asteroid is the next destination, and each node is labeled with the current asteroid. Every feasible permutation of asteroids is trivially encoded as a path from the root to the terminal, but so are several infeasible solutions, such as $A \to B \to A$. A relaxed DD can also be *weighted*. In a weighted DD, every arc is bound by the impact on the objective function of transitioning from the arc's origin to its destination. In weighted relaxed DDs, the length of the shortest path from the root to the terminal is a relaxed (dual) bound on the optimization problem being represented. This result is formalized in the following lemma:

LEMMA 1 **(Lower Bound from Shortest Path)**. *Let a weighted relaxed decision diagram be given, where each arc a has an associated cost $w(a)$. Denote by $z_{\downarrow}(t)$ the cost of the shortest path from the root to*

Figure 1 **Relaxed decision diagrams for an ARP with asteroids $\{A, B, C\}$.**



(a) Unweighted (b) Weighted (c) Constraint Propagation

the terminal node t. Then, $z_{\downarrow}(t)$ is a valid lower bound on the cost of any feasible solution to the optimization problem represented by the DD. That is, for any feasible route, the cost is at least $z_{\downarrow}(t)$.

Fig. 1b displays a weighted version of Fig. 1a. Each arc still assigns the same decision, but the arc weights now display the cost of transferring between those two asteroids at that point in the sequence. Each node is now labeled with the shortest path length to that node. Thus, the length of the shortest path ($C \to A \to C$, which is highlighted) is a valid lower bound on this TD-TSP. In other words, no matter what permutation of asteroids is used, there is no route for the spacecraft that admits a cost of less than 9 for this example. However, since C is visited twice, the shortest path does not yield a feasible solution. To improve bounds and obtain feasible solutions, we *refine* the relaxed DD using the technique discussed next.

3.2. Refining a Relaxed Diagram

To get an optimal solution to the ARP, we will leverage three DD refinement techniques. We will illustrate the TD-TSP. Let $\mathcal{M}$ be our relaxed DD, let π^* be the shortest path through $\mathcal{M}$, let $\underline{z}$ be the length of π^* (and thus a relaxed bound on the optimal cost). In Fig. 1b, we have a relaxed DD with a shortest path of $\pi^* = (C \to A \to C)$ (which is infeasible) and a cost of $\underline{z} = 9$. To make progress, we must now remove the path encoding π^* from the DD without removing any different and potentially optimal solutions. This process is called refinement. We will now discuss three techniques for refining DDs.

3.2.1. Constraint Propagation The first refinement technique is constraint propagation. In the context of DDs, this technique involves adding constraints to nodes or arcs such that any element violating a constraint can be removed. For example, in our approach, each node u (as illustrated in Fig. 1b) stores the length of the shortest path from the root to that node, denoted by $z_{\downarrow}(u)$. Additionally, if each node stores the length of the shortest path from u to the terminal, denoted by $z_{\uparrow}(u)$, we can apply the following result:

LEMMA 2 **(Constraint Propagation via Path Lengths).** *Let u be any node in a relaxed DD. If*

$$z_{\downarrow}(u) + z_{\uparrow}(u) \geq \overline{z},$$

then no complete path passing through u can yield a solution with cost strictly less than $\overline{z}$. Consequently, the node u (and any incident arcs) can be safely removed from the DD without excluding any candidate solution that improves upon the best-known solution $\overline{z}$.

Cire and van Hoeve (2013) explore several such constraints for solving sequencing problems with DDs; our proposed algorithm includes all of them. Fig. 1c demonstrates the removal of arcs that cannot be included in a solution to our example when we have an upper bound of $\overline{z} = 12$.

3.2.2. Node Splitting In our example, we must remove π^* from our DD because it represents an infeasible solution. This can be achieved with our second refinement technique: node splitting. Informally, a node split: (1) creates a new node along with copies of the out-arcs of the old node, (2) chooses a subset of the arcs pointing at the old node, (3) redirects those arcs to point at the new node and (4) removes all out-arcs from the old and the new node that cannot be part of a feasible or optimal solution.

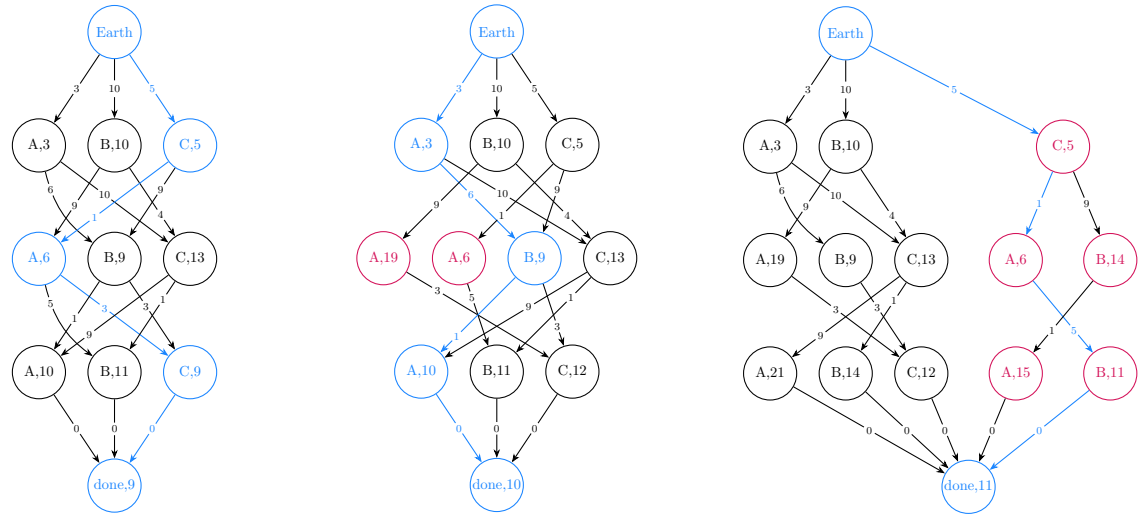
Normally when a node u is selected to be split, it is because a path being removed passes through u. Let ϕ be the label of the in-arc of u on that path. To choose a subset of the arcs pointing at the old node u during a split, we need a sorting function for the in-arcs of u. Let $All_v^{\downarrow}$ be the set of labels visited by every path from the root node r to node v. For the ARP, an arc label is the destination asteroid represented by that arc in the transfer. Thus, $All_v^{\downarrow}$ is the set of asteroids visited on every path from r to v. Let a_{vu} be an arc from u to v and let $l(a)$ be the label on arc a. We use the function $h(a_{vu}, \phi)$ from Cire and van Hoeve (2013) to determine which arcs to split on; it is defined as follows:

$$h(a_{vu}, \phi) = \begin{cases} \textbf{true} & \text{if } \phi \in \{All_v^{\downarrow} \cup l(a_{vu})\} \\ \textbf{false} & \text{otherwise} \end{cases} \tag{4}$$

Using this function when splitting enforces the constraint that if an asteroid ϕ is visited on every path from the root node to the new node u', then ϕ cannot be revisited on any path from u' to the terminal. Node splitting is explained using our example in the next paragraph, and visualized in Fig. 2.

A node u is split by making a copy u'. The in-arcs are heuristically sorted into two sets, and distributed between the two nodes. The out-arcs are copied so that both nodes have a copy of each arc. Then arcs are removed if they can be shown not to contain the optimal solution. In our example (Figure 2b), node $(A, 6)$ has two in-arcs $\{(B \to A), (C \to A)\}$, and two out-arcs $\{(A \to B), (A \to C)\}$. We split the node and give one in-arc to each, as well as a copy of both out-arcs. Then we observe that one node represents only paths starting with $(B \to A)$, and the path $(B \to A \to B)$ is infeasible, so we can remove the B arc from this node. We also observe that the other node represents only paths starting with $(C \to A)$, so we can remove the C arc from this node as well. Thus, splitting the $(A, 6)$ node strengthens the lower bound from 9 to 10.

Figure 2 **Refining relaxed decision diagrams with node splitting and node peeling**



(a) Initial Diagram (b) Node Splitting (c) Node Peeling

3.2.3. Peeling The third refinement technique is the peel operation (Rudich et al. 2022, 2023). The peel operation starts by picking a node u, and then iteratively splits nodes top-down until we have separated all of the paths passing through u in $\mathcal{M}$ into a discrete graph that only connects at the root and terminal. The process of repeatedly peeling and splitting nodes in a relaxed DD until reaching an optimal solution is called Peel-and-Bound (PnB). Algorithm 1 provides a formal treatment of the peel operation.

In our example (Figure 2a), we want to perform a peel operation that removes the infeasible path $\pi^* = (C \to A \to C)$ while also splitting the graph into a DD containing only paths that start with C, and a separate DD that contains no paths starting with C. To do this, we start by splitting the A and B nodes on the third layer into a version with C as a parent, and a version that does not have C as a parent. Subsequently, the A and B nodes on the fourth layer are split so only paths that do not start with C remain with the original nodes. Finally, we remove several arcs from both parts of the DD, because they represent paths that visit the same asteroid multiple times. As an example, all paths in the right sub-DD start with asteroid C, thus all C arcs are removed from that part of the DD. Not only did the peeling strengthen the DD, but it produced an optimal solution, since the shortest path through the DD, $(C \to A \to B)$ with $z = 11$, is feasible.

3.3. Feasible Solutions from Embedded Restricted DDs

Restricted DDs are a tool for obtaining feasible solutions in DD-based optimization. In our setting, a relaxed DD (constrained by a maximum width ω) contains every permutation of decisions as a path from the root to the terminal, but accomplishes this by admitting infeasible permutations. A restricted DD is similarly constrained by a maximum width ω_s, but it contains only feasible permutations of decisions as paths from the root to the terminal. This can be thought of as a generalized greedy heuristic, similar to beam search (Ow and Morton 1988), that generates a fixed number of solutions and only continues to explore the ones

Algorithm 1: Peeling Procedure (Rudich et al. 2022, 2023)

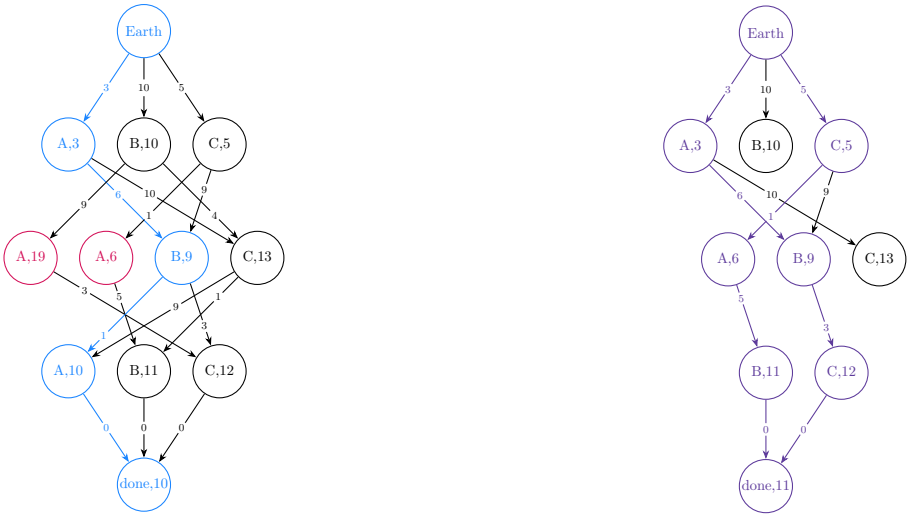
Input: a relaxed decision diagram $\mathcal{D}$ with root r and terminal t, and a node u in $\mathcal{D}$
Output: a relaxed decision diagram $\mathcal{D}_u$ peeled from $\mathcal{D}$, and what remains of $\mathcal{D}$
Let $in(u)$ for some node (or diagram) u be the set of arcs that end at u, and $out(u)$ be the set of arcs originating from u
$in(u) \leftarrow \emptyset$ // Remove the in-arcs of u
$\mathcal{D} \leftarrow \mathcal{D} \backslash u$; $\mathcal{D}_u \leftarrow u$
while $in(\mathcal{D}) \cap out(\mathcal{D}_u) \neq \emptyset$ **do**
  foreach node $m \in \mathcal{D}$ with an in arc that originates in $\mathcal{D}_u$ **do**
    Create a new node m' and add it to $\mathcal{D}_u$
    foreach arc $a_{md} \in out(m)$ **do**
      Add arc $a_{m'd}$
    end
    foreach arc $a \in in(m)$ that originates in $\mathcal{D}_u$ **do**
      Change the destination of a to m'
      filter(a)
    end
    foreach arc $a \in out(m)$ **do**
      filter(a)
    end
  end
end
while $\exists m \in \mathcal{D}$ with $in(m) = \emptyset \vee out(m) = \emptyset$ (excluding r and t) **do**
  $in(m) \leftarrow \emptyset$; $out(m) \leftarrow \emptyset$
  $\mathcal{D} \leftarrow \mathcal{D} \backslash \{m\}$
end
return $(\mathcal{D}_u, \mathcal{D})$

with the lowest cost. Typical search procedures for DD-based solvers use relaxed DDs to generate lower bounds on minimization problems and separately construct restricted DDs to generate upper bounds. After generating a restricted DD, the shortest path from the root to the terminal t is the best (lowest cost) solution found using the search procedure, and its length $z_{\downarrow}(t)$ is the solution cost. In this work, we do not generate these restricted DDs from scratch, but we expand on the search procedure described by Rudich et al. (2023), an extension of an idea from Coppé et al. (2024).

A path in both a restricted DD and a relaxed DD represents a sequence of decisions. In a restricted DD, each path from the root to a node in the DD represents a feasible partial solution to the problem being solved. A relaxed DD contains every feasible solution, partial or otherwise. Thus, each path in a restricted DD will also exist in the associated relaxed DD. Furthermore, DDs are deterministic, so each path in a DD maps to exactly one node. Consequently, any possible path (sequence of decisions) to a node $\overline{u}$ in a restricted DD $\overline{\mathcal{M}}$ will map to exactly one node $\underline{u}$ in the associated relaxed DD $\underline{\mathcal{M}}$.

Let the domain of a node u be the set of feasible arc labels on out-arcs of u. When generating $\overline{\mathcal{M}}$, each node in $\overline{\mathcal{M}}$ creates a child node on the next layer for every element in its domain, and then the new

Figure 3 **Embedded restricted decision diagram with width $\omega_s = 2$.**



(a) DD Post-Split

(b) Embedded Restriction

layer is trimmed down to a pre-chosen maximum width. The mapping from restricted DDs to relaxed DDs can be leveraged to improve the restricted DD as it is being generated. Let $d(\overline{u})$ be the domain of $\overline{u}$; set $d(\overline{u}) = d(\overline{u}) \cap d(\underline{u})$ before generating the child nodes of $\overline{u}$. This way, if an arc has been proven to be sub-optimal in $\underline{\mathcal{M}}$, it will not be created when generating $\overline{\mathcal{M}}$. This also allows for easy intensification of the search when combined with peel operations. If a node u has been peeled from $\underline{\mathcal{M}}$ into $\underline{\mathcal{U}}$, then $\overline{\mathcal{M}}$ will not include any solutions that pass through u. Similarly, if a restricted DD $\overline{\mathcal{U}}$ is generated that is embedded in $\underline{\mathcal{U}}$, it will only explore solutions that pass through u.

Without using this intersection operation as a way to trim the domain, $\overline{\mathcal{M}}$ will search the entire solution space that starts from the same root as $\underline{\mathcal{M}}$, even if the peeled DDs have already been fully explored and are known to be sub-optimal. Using this method, each restricted DD will only search the solutions encoded within the matching relaxed DD. This means that each restricted DD has a significantly improved chance of finding the best solution embedded in the relaxed DD it maps to because the solution space it must explore becomes smaller with each peeled node.

Figure 3 shows a restricted DD with width $\omega_s = 2$ embedded in the relaxed DD depicted in Figure 2b. The embedded restricted DD is indicated in purple; the black nodes are nodes that were generated during the search, but not expanded. Only the $\omega_s = 2$ best nodes, determined by $z_{\downarrow}(u)$, which is displayed as label on each node u, are selected in each layer. The shortest path in the embedded restricted DD, $(C \to B \to A)$, is a feasible solution and provides an upper bound of $\overline{z} = 11$ for our TD-TSP example.

3.4. Peel-and-Bound

The previous sections provide all the ingredients for the Peel-and-Bound algorithm originally proposed by Rudich et al. (2022) and later improved by Rudich et al. (2023). Here, we briefly describe this algorithm,

which is capable of handling TD-TSP problems like the one shown in our running example. In Section 5, we explain how to adapt the algorithm to DDs with arc bounds instead of exact values, enabling us to solve the ARP where computing the exact cost of an arc requires calling an expensive black-box function $\mathcal{B}$.

Let $\underline{\mathcal{M}}(u)$ be a DD with root node u, and let $\underline{\mathcal{M}}(r)$ be the entire initial relaxed DD starting at the (global) root node r. Place the entire DD into a queue Q such that $Q = \{\underline{\mathcal{M}}(r)\}$. Then, a DD $\underline{\mathcal{M}}(u)$ is selected from Q (for the first iteration $\underline{\mathcal{M}}(u) = \underline{\mathcal{M}}(r)$). From that DD, $\underline{\mathcal{M}}(u)$, a single exact node e is selected. A node e is *exact* if all partial paths from the root to e have the same set of feasible completions. Then e is peeled from $\underline{\mathcal{M}}(u)$, yielding a new sub-diagram $\mathcal{D}_e$ and the now strengthened diagram $\underline{\mathcal{M}}(u)$. Both DD selection and node selection are heuristic decisions described in detail by Rudich et al. (2023) (see Online Supplement B Section B.2 for a brief summary).

If the shortest path through the modified $\underline{\mathcal{M}}(u)$ is less than the best-known solution, $\underline{\mathcal{M}}(u)$ is put back into Q. After that, $\underline{\mathcal{M}}(u)$ is used as the basis for a heuristic search for a feasible solution using an embedded restricted DD $\overline{\mathcal{M}}(u)$, possibly leading to an improved upper bound $\overline{z}_{opt}$. If the embedded restricted DD is not exact (if some paths were not fully explored), the DD $\mathcal{D}_e$ is strengthened using node splits. Let $\underline{\mathcal{M}}(e)$ be the result; if the shortest path through the refined DD $\underline{\mathcal{M}}(e)$ is less than the best known solution, $\underline{\mathcal{M}}(e)$ is added to Q. The whole procedure is repeated until no DDs are left in the queue ($Q = \emptyset$). This implementation of Peel-and-Bound is formalized in Algorithm 2. Peel-and-Bound terminates when no DDs are remaining that could possibly contain a solution better than the best-known solution. Therefore, when the algorithm terminates, the best-known solution is known to be optimal. However, it performs a search for feasible solutions at every iteration and may find the optimal solution long before the optimality proof.

4. Relaxing the Black Box: DDs with Arc Bounds

This section introduces the concept of DDs with arc bounds, which makes it possible to solve black-box time-dependent routing problems such as the ARP. We then describe an approach for obtaining arc bounds by relaxing the inner trajectory optimization problem. Finally, we show how to construct an initial relaxed DD that can serve as a starting point for solving the ARP with Peel-and-Bound.

4.1. Decision Diagrams with Valid Arc Bounds

In the TD-TSP example from the previous section, the weights on the arcs are transition costs of going from one city to the next. These arc weights are not dependent on *all* preceding cities, and the transition costs are easy to calculate. However, in this paper we are handling a problem where calculating arc weights requires calling a black-box function $\mathcal{B}$ with non-negligible computation time. Furthermore, the exact arc weight for an arc a depends on the path to reach it. In principle, this dependency requires the evaluation of all possible paths from the root node r to a to find the optimal one.

Performing such an evaluation would result in a prohibitively high computational burden which makes it impractical to use existing DD approaches. A key contribution of this paper is to introduce DDs with

Algorithm 2: Peel-and-Bound (PnB) Algorithm (Rudich et al. 2023)

Input: The initial relaxed DD $\underline{\mathcal{M}}(r)$
Let $v^*(u)$ be the shortest path that passes through u (the lower bound encoded by only u)
$Q \leftarrow \{\underline{\mathcal{M}}(r)\}$
$z_{\text{opt}} \leftarrow \infty$ // Value of the best known solution
while $Q \neq \emptyset$ **do**
  $\mathcal{D} \leftarrow$selectDiagram(Q), $Q \leftarrow Q \backslash \{\mathcal{D}\}$
  $u \leftarrow$selectExactNode($\mathcal{D}$)
  $\mathcal{D}_u$
end
, $\mathcal{D}^* \leftarrow$ peel($\mathcal{D}$, u) // See Algorithm 1 **if** $v^*(\mathcal{D}^*) < z_{\text{opt}}$ **then**
  $Q \leftarrow Q \cup \{\mathcal{D}^*\}$
end
$\overline{\mathcal{M}} \leftarrow \overline{\mathcal{M}}(u)$ // Using Algorithm 3, with u as the root, also see Sec. 5.1
if $v^*(\overline{\mathcal{M}}) < z_{\text{opt}}$ **then**
  $z_{\text{opt}} \leftarrow v^*(\overline{\mathcal{M}})$
end
if $\overline{\mathcal{M}}$ is not exact **then**
  $\underline{\mathcal{M}} \leftarrow$ refine($\mathcal{D}_u$) // Using node splitting, see Sec 3.2.2
  if $v^*(\underline{\mathcal{M}}) < z_{\text{opt}}$ **then**
    $Q \leftarrow Q \cup \{\underline{\mathcal{M}}\}$
  end
end

return z_{opt}

valid arc bounds instead of exact arc weights. These are globally valid lower bounds on the true cost of the weights that are computed via relaxation of $\mathcal{B}$ when constructing an initial relaxed DD. They are used to prune sub-optimal paths and remain valid during the whole Peel-and-Bound algorithm, avoiding the need for extensive calls to $\mathcal{B}$ when refining the DD. After creating the initial relaxed DD, $\mathcal{B}$ only needs to be called when searching for, or evaluating, feasible solutions.

4.2. Relaxing the Black Box

In real-world problems that depend on a black-box function or an inner problem, it is often possible to define relaxed versions of such functions that provide bounds or heuristic values (Ehmke et al. 2016, Andersson et al. 2015). In the ARP, we can define a relaxed version of Eq. (3) that removes the cost of waiting:

$$f'_{\text{inner}}(a, a', \eta, \tau, t) = \Delta V + \frac{2\,\text{km/s}}{30\,\text{days}} \cdot t$$
$$\text{s.t.} \quad \tau \in [0, \tau_{\text{f}}],\ t \in [1, t_{\max}] \tag{5}$$
$$\text{where} \quad \Delta V = \|\Delta\vec{v}_1\|_2 + \|\Delta\vec{v}_2\|_2 \quad \text{and} \quad (\Delta\vec{v}_1, \Delta\vec{v}_2) = \text{Lambert}(a, a', \eta + \tau, t)$$

Since f'_{inner} removes the cost of waiting, we also relax the upper bound of τ to a parameter τ_{f}, denoting the final time at which departure remains feasible. We can now define the following relaxed variant of $\mathcal{B}$:

$$\mathcal{B}'(a, a', \eta, \tau_{\text{f}}) \colon A \times A \times \mathbb{R}^+ \to [0, \tau_{\text{f}}] \times [1, t_{\max}] \times \mathbb{R}^+ \tag{6}$$

which returns three values (τ, t, z), where τ and t are the values that minimize f'_{inner} given a, a', η and τ_{f}, and z is its minimal value. This is only a relaxation in the sense that it removes part of the cost from the objective function, it does not change the method used to solve it. The procedure for solving $\mathcal{B}'$ is exactly the same as for solving $\mathcal{B}$. The strength of the relaxation depends on η and τ_{f}, which depend on earliest start time and latest start time.

4.3. Creating a Relaxed DD with Strong Valid Arc Bounds

The Peel-and-Bound algorithm starts from an initial relaxed DD. In this section, we explain how to initialize a relaxed DD with strong arc bounds. Recall that calculating arc bounds is expensive because it requires evaluating the black-box function $\mathcal{B}$. In the ARP, this means solving the inner optimization problem. The overall algorithm we propose is designed to minimize the number of evaluations of $\mathcal{B}$. In particular, we intentionally allocate substantial computational effort to the initial DD construction, with the goal of not needing to evaluate $\mathcal{B}$ after the initial relaxed DD is constructed, except when evaluating feasible solutions.

After the initial construction, when using constraint propagation, arcs are filtered and removed but never added. When splits and peels are performed, the number of arcs in the DD may grow, but only when the out-arcs of a node are copied. When this occurs, the asteroid that an arc is traveling from and the asteroid that the arc is traveling to remain unchanged. Therefore, when splitting or peeling, a valid bound on the cost of an arc is also a valid bound on its copy. While it may be desirable to recalculate the bound on an arc after a split to get a tighter bound, peels and splits can be carried out as many times as memory limitations allow without the need to re-evaluate $\mathcal{B}$.

4.3.1. Structure of the relaxed DD We begin by constructing a relaxed DD $\mathcal{M}$ exactly as shown in Fig. 1a. We start with a root node at layer 0, a terminal node at layer $n+1$, and n nodes on each layer with index i, where $1 \leq i \leq n$. Each node u in a layer i is labeled $l(u)$ unique to that layer. Node labels are conceptually the same as arc labels. Each node and each arc has exactly one label, always representing either *Earth* or an asteroid. An arc represents the transition from one node to another, so in the ARP, an arc represents a spacecraft transfer. The arc label is the asteroid being transferred to. A node represents a state and in the ARP, part of that state is the location of the spacecraft (either *Earth* or an asteroid). Thus, the node label is either *Earth* for the root or an asteroid for the other non-terminal nodes. The terminal node is a dummy node. Then, each node v in layer $i-1$ is the origin for an arc ending at u, as long as v has a different label. An arc with a null label is added from each node in layer n to the terminal. The result is a relaxed DD that encodes every feasible sequence of asteroids as a path from the root to the terminal.

4.3.2. Initializing the Arc Bounds Now we weight the DD for the arcs going from layer 0 to layer 1. A valid weight for an arc must be a valid lower bound on the cost of making the transfer represented by that arc. So a valid weight on arc a_{uv} going from node u to v with labels $l(u) = \textit{Earth}$ and $l(v)$ is given by the z value returned by $\mathcal{B}(\textit{Earth}, l(v), \eta = 0)$, the smallest possible cost of going from Earth to $l(v)$.

The arcs going from layers i to $i+1$ for $i \in \{1, n-1\}$ remain to be weighted. We begin by finding a valid bound on the transfer between each ordered pair of asteroids $a, a' \in A$, with $a \neq a'$. We can use the no-wait $\mathcal{B}'$ (Eq. 6) for that purpose by noticing that the maximum time that any solution requires is trivially bounded by the sum of the upper bounds on the waiting times and travel times of the trajectories, i.e., $n(\tau_{\max} + t_{\max})$. Therefore, the latest start time for the final transfer is bounded by $n(\tau_{\max} + t_{\max}) - t_{\max}$. That means that the transfer from an asteroid a to an asteroid a' can wait for $\tau \in [est(a), n(\tau_{\max} + t_{\max}) - t_{\max}]$, where $est(a)$ is the earliest start time at a. Initially, $est(a) = \tau^* + t^*$, where τ^* and t^* are the values returned by $\mathcal{B}(Earth, a, \eta = 0)$ as described above. Generally, the est of a node is the shortest path to that node where the only cost is time. This is explored and formalized in Online Supplement C. We can now define:

$$z_{\min}(a, a') := z \quad \text{where} \quad (\tau, t, z) = \mathcal{B}'(a, a', \eta = est(a), \tau_{\mathrm{f}} = n(\tau_{\max} + t_{\max}) - t_{\max}) \tag{7}$$

Because $\mathcal{B}'$ does not penalize waiting time, the waiting time τ returned by $\mathcal{B}'$ above gives an optimal η for minimizing the actual cost of the transfer using $\mathcal{B}$ (Eq. 1). Thus, the cost $z_{\min}(a, a')$ is the smallest possible cost of the transfer from a to a', and thus a lower bound of the z value returned by $\mathcal{B}(a, a', \eta = est(a) + \tau)$. Computing $z_{\min}(a_i, a_j)$ for all pairs a_i, a_j, where $i, j \in [1 \,..\, n]$ and $i \neq j$ requires $n^2 - n$ calls to $\mathcal{B}'$.

We add the pre-calculated $z_{\min}(a, a')$ values as arc weights. A valid weight for an arc a_{uv} going from node u with label $l(u)$ to v with label $l(v)$, is simply $z_{\min}(l(u), l(v))$, which is the best possible transfer between $l(u)$ and $l(v)$ when ignoring waiting time at $l(u)$. The resulting relaxed DD contains valid weights such that the sum of the costs of the arcs on any path from the root to the terminal provides a lower bound on the true cost of the permutation represented by the labels of the nodes on that path. However, this lower bound is quite weak, so we perform a second phase to strengthen the arc bounds.

4.3.3. Strengthening the Arc Bounds We will use a heuristic solution $\overline{z}$ to strengthen the arc weights in this phase. Many methods are available in the literature for finding a strong heuristic solution in trajectory optimization problems such as the ARP (Simões et al. 2017, Hennes et al. 2016). For simplicity, our implementation uses an Euclidean distance heuristic to find the initial $\overline{z}$. It picks asteroids one by one, always moving to the closest unvisited asteroid as measured by Euclidean distance at the time of arrival to the last visited asteroid. This simple heuristic is exceptionally effective, and it has no trouble rapidly finding strong feasible solutions. In Sec. 5.1, we describe our method for finding feasible solutions in more detail.

Before we can use $\overline{z}$, we need to store some information on the nodes. We begin by recursively updating each node u with the length of the shortest path from the root to u, denoted by $z_{\downarrow}(u)$, as well as the length of the shortest path from u to the terminal, denoted by $z_{\uparrow}(u)$ (Cire and van Hoeve 2013). As stated in Lemma 2, if $z_{\downarrow}(u) + z_{\uparrow}(u) > \overline{z}$, then the optimal solution cannot pass through u, and we can remove u from the DD because the shortest path that passes through u has a higher cost than a known feasible solution.

Now consider an arc a_{uv} connecting nodes u to node v with weight $w(a_{uv})$. Similarly, $z_{\downarrow}(u) + w(a_{uv}) + z_{\uparrow}(v)$ is a valid lower bound on the cost of any path that can pass through a_{uv}. This is because any shortest

path that includes a given arc can be decomposed into three segments: the shortest path from the root to the arc's origin, the arc itself, and the shortest path from the arc's destination to the terminal. As before, we know that if $z_{\downarrow}(u) + w(a_{uv}) + z_{\uparrow}(v) > \overline{z}$, then the optimal solution cannot pass through a_{uv}, and thus a_{uv} can be removed from the graph. We will leverage this constraint to calculate new stronger arc weights.

We do so by observing two important aspects. First, the strength of the black box relaxation $\mathcal{B}'$ can be increased by decreasing τ_{f}. Second, while $\mathcal{B}'$ ignores waiting costs, the true cost of $w(a_{uv})$ includes the cost of wait time as a component. Let τ_a be the wait time used by a_{uv}, and recall that the cost of time in the *inner* objective function (Eq. 3) is $\frac{2}{30}$. Now, we can conclude that if a_{uv} that will not be filtered by objective function value needs to satisfy the condition $z_{\downarrow}(u) + \frac{2}{30}\tau_a + z_{\uparrow}(v) \leq \overline{z}$. This is because if τ_a is large enough to cause that constraint to be violated, then $w(a_{uv})$ will also be large enough to cause its constraint to be violated because $w(a_{uv})$ is a sum of components that include $\frac{2}{30}\tau_a$. This yields the constraint that: $\tau_a \leq \frac{30}{2}(\overline{z} - z_{\downarrow}(u) - z_{\uparrow}(v))$. The right-hand-side of this constraint is a constant because $z_{\downarrow}(u)$ and $z_{\uparrow}(v)$ are known. This means that we can use this bound for τ_a to define a stronger bound per layer:

$$z_{\min}(u, v, i) := z \text{ where } (\tau, t, z) = \mathcal{B}'(l(u), l(v), \eta = est(u), \tau_{\mathrm{f}}) \text{ and } \tau_{\mathrm{f}} = \tfrac{30}{2}(\overline{z} - z_{\downarrow}(u) - z_{\uparrow}(v)) \quad (8)$$

Starting with $i = 1$, we calculate a new bound for every arc on a layer ($z_{\min}(u, v, i)$), then we update the values of $z_{\downarrow}(v)$ for each v (in other words, each node on layer $i + 1$), then we set $i \leftarrow i + 1$, and repeat until $i = n + 1$. Therefore, we do not update the arcs with a weight of 0 that go to the terminal. After this process is done, the $z_{\uparrow}$ values of the nodes need to be updated as they depend on the arcs' weights. This phase requires a total of $n^3 - 2n^2 + n$ evaluations of $\mathcal{B}'$, but results in strong enough initial bounds that, after this initial setup, our algorithm for the ARP only needs to call the black-box function $\mathcal{B}$ to evaluate the true cost of partial solutions during the search for feasible solutions based on embedded restricted DDs (see Sec. 5.1); it does not need to evaluate $\mathcal{B}$ or $\mathcal{B}'$ to bound.

The weights improve based on the $z_{\downarrow}$ and $z_{\uparrow}$ values, but $z_{\downarrow}$ and $z_{\uparrow}$ are updated based on the new arc weight. It is possible to repeat the above steps until the values converge. However, each additional update requires significant computation and, in practice, yields minuscule changes. We found that this process rarely improves the bound enough to be useful, so we did not include it and just perform a single pass.

5. Peel-and-Bound for DDs with Arc Bounds

The original Peel-and-Bound algorithm (Rudich et al. 2022, 2023) operates on DDs with exact arc weights. To extend it to DDs with relaxed arc bounds and expensive black-box transition costs, we adapt several aspects of the algorithm. In particular, we modify the search for feasible solutions in embedded restricted DDs, and the use of these solutions to filter relaxed DDs. We also adjust the overall Peel-and-Bound algorithm. Finally, we discuss the algorithm's dependency on the quality of the inner optimizer solving $\mathcal{B}$.

5.1. Searching for Feasible Solutions in a DD with Arc Bounds

When handling a relaxed DD with arc bounds, the search for feasible solutions using embedded restricted DDs requires computing the exact weight of each non-exact arc in the relaxed DD that becomes part of the embedded restricted DD. For the embedded restricted DDs to be generated quickly, it is critical to limit the number of calls to $\mathcal{B}$. Let ω_s be the maximum width allotted to the search for feasible solutions in the embedded restricted DD. Evaluating a single feasible solution to the ARP requires $n-1$ calls to $\mathcal{B}$. We aim to perform a heuristic search that requires at most $\omega_s(n-1)$ calls to $\mathcal{B}$. We achieve this by exploiting the fact that each node u in a relaxed DD can be labelled with the length of the shortest path from u to the terminal, which we have already denoted as $z_{\uparrow}(u)$ in this paper. This provides a simple but powerful heuristic for deciding which nodes to explore. This heuristic is directly inspired by, but substantially different from, the rough bounding proposed in Gillard et al. (2021).

When processing node $\overline{u}$ in a partially constructed restricted DD $\overline{\mathcal{M}}$, the typical method to determine which children of $\overline{u}$ to explore would be to generate all of them and then repeatedly remove the node $\overline{v}$ in the layer being constructed with the highest $z_{\downarrow}(v)$ until the width of the layer is ω_s. However, for the ARP, each creation of a child requires a call to $\mathcal{B}$, which quickly becomes intractable.

Let $\overline{V}$ be the set of potential child nodes that could be created during the procedure. Instead of generating every possible child node $\overline{v} \in \overline{V}$, we calculate a bound on the true cost of a solution passing through each $\overline{v}$. Then, we generate the ω_s child nodes in $\overline{V}$ with the lowest bounds. For a potential node $\overline{v}$ with parent $\overline{u}$, associated nodes in the relaxed DD $\underline{v}$ and $\underline{u}$ respectively, and arc $a_{\underline{uv}}$ with weight $w(a_{\underline{uv}})$, the bound is:

$$b(\overline{v}) = z_{\downarrow}(\overline{u}) + w(a_{\underline{uv}}) + z_{\uparrow}(\underline{v}) \tag{9}$$

For any node $\overline{u}$ in a restricted DD $\overline{\mathcal{M}}$, let $\underline{u}$ be the associated node in $\underline{\mathcal{M}}$. Let $\underline{r}$ be the root of an existing relaxed DD $\underline{\mathcal{M}}$. The procedure begins by creating $\overline{r}$ without creating its children. Then it iterates over the nodes in $\underline{\mathcal{M}}$ associated with the potential children of $\overline{r}$, and creates the ω_s best candidate children of $\overline{r}$, where candidacy is determined by the value returned by Eq. (9). Then this is repeated for each layer of the restricted DD. This procedure is formalized in Algorithm 3. Observe that since $\mathcal{B}$ is called at most ω_s times in each of the $n-1$ layers other than 0, the total number of calls is indeed restricted to $\omega_s(n-1)$ for each call of the heuristic search.

5.2. Using Feasible Solutions for Strengthening a Relaxed DD with Arc Bounds

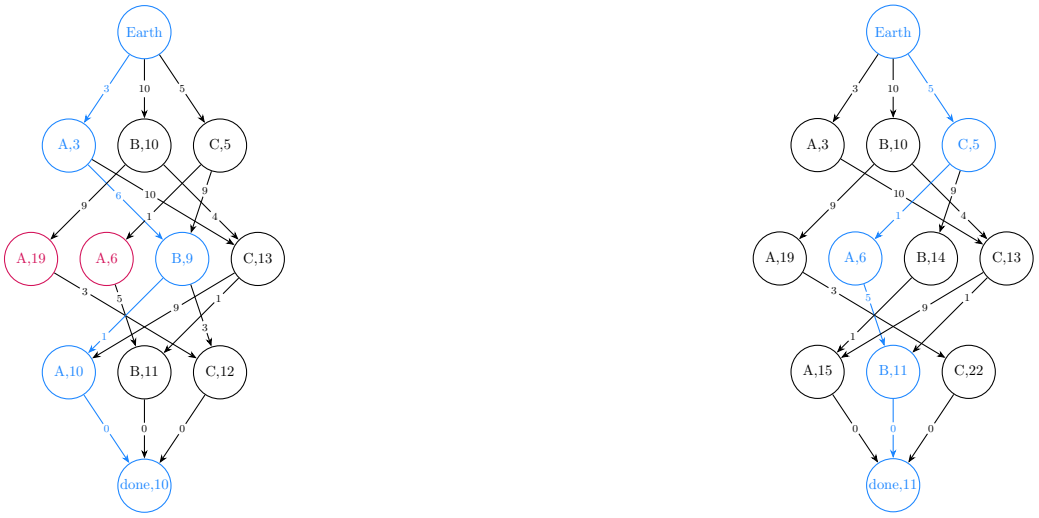
In a relaxed DD with exact arc values representing the cost function, a feasible path from the root to the terminal has a length equal to its exact solution value. When arc weights represent bounds rather than exact values, the path length itself is only a bound. However, if we know the exact value of a path (e.g., from a heuristic search in an embedded restricted DD), we can remove that path from the relaxed DD by applying the refinement techniques reviewed in Section 3.2. Notably, removing a path not only excludes it, but also strengthens the relaxed DD overall by enabling node and arc pruning through constraint propagation.

Algorithm 3: Heuristic Search Procedure

Input: maximum width ω_s; relaxed DD $\underline{\mathcal{M}}$ with root r on layer 0
Let $w(a_{xy})$ be the weight of the arc connecting x to y.
Let $\overline{\mathcal{M}}$ be a restricted DD with the same number of layers as $\underline{\mathcal{M}}$, where each layer is initialized to $\emptyset$ except layer 0 which is initialized to $\{r\}$.
For any node $\overline{u} \in \overline{\mathcal{M}}$, let $\underline{u}$ be the associated node in $\underline{\mathcal{M}}$.
foreach layer of nodes L in $\overline{\mathcal{M}}$ **do**
    $q \leftarrow \emptyset$, $b \leftarrow \emptyset$ // q: empty list of nodes; b: empty mapping of nodes to values.
    foreach node $\overline{u} \in L$ **do**
        Let the domain of $\overline{u}$ be $d(\overline{u})$, initialized to be the set of node labels on the path from r to $\overline{u}$
        Let $d(\underline{u})$ be the set of node labels used by children of $\underline{u}$
        $d(\overline{u}) \leftarrow d(\overline{u}) \cap d(\underline{u})$
        foreach label $l \in d(\overline{u})$ **do**
            Create a new node $\overline{v}$ as a child of $\overline{u}$ with label l, but do not weight arc $a_{\overline{uv}}$
            $b(\overline{v}) \leftarrow z_{\downarrow}(\overline{u}) + w_{a_{\underline{uv}}} + z_{\uparrow}(\underline{v})$
            $q \leftarrow q \cup \{\overline{v}\}$
        end
    end
    Sort the elements $\overline{v} \in q$ from least to greatest by their bounds $b(\overline{v})$
    while $|q| > \omega_s$ **do**
        Remove the last element of q (the one with the largest bound) from both q and $\overline{\mathcal{M}}$
    end
    foreach node $\overline{v} \in q$ **do**
        $\tau^*, t^*, z^* \leftarrow \mathcal{B}(l(\overline{u}), l(\overline{v}), est(\overline{u}))$
        $w(a_{\overline{uv}}) \leftarrow z^*$
        $est(\overline{v}) \leftarrow est(\overline{u}) + \tau^* + t^*$
    end
end
return $\overline{\mathcal{M}}$

This can be illustrated using the example from Section 3. Figure 4 shows the relaxed DD that was obtained by node splitting (see Section 3.2.2). The shortest path in that DD is $(A \to B \to A)$, which is infeasible and provides a lower bound of 10. Suppose a feasible solution search finds that the exact length of $(A \to B \to C)$ is 13, we may now remove this path and all other partial paths with lengths ≥ 13 from the relaxed DD. Since $(A \to B \to C)$ is the only feasible path beginning with $A \to B$, we can remove the arc from the A node on the second layer to the B node on the third layer, and similarly, the arc from the B node on the third layer to the C node on the fourth layer. These removals exclude $(A \to B \to C)$ without eliminating any other potentially optimal path, because only suboptimal solutions pass through those arcs. The new optimal (and feasible) path in the resulting relaxed DD is $(C \to A \to B)$ with a value of 11. If at least one arc in this path has a non-exact cost, the value 11 serves as a dual bound for the true optimal objective, and evaluating its exact value would require calling $\mathcal{B}$ for each arc with non-exact costs. Finally, since the removed feasible solution has an exact value of 13, objective-based constraint propagation can remove all nodes not on the optimal path from the DD on the right-hand side of the figure. In general, any node or arc

Figure 4 **Eliminating a feasible path with known exact cost from a relaxed DD.**



(a) DD Post-Split

(b) DD Post-Propagation

that lies exclusively on paths with values no better than the incumbent can be removed without eliminating potentially optimal solutions.

5.3. Adapting the Peel-And-Bound Algorithm

The previous sections described how the details of the general Peel-and-Bound algorithm (see Section 3) can be adapted to solve the ARP and other black-box time-dependent routing problems represented by DDs with arc bounds. In this section, we discuss further modifications to Peel-and-Bound (Algorithm 2) that are necessary or advantageous for addressing these problems.

The first modification to Algorithm 2 affects the order of the main steps applied to each relaxed DD $\underline{\mathcal{M}}(u)$ that was selected from the queue Q. Since feasible solutions can be very helpful for pruning DDs with arc bounds, the call of the search for feasible solutions based on embedded restricted DDs (Algorithm 3) is applied first, before the peeling step. For the peeling step, a single exact node e from $\underline{\mathcal{M}}(u)$ is selected. An exact node for the ARP is simply a node u whose $z_{\downarrow}(u)$ is an exact value and not a bound. In the initial DD, all nodes on layer 1 are exact because there is only one path from r (Earth) to those nodes, and the true cost of those arcs is known. The selection of a DD and an exact node involves heuristic decisions aimed at balancing exploration of promising regions with the need to control computational effort. In particular, we prioritize DDs and nodes that are most likely to improve the relaxed bound on the problem. Further implementation details are provided in Online Supplement B.

Finally, a necessary change required to use Peel-and-Bound for the ARP is deciding when to add a DD to the queue. In the original presentation of Peel-and-Bound, when the shortest path through a relaxed DD is a feasible solution, that DD is not placed back into the processing queue because the best-known feasible

solution encoded in that DD is known. However, as explained above, the shortest path through a relaxed DD for the ARP being a feasible solution does not prove that path to be the optimal path in that DD because the arc weights are not exact. Thus, to refrain from adding a DD back into the queue for the ARP, there must be a known feasible solution with a cost that is not larger than the length of the shortest path in that DD.

5.4. Limitations of the Inner Optimizer

For some inner problems, it may be possible to ensure that the inner optimizer returns a global optimum. In the ARP, however, the inner problem is non-convex and non-linear, making it impossible, in general, to prove that a local optimum is also a global optimum. López-Ibáñez et al. (2022) use a deterministic optimizer so that the same permutation will yield the same local optima for the inner optimization problem. Thus, there is a globally optimal permutation independently if the inner problem returns a local optimum in some cases. In $\mathcal{B}$, the search space is small enough that local optima are also likely to be global optima. When using $\mathcal{B}'$, the inner optimizer searches within a superset of the search space induced by $\mathcal{B}$, potentially finding a local optimum that is worse than the one returned by $\mathcal{B}$. In such cases, the outer optimization might be misled about which permutation is optimal. However, the permutation found is still a solution (possibly suboptimal) to the original ARP, because feasible solutions are always evaluated using $\mathcal{B}$, not $\mathcal{B}'$.

More concretely, the inner optimization that computes $\mathcal{B}$ in López-Ibáñez et al. (2022) consists of a single deterministic run of SLSQP, as explained in Section 2.2. A single run of SLSQP almost always returns a global optimum when $\tau_f \leq \tau_{max}$ as set in the original paper. However, SLSQP may return a locally optimal solution when $\tau_f \gg \tau_{max}$, as we set it here when computing $\mathcal{B}'$ (Eq. 6). That is, $\mathcal{B}'(a, a', \eta = 0, \tau_f)$ may return (τ', t', z') as optimal yet $\mathcal{B}'(a, a', \eta = \tau', \tau_f)$ may return a better solution even though the time interval $[\tau', \tau_f]$ is contained within $[0, \tau_f]$. This problem can be alleviated by performing multiple restarts of SLSQP for each evaluation of $\mathcal{B}'$. In other words, when optimizing the relaxed inner problem (Eq. 5), we can partition the range $[\eta, \eta + \tau_f]$ into disjoint intervals, run SLSQP for each interval, and keep the best solution. We report experiments with different numbers of restarts that show their effect on runtime and solution quality.

Although SLSQP could be replaced with a more effective optimizer, we decided to keep SLSQP as the inner optimizer to compare our results on the ARP with those reported by López-Ibáñez et al. (2022). The following experiments (Section 6) show that increasing the quality of the inner optimizer does lead to better solutions, but not always, and not by much, so it is likely that we are already finding the optimal solutions for several of the ARP instances.

6. Experimental Results

An ARP instance with n asteroids is constructed by randomly selecting n asteroids from a database containing 83,453 asteroids. The selection process is controlled by a *seed* value, ensuring that specific instances can be consistently replicated. López-Ibáñez et al. (2022) test their algorithms with $n \in \{10, 15, 20, 25, 30\}$ and $seed \in \{42, 73\}$. We ran experiments with the same n values but added three randomly chosen seeds:

8, 22, and 59. The experiments were performed on a computer with an AMD Rome 7532 at 2.40 GHz and 64Gb RAM. Our code and raw experimental data are available Rudich et al. (2024).

6.1. Note on Optimality

As discussed in Section 5.4, changing the quality of the inner optimizer may impact the quality of solutions returned by our framework. In this section, we report different optimal solutions using different settings for several problems. These different solutions result from the inner optimizer not returning a global optimum, and thus are intentional and do not represent a mistake in the implementation. The parameter *multi* is the number of restarts used by the inner solver, which increases the probability of returning a global optimum.

6.2. Initial Experiment: Determining Best Settings

In our first test, we sought to determine which peel setting would be the most effective, which DD-widths ω would be the most effective, and whether it is worth ordering the processing queue to prioritize smaller sub-problems (depth-first-search). The value of ω used can greatly impact the solving time because relaxed DDs with larger widths generally yield better bounds but also take longer to compute. The best value for ω is the one that balances this quality/time trade-off. We tested $n = 15$ with all 5 seeds, 2 days of runtime, and no SLSQP restarts ($multi = 1$ means one SLSQP run for solving the inner optimization problem).

We tested two peel settings: maximal and last exact node. Maximal peels the node in the shortest path through the DD on the second layer. Last exact node peels the node in the shortest path with at least one child that is not exact. See Online Supplement B Section B.2. Fig. 5 summarizes these results, and the raw data is available in Online Supplement D Table EC.2. Although there is no clear winner, maximal generally performs better and performs only slightly worse when it does not. Consequently, we will use the maximal setting in all future experiments. The effectiveness of the depth-first-search (dfs) queue ordering

Figure 5 **ARP instances with $n = 15$, a 2 day runtime, an embedded search width of 100, and *multi* $= 1$. Lines show the mean gap (or mean time) over the ARP instances (Online Supplement D Table EC.2)**

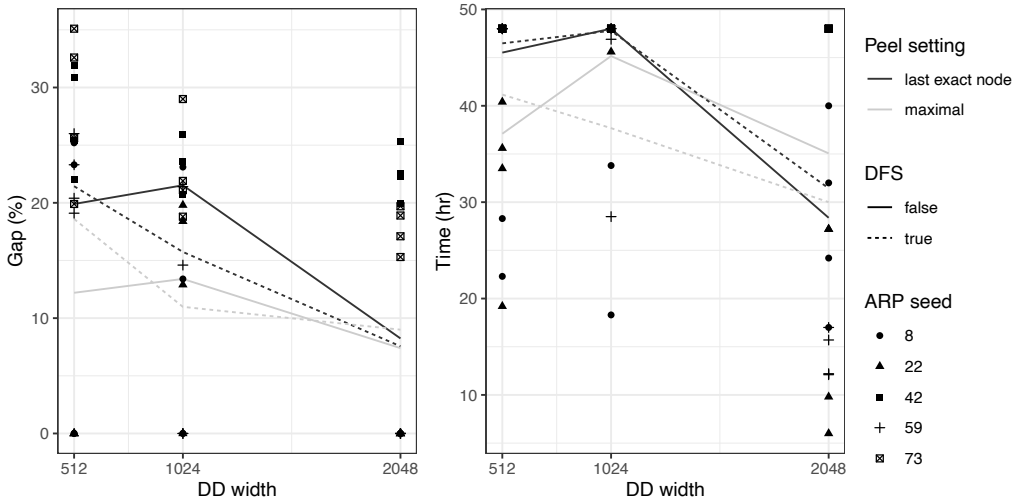


Table 1 Summary data for $n = 15$ and $n = 20$ (7-day runtime)*

n	DD width	*multi* = 1		*multi* = 2		*multi* = 3	
		gap (%)	time (hr)	gap (%)	time (hr)	gap (%)	time (hr)
15	256	5.1	97.2	11.7	151.0	18.2	168.0
	512	2.8	85.8	11.2	140.7	15.6	161.5
	2048	0	28.5	7.2	110.2	12.4	138.0
20	256	29.3	–	31.9	–	37.6	–
	512	27.7	–	31.2	–	37.0	–
	2048	27.0	–	30.2	–	32.5	–

* All the gap and time values in this table are averages. Timeouts included as 168h (7 days).

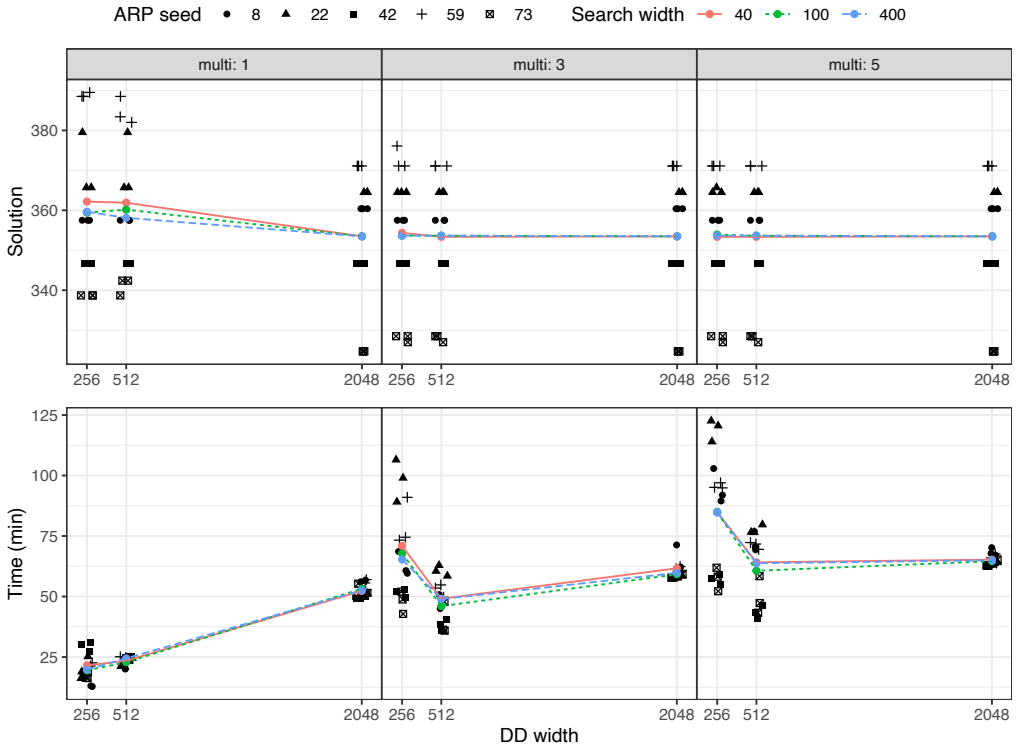
appears random based on summary data. The raw data suggests the success of dfs mainly depends on whether the best solution is found in an early branch. It also suggests that dfs is slightly worse for the *maximal* peel setting and both $\omega = 512$ and $\omega = 2048$, as also shown in the left plot of Fig. 5. We disabled dfs ordering in the remaining experiments, opting for the default ordering that processes the DD with the weakest bound, likely reducing the optimality gap in unresolved problems. Regarding relaxed DD-width, we tested 512, 1024, and 2048. While 2048 showed the greatest success in closing problems and narrowing the optimality gap, 512 resolved specific settings significantly faster. Moving forward, we will retest 512 and 2048, and introduce a new width of 256. We do not include tests with widths larger than 2048 because in our experience with DDs, the slowdown in construction time from increasing the width starts to become substantial around 2048. This happens because the number of arcs at each layer is quadratic in ω, and small increases in the width beyond 2048 can lead to enormous increases in the construction time.

6.3. Second Experiment: Test of Smaller Instances

Our first round of experiments gave us a rough idea of what settings might be successful. Our second round of experiments makes up the bulk of our tests. As discussed in the previous paragraph we test relaxed DD-widths of 256, 512, and 2048. For the embedded search widths (ω_s from Algorithm 3) we tested 40, 100, and 400, where in Sec. 6.2 we just used 100. In the DD literature (Bergman et al. 2016), the size of the search for feasible solutions is often as big or bigger than the size of the relaxed DD, because it is relatively computationally cheap. However, for the ARP, the search for feasible solutions is computationally expensive because of the inner-optimization that requires evaluating the black-box function $\mathcal{B}$. This suggested to us that larger embedded search widths are unlikely to be worth it. However, the following results indicate that might not be true. We also tested *multi* $\in \{1, 3, 5\}$. Summary data is shown in Table 1. Fig. 6, 7 and 8 visualize the results of these experiments for $n = 10$, $n = 15$ and $n = 20$, respectively. The raw data is in Online Supplement D, Tables EC.1, EC.3 and EC.5, respectively.

The largest search settings, i.e., an embedded search width of 400 and a relaxed DD-width of 2048, clearly dominate the results, with tighter optimality gaps and more problems solved. For the remainder of the discussion, we will be referring primarily to those results. Recall from Section 5.4 that different *multi*

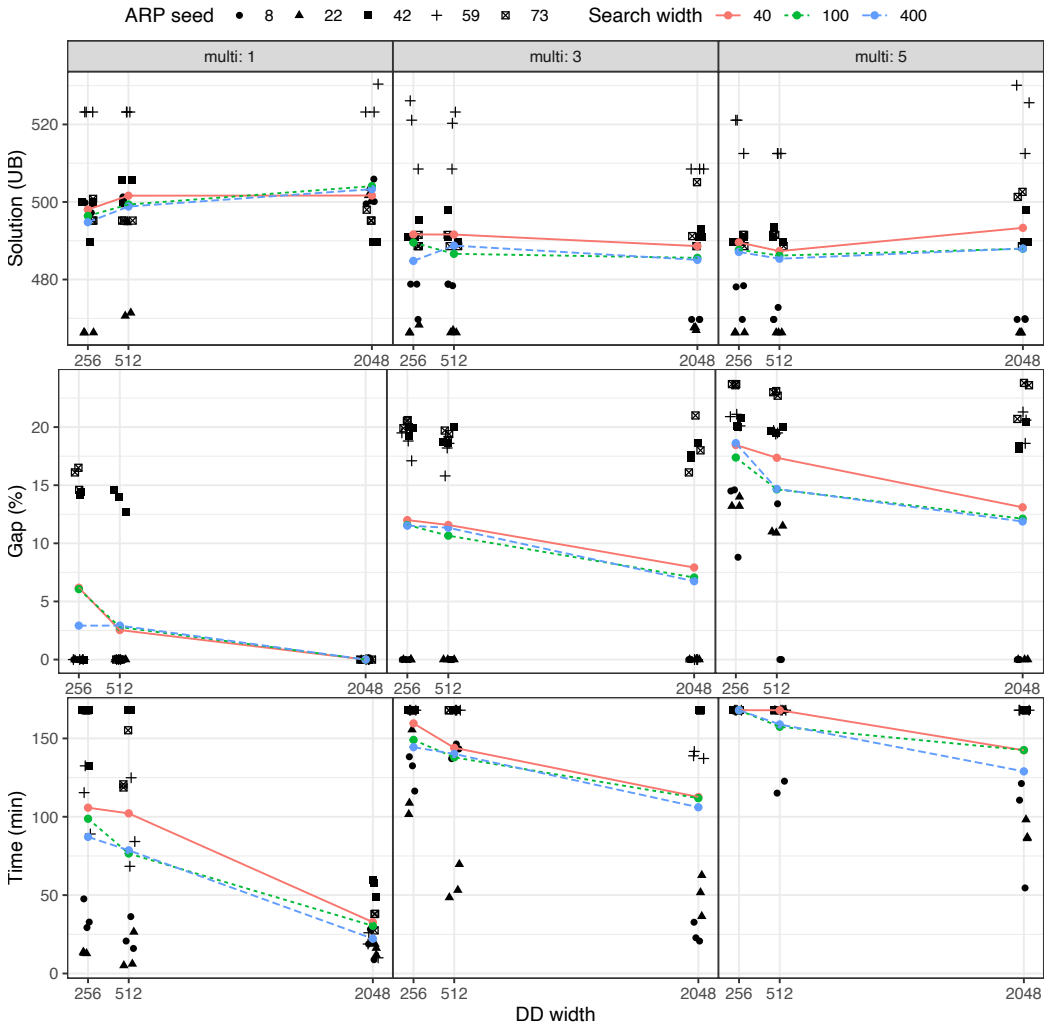
Figure 6 **ARP instances with $n = 10$. Lines: mean value over ARP instances (Online Supplement D Table EC.1)**



values can have different optimal solutions because the inner optimizer may return a local optimum instead of the global optimum. When $n = 10$ (Online Supplement D Table EC.1 and Fig. 6) and *multi* $= 1$, all of the problems are solved to optimality in under 1 hour, and when *multi* $= 5$, they are solved to optimality in under 2 hours. In some cases the largest search settings alleviate some of the negative effects of *multi* $= 1$ (in Online Supplement D Table EC.1, seed 22, DD width 512, compare search width 40 to search width 100). This is likely because a larger search provides more opportunities to find improved solutions that will later be trimmed as a result of using a low *multi*. For the the largest search settings, the effect of *multi* is only to increase the runtime, however settings with smaller DD widths make it clear that *multi* can have a strong impact on the underlying arc calculations. For example, seed 59, with search width 40, and relaxed DD-width 256, has a different exact solution with *multi* $= 1$ than it does with *multi* $= 5$; the cost drops from 388.5 to 371.1. Recall that because the inner problem in the ARP is nonlinear and nonconvex, it is not possible to prove overall optimality in general. In lieu of that, *multi* can be arbitrarily increased to increase confidence in the solutions. If *multi* is increased, and the algorithm returns the same permutation as it did for a lower *multi* value, that provides evidence that the optimal solution is stable.

We now focus on the results obtained with an embedded search width of 400 and a relaxed DD-width of 2048 on instances of size $n \in \{15, 20\}$ (Online Supplement D Tables EC.3 and EC.5) with a maximum runtime of 7 days. With *multi* $= 1$, three of the problems were solved in under a day, with the other two

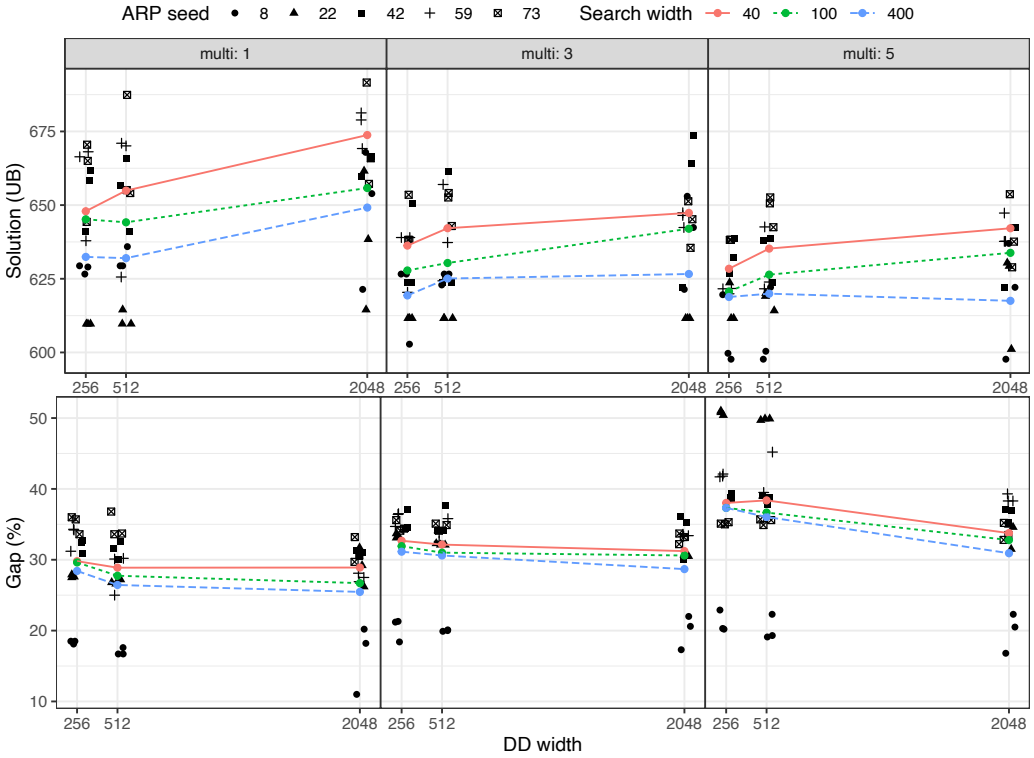
Figure 7 **ARP instances with $n = 15$ and a 7 day runtime (Online Supplement D Table EC.3)**



taking around two days. However, many of the solutions are improved with a higher *multi* (see the top row of Figs. 7 and 8). For $n = 15$, the setting *multi* $= 1$ is unlikely to yield optimal solutions. With *multi* $= 5$, only seeds 8 and 22 were solved exactly, but they yielded much better solutions. In seed 22 for example, the best solution drops from 501.7 to 466.3. With $n = 20$, no problems were solved to optimality within 7 days.

The only comparison to the literature we can make is with López-Ibáñez et al. (2022), who compare variants of Combinatorial Efficient Global Optimization (CEGO) (Zaefferer et al. 2014), a Bayesian optimizer for black-box combinatorial problems, and Unbalanced Mallows Model (UMM) (Irurozki and López-Ibáñez 2021), an estimation-of-distribution algorithm based on probabilistic Mallows models. CEGO builds a surrogate model of the objective function based on a distance metric and optimizes the surrogate model to find the next point to evaluate on the actual objective function. Since building a surrogate model is computationally expensive, UMM estimates a probabilistic distribution over the permutation space that gives more probability to solutions with higher objective function value. UMM then uses this distribution to sample new solutions and update the estimation. López-Ibáñez et al. (2022) only used seeds 42 and 73, so we

Figure 8 **ARP instances with $n = 20$ and a 7 day runtime (Online Supplement D Table EC.5)**



re-ran their best settings of CEGO and UMM on all instances considered here. We use the same termination criteria as in the original paper, which produce an average runtime of 9.25 and 0.26 CPU-hours for CEGO and UMM, respectively. However, because they treat the inner problem as completely black-box, whereas we take advantage of known properties of the inner problem, any runtime comparisons are meaningless. Thus, we focus on comparing solution quality.

These results are shown in Table 2. The solution obtained by Peel-and-Bound (PnB)'s preferred setting ($\omega_s = 400$, $\omega = 2048$, *multi*$= 5$) is the same as the best previously found for $n = 10$, and better than those previously reported for $n = 15$ and $n = 20$. In some cases, the gap is quite large, for example with $n = 20$, seed$= 42$, our solution is 12% better. We also report the best observed solution from any setting, and for 13 of the 15 instances the preferred setting found the same solution as the best solution found by any setting. All of these results use $\mathcal{B}$ to evaluate the cost of a permutation, and thus they are directly comparable.

6.4. Final Experiment: Test of Larger Instances

In our final experiments we consider the larger instances with $n \in \{25, 30\}$. We ran these experiments on a slightly less powerful computer, Intel E5-2650v4 Broadwell 2.2GHz CPU with 64Gb RAM. We use a relaxed DD-width of 2048, and we test larger embedded search sizes than before: 400, 1024, and 2048. We limit the runtime to 3 days to show that the algorithm produces good solutions with a time limit shorter than 7 days.The results (Online Supplement D Tables EC.4 and EC.6) suggest that increasing the width of the

Table 2 Best solutions for $n \in \{10, 15, 20\}$

n	seed	López-Ibáñez et al. (2022)*		CEGO*		UMM*		PnB: Preferred Settings**		PnB: Best Found***	
		Average	Best	Average	Best	Average	Best	Value	Optimality Gap (%)	Value	Optimality Gap (%)
10	8	-	-	383.9	360.4	402.8	371.4	360.4	0.0	**357.5**	0.0
	22	-	-	396.6	365.7	408.5	385.8	364.5	0.0	364.5	0.0
	42	374.9	346.7	386.8	367.2	388.8	382.2	346.7	0.0	346.7	0.0
	59	-	-	399.8	381.1	413.2	383.5	371.1	0.0	371.1	0.0
	73	355.9	324.7	354.8	330.9	361.4	337.2	324.7	0.0	324.7	0.0
15	8	-	-	587.5	538.2	640.0	596.3	469.7	0.0	469.7	0.0
	22	-	-	606.6	538.6	634.0	566.9	466.3	0.0	466.3	0.0
	42	497.2	490.9	508.2	508.2	508.2	508.2	489.7	18.1	489.7	0.0
	59	-	-	581.1	574.5	581.6	581.6	525.6	20.6	**508.5**	0.0
	73	525.6	519.9	538.5	533.6	565.2	540.5	488.6	20.7	488.6	16.1
20	8	-	-	773.4	767.4	784.1	770.9	597.7	16.8	597.7	20.3
	22	-	-	782.0	774.8	782.5	782.5	601.1	31.5	601.1	31.5
	42	737.0	707.2	792.9	749.6	836.6	804.4	622.2	35.2	622.2	30.1
	59	-	-	734.3	728.8	750.1	730.1	637.7	38.3	**619.9**	41.7
	73	661.8	652.5	688.6	688.2	691.0	688.9	628.9	32.8	628.9	32.8

* *Average Value*: average solution cost over 30 runs of the same stochastic algorithm. *Best Value*: best solution found in any of those runs.
** Preferred settings: $\omega_s = 400$, $\omega = 2048$, *multi*$= 5$. *** The bold values are the only instances where the best solution found by any setting is better than the solution found by the preferred setting.

Table 3 Best solutions for $n \in \{25, 30\}$

n	seed	López-Ibáñez et al. (2022)*		CEGO*		UMM*		PnB: Preferred Settings**		PnB: Best Found***	
		Average	Best	Average	Best	Average	Best	Value	Optimality Gap (%)	Value	Optimality Gap (%)
25	8	-	-	888.2	887.8	888.3	887.8	763.3	34.5	**760.6**	33.8
	22	-	-	921.3	921.3	921.2	919.2	784.6	40.9	**774.0**	38.6
	42	881.5	865.7	927.9	904.0	941.5	910.3	751.3	37.8	**733.3**	36.3
	59	-	-	872.5	870.0	872.7	870.0	718.2	35	718.2	35.0
	73	873.6	863.7	904.6	882.5	911.7	883.7	743.4	39.8	743.4	39.8
30	8	-	-	1053.8	1047.9	1055.3	1055.3	898.7	48.4	**892.2**	45.4
	22	-	-	1078.1	1078.1	1078.1	1078.1	892.8	43.6	892.8	43.6
	42	1084.6	1065.2	1118.9	1103.2	1125.4	1103.2	835.1	44.2	835.1	44.2
	59	-	-	1118.7	1100.6	1115.5	1100.6	864.1	48.0	**858.1**	37.6
	73	967.7	952.1	1024.8	1024.8	1024.8	1024.8	882.9	45.8	882.9	45.8

* *Average Value*: average solution cost over 30 runs of the same stochastic algorithm. *Best Value*: best solution found in any of those runs.
** Preferred settings: $\omega_s = 2048$, $\omega = 2048$, *multi*$= 5$. *** The bold values are the instances where the best solution found by any setting is better than the solution found by the preferred setting.

embedded search may be worth the extra computational cost. Setting $\omega_s = 2048$ finds the best solutions on 24 of the 30 settings, so the largest search setting is again the preferred one.

We again compare our results with CEGO and UMM, and record the best solution found using our preferred setting as well as the best solutions found by any setting, as shown in Table 3. As before, the solutions from our preferred setting are better than the best solutions found by CEGO and UMM for all of the problems. The preferred solution only matches the best found solution for 5 of the 10 problems. However, the gap between the two solutions is quite small for the other 5 instances. The largest difference, which occurs with $n = 25$ and seed$= 42$, is only 2.4%.

7. Conclusions and Looking Forward

We study outer-inner optimization problems, where the outer problem is combinatorial and the inner problem is numerical and black-box. We introduce the first optimization framework designed to find exact solutions for such problems under mild assumptions about the inner problem. Global trajectory optimization is

one real-world example of such problems, and we use the Asteroid Routing Problem (ARP) as a case study. Although the inner optimizer in the ARP returns a deterministic local optimum, we show how to control the likelihood that this local optimum is also global by using multiple starting seeds. From the perspective of the outer problem, our proposed method optimally solves instances of up to 15 celestial bodies, and finds high-quality solutions for larger problems. We also find new best-known solutions for several instances. Our implementation, data, and a detailed guide on how to use the solver are publicly available.

We ran all of our experiments on a single thread, but the process of solving a DD using Peel-and-Bound is extremely parallelizable (Bergman et al. 2014, Perez and Régin 2018, Rudich et al. 2023). Each peeled DD can be considered a discrete problem to be solved, allowing the problem to be easily divided among any number of available processors, without requiring the processors to communicate (with the exception of requesting new tasks). Additionally, setting $multi > 1$ is parallelizable, as it represents *multi* independent restarts of SLSQP at disjoint areas of the search space. Thus, an implementation that included parallel processing could handle problems at a much larger scale if many CPUs were available.

For global trajectory optimization, our methods offer a novel approach to finding both optimal and high-quality feasible solutions. This opens the door to future research adapting our solver to other complex challenges, such as those involving multiple gravity assists. We provide a robust framework for addressing sequencing problems where the cost function is computationally demanding, the proposed approach is highly scalable, and its underlying principles promise broad applicability.

References

Abdelkhalik O, Gad A (2012) Dynamic-size multiple populations genetic algorithm for multigravity-assist trajectory optimization. *Journal of Guidance, Control, and Dynamics* 35(2):520–529.

Andersson H, Fagerholt K, Hobbesland K (2015) Integrated maritime fleet deployment and speed optimization: Case study from RoRo shipping. *Comput. Oper. Res.* 55:233–240.

Bergman D, Cire AA, Sabharwal A, Samulowitz H, Saraswat V, van Hoeve WJ (2014) Parallel combinatorial optimization with decision diagrams. *Proceedings of the International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, 351–367.

Bergman D, Cire AA, van Hoeve WJ, Hooker J (2016) *Decision Diagrams for Optimization* (Springer).

Ceriotti M, Vasile M (2010) Automated multigravity assist trajectory planning with a modified ant colony algorithm. *Journal of Aerospace Computing, Information, and Communication* 7(9):261–293.

Chicano F, Derbel B, Verel S (2023) Fourier transform-based surrogates for permutation problems. Silva S, Paquete L, eds., *GECCO*, 275–283 (ACM Press).

Cire AA, van Hoeve WJ (2013) Multivalued decision diagrams for sequencing problems. *Operations Research* 61(6):1259–1462.

Coppé V, Gillard X, Schaus P (2024) Decision diagram-based branch-and-bound with caching for dominance and suboptimality detection. *INFORMS Journal on Computing* 36(6):1522–1542.

Ehmke JF, Campbell AM, Thomas BW (2016) Vehicle routing to minimize time-dependent emissions in urban areas. *Eur. J. Oper. Res.* 251(2):478–494.

Gendreau M, Ghiani G, Guerriero E (2015) Time-dependent routing problems: A review. *Comput. Oper. Res.* 64:189–197.

Gillard X (2022) *Discrete Optimization with Decision Diagrams: Design of a Generic Solver, Improved Bounding Techniques, and Discovery of Good Feasible Solutions with Large Neighborhood Search*. Ph.D. thesis, Université Catholique de Louvain.

Gillard X, Coppé V, Schaus P, Cire AA (2021) Improving the filtering of Branch-And-Bound MDD solver. Stuckey PJ, ed., *CPAIOR 2021*, 231–247 (Springer).

Hennes D, Izzo D (2015) Interplanetary trajectory planning with Monte Carlo tree search. Yang Q, Wooldridge M, eds., *Proc. of IJCAI-15*, 769–775 (IJCAI/AAAI Press, Menlo Park, CA).

Hennes D, Izzo D, Landau D (2016) Fast approximators for optimal low-thrust hops between main belt asteroids. Chen X, Stafylopatis A, eds., *Computational Intelligence (SSCI), 2016 IEEE Symposium Series on*, 1–7.

Irurozki E, López-Ibáñez M (2021) Unbalanced mallows models for optimizing expensive black-box permutation problems. Chicano F, Krawiec K, eds., *GECCO*, 225–233 (ACM Press).

Izzo D (2015) Revisiting Lambert's problem. *Celestial Mechanics and Dynamical Astronomy* 121:1–15.

Izzo D, Getzner I, Hennes D, Simões LF (2015) Evolving solutions to TSP variants for active space debris removal. Silva S, Esparcia-Alcázar AI, eds., *GECCO*, 1207–1214 (ACM Press).

Izzo D, Simões LF, Märtens M, de Croon GC, Heritier A, Yam CH (2013) Search for a grand tour of the Jupiter Galilean moons. Blum C, Alba E, eds., *GECCO*, 1301–1308 (ACM Press).

Kraft D (1988) A software package for sequential quadratic programming. Technical Report DFVLR-FB 88-28, DLR German Aerospace Center, Institute for Flight Mechanics, Koln, Germany.

Lee T, Leok M, McClamroch NH (2007) A combinatorial optimal control problem for spacecraft formation reconfiguration. *2007 46th IEEE Conference on Decision and Control* (IEEE).

López-Ibáñez M, Chicano F, Gil-Merino R (2022) The asteroid routing problem: A benchmark for expensive black-box permutation optimization. Jiménez Laredo JL, et al., eds., *EvoApplications 2022*, 124–140 (Springer Nature).

Ow PS, Morton TE (1988) Filtered beam search in scheduling. *Int. J. Prod. Res.* 26:297–307.

Perez G, Régin JC (2018) Parallel algorithms for operations on multi-valued decision diagrams. *Proceedings of the AAAI Conference on Artificial Intelligence* 32(1).

Petropoulos AE, Bonfiglio EP, Grebow DJ, Lam T, Parker JS, Arrieta J, Landau DF, Anderson RL, Gustafson ED, Whiffen GJ, Finlayson PA, Sims JA (2014) GTOC5: Results from jet propulsion lab. *Acta Futura* 8:21–27.

Rudich I (2024) *Peel and Bound: Solving Discrete Optimization Problems with Decision Diagrams and Separation*. Ph.D. thesis, Polytechnique Montréal.

Rudich I, Cappart Q, Rousseau LM (2022) Peel-and-bound: Generating stronger relaxed bounds with multivalued decision diagrams. *International Conference on Principles and Practice of Constraint Programming*.

Rudich I, Cappart Q, Rousseau LM (2023) Improved Peel-and-Bound: Methods for generating dual bounds with multivalued decision diagrams. *J. Artif. Intell. Res.* 77:1489–1538.

Rudich I, López-Ibáñez M, Römer M, Cappart Q, Rousseau LM (2024) An Exact Framework for Solving the Space-Time Dependent TSP. Available for download at https://github.com/INFORMSJoC/2024.0866. The paper is available at https://doi.org/10.1287/ijoc.2024.0866.

Santucci V, Baioletti M (2022) A fast randomized local search for low budget optimization in black-box permutation problems. *Proceedings of the 2022 World Congress on Computational Intelligence (WCCI 2022)* (IEEE Press).

Shirazi A, Ceberio J, Lozano JA (2018) Spacecraft trajectory optimization: A review of models, objectives, approaches and solutions. *Progress in Aerospace Sciences* 102:76–98.

Simões LF, Izzo D, Haasdijk E, Eiben AE (2017) Multi-rendezvous spacecraft trajectory optimization with Beam P-ACO. Hu B, López-Ibáñez M, eds., *EvoCOP*, 141–156 (Springer).

Vasile M, De Pascale P (2006) Preliminary design of multiple gravity-assist trajectories. *Journal of Spacecraft and Rockets* 43(4):794–805.

Zaefferer M, Stork J, Friese M, Fischbach A, Naujoks B, Bartz-Beielstein T (2014) Efficient global optimization for combinatorial problems. Igel C, Arnold DV, eds., *GECCO*, 871–878 (ACM Press).