ASSISTED SHORTEST PATH PROBLEMS AND GLOBAL OPTIMIZATION OF MIXED-INTEGER NONLINEAR PROGRAMS WITH TRIGONOMETRIC FUNCTIONS

A Dissertation

by

CHRISTOPHER M. MONTEZ

Submitted to the Graduate and Professional School of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2023

Major Subject: Mechanical Engineering

ABSTRACT

Cooperative behavior between mobile agents has received increasing interest in fields such as robotics, operations research, agriculture, and more. The use of multiple agents allows for a variety of problems to be addressed that a single agent could not complete otherwise. This dissertation considers a particular cooperative path planning problem referred to as the assisted shortest path problem (ASPP). In the ASPP, a primary agent, referred to as a convoy, wishes to travel from a starting location to a destination in minimum time. The environment the convoy travels in contains obstructions that impedes the movement of the convoy. Depending on the obstructions present, the convoy may or may not be capable of removing these obstructions on its own. In either case, a second agent, referred to as the support vehicle, is simultaneously deployed with the convoy. The support vehicle can remove obstructions that impede the convoy and aims to assist the convoy to its destination so the convoy may reach the destination in minimum time. The support vehicle may be allowed to terminate anywhere in the environment or may have a specified destination of its own. Multiple variations of the ASPP are presented and solved in the first half of this dissertation. In the second half, the problem of solving factorable mixed-integer nonlinear programs (MINLPs) with trigonometric terms is considered. An exact algorithm solving this class of problems is then presented. The problem of identifying reasonable weights to assign to the graph used to represent an instance of the ASPP may be posed as a MINLP with trigonometric terms and is used as a motivating example in the discussion of solving this class of MINLPs. This approach can be further extended to solve factorable MINLPs with differentiable, periodic terms.

DEDICATION

To my friends and family that have supported me throughout my life.

I am truly a fortunate man.

ACKNOWLEDGMENTS

I sincerely thank everyone who has helped me throughout the duration of my Ph.D. I have learned a great deal throughout these years and I would not have been able to do it without the support I received along the way.

I want to thank Dr. Darbha and Dr. Rathinam for their guidance and patience they have shown me. They helped me grow both as an academic and as a person. They have given me opportunities that I can never thank them enough for.

I want to thank the Department of Mechanical Engineering at Texas A & M University for their support.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a committee consisting of Professor Swaroop Darbha (advisor), Professor Sivakumar Rathinam (co-advisor), and Professor Prabhakar Pagilla of the Department of Mechanical Engineering and Professor Lewis Ntaimo of the Department of Industrial & Systems Engineering. All other work conducted for the dissertation was completed by the student independently.

Funding Sources

Graduate study was supported by the Engineering Graduate Merit Fellowship, the Continuing Student Fellowship, and the Graduate Teaching Fellowship from Texas A&M University. Additional funding was also provided by Sandia National Laboratories, the Air Force Research Laboratory, and Los Alamos National Lab.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

In recent years, much attention has been given to problems concerned with the coordination and cooperation of two or more agents (autonomous, semi-autonomous, human-operated, or a combination thereof) in order to achieve some specified task(s). These problems find application in many fields such as robotics [1, 2, 3], search-and-rescue [4], military combat, and vehicle routing [5, 6, 7] to name a few, and these problems come in many forms. Large teams of cooperative autonomous robots are used to manage automated warehouses and sortation facilities [1, 2, 3]. The use of large teams of robots for managing such facilities leads to large improvements in the overall performance (e.g., throughput of goods) of these facilities. Teams of semi-autonomous robots may be used to aid in the search of those affected by a disaster [4]. Such teams of semi-autonomous robots may be used to cover large areas that have been affected by a disaster while communicating with one or many human operators in order to make decisions on how to proceed with the search. In both of the examples given, a team of agents work together to accomplish some task(s) by making use of the capabilities of the agents available. In some scenarios, all agents may be identical to one another and the use of multiple agents leads to improvements by dividing many tasks amongst the agents (such as moving goods in an automated warehouse). In other scenarios, agents may have different capabilities and so an agent is able to overcome obstacles that it would not have been able to otherwise without assistance from one or many other agents. This dissertation will focus on the latter case, though many of the results and methods herein may be adapted to the former case.

In much of the robotics literature concerning the cooperation of multiple agents, the path planning algorithms used require the robots avoid "collisions" on a graph used to represent the environment the robots operate in. These "collisions" do not necessarily correspond to a true, physical collision, as a vertex in the environment's graph representation may represent a large region. Regardless, by requiring the path planning algorithms avoid "collisions", many complications and true collisions may be avoided. This, however, limits how the agents may cooperate. For example, if the power of two or more agents is required to move an obstruction in a search-and-rescue scenario, these agents necessarily must occupy the same region in the environment and hence likely occupy the same vertex or edge in the environment's graph representation. Such a scenario cannot be handled when requiring the agents avoid "collisions" in the graph. As such, this requirement will be avoided to further facilitate cooperation between agents.

1.1 Focus of the Dissertation

This dissertation considers the problem of a primary agent (robot, UAV, etc.) traveling in a graph representing an impeded environment to a destination in minimum time while being assisted by secondary agent (robot, UAV, etc.). The primary agent is referred to as the convoy and the secondary agent is referred to as the support vehicle. The obstructions present in the environment impeding the convoy's travel may be physical, such as debris, or abstract, such as requiring a certificate of safety after inspection. The obstructions are represented by assigning an additional weight to select edges in a graph representing the environment. These edges are referred to as initially impeded edges. All other edges in the graph without obstructions are referred to as unimpeded edges. All edges have an unimpeded travel cost (time) associated with them and all impeded edges have a second, higher impeded travel cost (time) associated with them. The act of removing an obstruction from an edge is referred to as servicing an edge and is accomplished when an appropriate vehicle fully traverses the edge. When an edge is serviced, it is assumed to always remain serviced, i.e., the obstruction never returns, and the edge's cost is changed to the unimpeded travel cost. The convoy is permitted to service an initially impeded edge at a relatively high cost (travel time). In the problem's slightly more restricted form, the convoy is unable to traverse an impeded edge and must wait for the support vehicle to service the edge before taking it. In either case, the support vehicle assists the convoy by servicing impeded edges in the graph so as to reduce the cost (travel time) for the convoy when taking these edges. The support vehicle is typically (but not required to be) better equipped for servicing an edge and hence the cost of servicing an edge with the support vehicle is typically lower than the cost of the same with the convoy. The convoy and support vehicle are permitted to wait at any vertex and the support vehicle is permitted to terminate at any vertex in the general case. The problem is then to find a pair of paths, one for the convoy and one for the support vehicle, such that the convoy reaches its destination in minimum time while being assisted by a support vehicle (which may not be deployed in general). This problem is herein referred to as the assisted shortest path problem (ASPP).

The ASPP as described is defined on a graph representing a real-world environment in which obstructions are present that limit the primary agent's mobility. In order to properly define the ASPP, weights must be assigned to each edge in the graph (with multiple weights being assigned to each edge to account for there being two vehicles and an edge being obstructed or unobstructed). In the most classical case, these edge weights will correspond to travel times along some path in the real-world environment and the path taken is typically a shortest path of some sort. An edge (i, j) may represent a path in the real-world environment where the agent must travel through various waypoints specified by a designer, with the first waypoint being i and the final waypoint being j. The intermediate waypoints between i and j may take into account the constraints on terrain and motion of the agents. To account for the motion constraints of each agent, each agent may be modelled as a Dubins vehicle [8], where the agent operates in a plane, travels at a constant speed along a path, and has a minimum turning radius determined by the construction of the agent. The problem of finding the shortest path through a sequence of points using a Dubins vehicle is herein referred to as the Markov-Dubins path planning problem (MDPPP). The MDPPP is a difficult problem to solve and currently an analytical solution is only known for up through 3 points [9]. In [10], it is shown the MDPPP can be posed as a mixed-integer non-linear program (MINLP). The non-linearities present in the MINLP formulation correspond to bilinear and trigonometric terms. The bilinear terms can currently be handled using existing methods, but the presence of the trigonometric terms leads to many difficulties. As of writing this dissertation, there are only two solvers known to the author that claim to have the capability of handling trigonometric terms in an MINLP (LINDOGlobal [11] and Couenne [12]), but these solvers struggle both in speed and accuracy even for a moderate number of trigonometric terms. In this dissertation, a new global optimization algorithm is presented to solve MINLPs with trigonometric terms and a comprehensive computational study is performed to illustrate the effectiveness of this new approach when applied to solving the MDPPP. This approach can therefore be used by a designer to determine appropriate edge weights for the graph used to define the ASPP.

1.2 Structure of the Dissertation

This dissertation is divided into two main chapters following this chapter and a final chapter with concluding remarks and a discussion on future work. While both of the subsequent main chapters may be related to each other as done in this dissertation, the work in each chapter may be extended to a larger class of independent problems without reference to each other. Therefore, the work is divided into two self-contained main chapters for clarity in their impact on differing areas of research.

In Chapter 2, the ASPP is formally defined in Section 2.2 and three variants are presented in the subsequent sections. The first variant, referred to as the restricted ASPP, is covered in Section 2.4 and has the following additional constraints: (i) the convoy and support vehicle must take paths without cycles, (ii), the support vehicle is required to terminate at a specified vertex, and (iii) the convoy is unable to traverse initially impeded edges unless they have been serviced by the support vehicle. A mixed-integer non-linear programming (MINLP) formulation of the restricted ASPP is presented in Section 2.4.1. In Section 2.4 this MINLP formulation is converted to an equivalent mixed-integer linear programming (MILP) formulation using the standard big-M [13] technique. This MILP formulation is then solved for many test instances in Section 2.4.3. The second variant, referred to as the trailing convoy ASPP, differs from the ASPP only in that the convoy is unable to service edges, but it is permitted to trail behind the support vehicle as the support vehicle services an impeded edge. The convoy and support vehicle are permitted to take paths with cycles and the support vehicle is permitted to terminate at any vertex as in the ASPP. Due to both vehicles being permitted to take cycles, any MINLP formulation for the trailing convoy ASPP will prove to be computationally prohibitive to solve. With this in mind, a $(2 + \varepsilon)$-approximation algorithm is presented for the trailing convoy ASPP in Section 2.5 and a corresponding computational study is given in Section 2.5.5. In order to handle the ASPP without additional constraints or restrictions, an asynchronous permanent labeling algorithm is presented in Section 2.6. This labeling algorithm is an extension of the generalized labeling algorithm used to solve the closely related shortest path problem with time windows (SPPTW) [14]. The presented labeling algorithm provides a means of solving cooperative routing problems with general path structural constraints (such as the interactions between the convoy and support vehicle as in the ASPP).

In Chapter 3, a global optimization algorithm that can be applied to factorable MINLPs with differentiable, periodic functions in the constraints, with the primary focus being on trigonometric functions such as sine and cosine is presented. The global optimization algorithm solves a sequence of MILPs with the solutions of these MILPs approaching the optimal solution of the original MINLP in the limit. In Section 3.2, a general overview of the algorithm is shown. In Section 3.3, preliminary notation and terminology is given. In Section 3.4, relaxations for trigonometric and bilinear terms used to construct an MILP relaxation of the original MINLP are presented. These relaxations are constructed using a collection of partitions of the relevant variables' domains and these partitions are the subject of Section 3.5. In Section 3.6, a novel procedure is presented to significantly reduce the domain of the original MINLP (and hence also the MILP relaxations). In Section 3.7, the Markov-Dubins path planning problem [10] is presented as a motivating example. In Section 3.8, a computational study illustrating the effectiveness of the presented algorithm is given.

In Chapter 4, concluding remarks and a discussion of future work is given.

2. ASSISTED SHORTEST PATH PROBLEM[*]

2.1 Introduction

Cooperative behavior between mobile autonomous robots has received increasing interest in the robotics community due to its many applications such as traffic control, cooperative manipulation (box-pushing) [17], [18], and foraging [19]. Cooperation between multiple mobile robots leads to potential improvement in performance when compared to that of a single robot performing the same task. Additionally, cooperation between multiple robots with different capabilities allows a wider variety of problems to be solved. We refer an interested reader to [20] for a comprehensive survey of existing research related to the cooperative behavior of multiple autonomous mobile robots.

The focus of this chapter is the path planning of two autonomous agents operating in a partially impeded environment with the express goal of a designated agent reaching its destination in minimum time while being assisted by the other agent. We refer to the agent whose goal is to reach the destination in minimum time as the convoy vehicle and we refer to the agent assisting the convoy as the support vehicle. The starting positions of the convoy and support vehicle may be distinct in general. The support vehicle may or may not have a specified destination. The impeded environment is represented by a graph whose vertices and edges are chosen by a designer *a priori*. Select edges are used to represent the obstructions present in the environment and we refer to these edges as impeded edges. There are two scenarios to consider: (i) the convoy is capable of traversing impeded edges at an additional cost and (ii) the convoy is strictly unable to traverse an impeded edge. In both cases, the support vehicle is capable of removing obstructions so as to assist the convoy's travel. In the first case, this amounts to reducing the cost of the convoy traversing an initially impeded edge. In the second case, this amounts to making a previously unavailable edge available to the convoy. The objective is then to find a pair of paths, one for the convoy and one for the support vehicle, such that the convoy reaches its destination in minimum time, where the convoy may only take an impeded edge if it has first been attended to by the support vehicle. We refer

[*]Material presented in this chapter has been previously published by the author. See [15] and [16].

to the case where there is a single support robot as the *Assisted Shortest Path Problem* (ASPP) for ease of discussion. We will focus on the case where there is a single support robot, but this problem can naturally be generalized to n support robots. It should be noted typically in multi-robot path planning problems, it is assumed multiple robots may not simultaneously share an edge in the graph [20]. In our formulation (see Section 2.2), we do allow the cardinal and support robot to share an edge, but we add a restriction. In the event the support robot is traversing an impeded edge for the first time, the cardinal robot must remain behind the support vehicle if it is sharing this edge with the support robot. In this case, the support robot is interacting with or altering the environment and so the cardinal robot must remain behind the support robot as it makes the path accessible to the cardinal robot.

2.2 Problem Statement

Let $G = (V, E)$ be a simple, connected, undirected graph representing an impeded environment. V is the set of vertices and E is a set of undirected edges. These vertices and edges are chosen by a designer *a priori* and are assumed to reasonably represent the environment. The convoy and support vehicle start at vertices p and q, respectively, where p and q need not be distinct. Let $d \in V$ be the destination of the convoy. The support vehicle is permitted to terminate at any vertex (the presented work may be modified to assign a destination to the support vehicle). Let edges $K \subseteq E$ denote the impeded edges in the graph. An edge may be impeded due to a physical obstruction (debris, broken road, etc.) or an abstract obstruction (permission to take a path, verify certain condition is met, etc.). The impeded edges K are assumed to be known *a priori* by the designer. The remaining edges (i.e., $E \setminus K$) are referred to as the unimpeded edges of the graph. We will say an impeded edge has been *serviced* if either the convoy or support vehicle has completely traversed that edge. An impeded edge being serviced is equivalent to the relevant obstruction (physical or abstract) being removed in the impeded environment. We will use the term impeded travel cost to refer to the cost for a vehicle to take an impeded edge that has not yet been serviced. We use the term unimpeded travel cost to refer to the cost for a vehicle to take an initially unimpeded edge or a serviced impeded edge. Each edge $e \in E$ has four positive edge weights of the form $(T_e^u, T_e^i, \tau_e^u, \tau_e^i)$. T_e^u and T_e^i are the unimpeded and impeded travel cost for the convoy, respectively. The terms τ_e^u and τ_e^i are the unimpeded and impeded travel cost for the support vehicle, respectively. The costs T_e^u, τ_e^u, and τ_e^i are taken to be finite, but T_e^i is permitted to be infinite. A positive infinite cost T_e^i corresponds to the convoy being unable to take the impeded edge e without the assistance of the support vehicle. For unimpeded edges, we have $T_e^i = T_e^u$ and $\tau_e^i = \tau_e^u$. For impeded edges, we require $T_e^i > T_e^u$ and $\tau_e^i > \tau_e^u$, i.e., for any vehicle, the cost of taking an initially impeded edge after it has been serviced is at most the corresponding cost for the unimpeded or serviced edge. All edges are undirected and so the cost in both directions are considered to be identical. When an impeded edge is serviced, it is assumed to remain serviced. To avoid ambiguity, unless an initially impeded edge has been completely serviced, the cost associated with a vehicle taking this edge is taken to be the impeded cost.

We will assume the following: (i) the convoy and support vehicles can share vertices and edges without conflict, (ii) the two vehicles start at the same time, (iii) the two vehicles communicate at all times and information is shared in a negligible amount of time. The convoy is permitted to wait at any vertex to give the support vehicle time to service an impeded edge. Waiting will incur some additional cost. For the sake of simplicity, the cost will simply be the time elapsed since the start of the mission. The work presented in this chapter can be easily modified to accommodate more complex costs. The support vehicle is also permitted to wait, but doing so will incur zero additional cost unlike in the case of the convoy. This is done for two reasons. First, depending on the application, the support vehicle may be able to temporarily shut off or idle and incur a negligible power consumption cost (should the cost be related to power rather than strictly time itself). Second, in many applications the support vehicle will correspond to a vehicle such as a fixed-wing UAV which does not have the capability to easily stop in the environment. It should be noted in the case of $T_e^i = \infty$ for all impeded edges (i.e., the convoy is strictly unable to traverse impeded edges), the support vehicle will only stop at a vertex to terminate its journey rather than wait in an optimal solution. In such a case, which is the subject of Section 2.4, neglecting the waiting cost of the support vehicle simplifies the model. This zero waiting cost may be modified for cases where the support vehicle is physically capable of waiting at a vertex. Introducing a non-zero waiting cost for the support vehicle is addressed at the end of Section 2.6.

Let X_{pa} be a path from p to $a \in V$ for the convoy and $\bar{X}_{qb}$ be a path from q to $b \in V$ for the support vehicle. A vehicle's path may also include waiting at one or more vertices. The two paths are coupled by the vehicles' interactions. That is, the cost of X_{pa} will depend on the decisions made by the support vehicle in $\bar{X}_{qb}$ and vice-versa. Note that the support vehicle is permitted to remain at q for the duration of the mission, corresponding to the convoy traveling to its destination without any assistance from the support vehicle. For the sake of simplicity, the cost of the two paths will be taken to be the time the convoy reaches its destination, d, while adhering to the previously described travel cost rules. As previously mentioned, a more general cost scheme may be considered and the following work may be modified to accommodate this change. The cost of the coupled paths will be denoted by $C(X_{pa}, \bar{X}_{qb})$. The assisted shortest path problem (ASPP) is then to find the two coupled paths X_{pd} and $\bar{X}_{qv}$, where v is any vertex in V, such that $C(X_{pd}, \bar{X}_{qv})$ is minimized.

A simple example instance of the ASPP is shown in Figure 2.1. In this example, the costs represent units of time elapsed when taking an edge. In Figure 2.1, initially impeded edges are indicated by red and initially unimpeded edges are indicated by black. Suppose the convoy and support vehicle both start at vertex 0 (i.e., $p = 0$ and $q = 0$). In the instance shown in Figure 2.1, both impeded edges have finite costs and so the convoy is permitted to take an impeded edge without the assistance of the support vehicle, but at a higher cost. From Figure 2.1, the optimal solution is $X^*_{pd} = (0, 3, 2, d)$ and $\bar{X}^*_{q2} = (0, 3, 2)$. In this solution, both the convoy and support vehicle travel to vertex 3, with the convoy and the support vehicle taking 2 units of time and 1 unit of time, respectively. The support vehicle then immediately takes the edge $(3, 2)$ to service it. Once the convoy arrives at vertex 3, it will wait 1 unit of time for the support vehicle to finish servicing edge $(3, 2)$. After the support vehicle reaches vertex 2, it terminates and the convoy takes the now serviced edge $(3, 2)$ and finally the unimpeded edge $(2, d)$ to reach its destination. The total time elapsed for the convoy to reach d is 10 units of time. In this example, the two impeded edges $(0, 2)$ and $(2, 3)$ represent two kinds of obstructions that may be present in a real-world scenario. Impeded edge $(0, 2)$ has a relatively small difference between the convoy's impeded and unimpeded travel costs and so this edge represents path containing an obstruction in which the convoy is well-suited to handle, but the time needed to take this path is large. Conversely, impeded edge $(2, 3)$ has a relatively large difference between the convoy's impeded and unimpeded travel costs and so this edge represents a path containing an obstruction in which the convoy is ill-suited to handle, but the time needed to take this path without the obstruction is small. In both cases, the support vehicle is well-suited to handle the obstruction and so the presence of the support vehicle leads to an alternate path being made available that would have otherwise been discarded when planning the path for the convoy. This alternate path leads to an improved solution when compared to any path for the convoy without the assistance of the support vehicle.

2.3 Overview

This remainder of this chapter is divided into three major parts: (i) solving a restricted variant of the ASPP using mathematical programming, (ii) an approximation algorithm for the ASPP described in Section 2.2, and (iii) an asynchronous generalized permanent labeling algorithm for general ASPP variants. In part (i), the convoy and support vehicle will be restricted to taking elementary paths (i.e., paths with no cycles), the support vehicle will be assigned a destination rather than be allowed to terminate at any vertex, and the convoy is unable to traverse impeded edges without the assistance of the support vehicle. A mixed-integer linear programming (MILP) formulation will be presented for this variant. This MILP can then be solved using standard commercial solvers. In part (ii), an approximation algorithm for the ASPP from 2.2 is presented for the case T_e^i is infinite for each impeded edge. In part (iii), a permanent labeling algorithm is presented that can be extended to solve many variants of the ASPP, though at a high computational cost.

2.4 Mixed-Integer Linear Programming Formulation for Restricted ASPP

In this section we present a mixed-integer non-linear programming (MILP) formulation for the problem previously described, with the additional requirements that the support vehicle must take a path without cycles, the support vehicle must terminate at a specified vertex $d' \in V$, and the

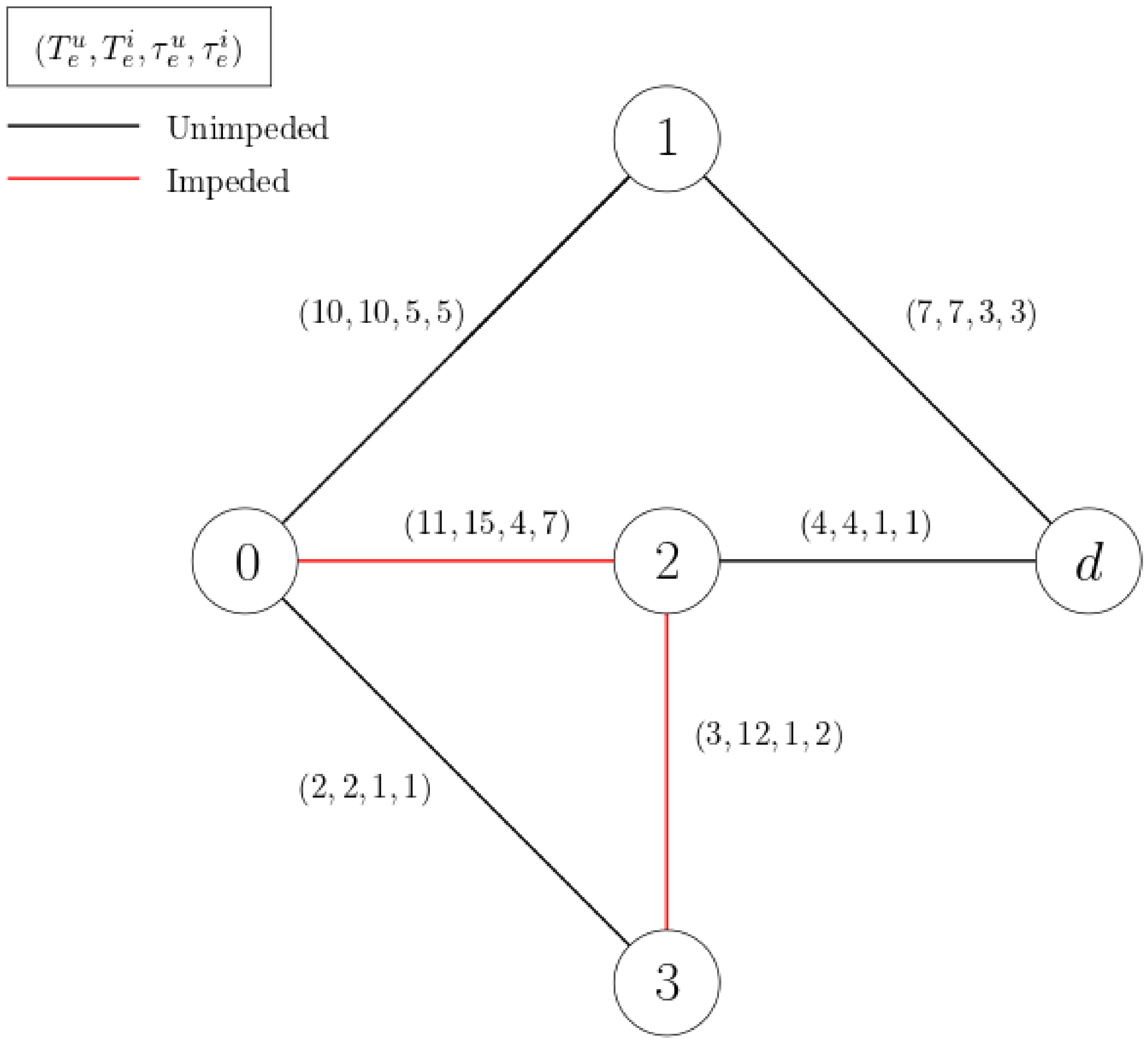


Figure 2.1: Example ASPP instance. Initially impeded edges are indicated by red edges and initially unimpeded edges are indicated by black edges. The initially impeded edge $(0, 2)$ represents a path with an obstruction the convoy is well-equipped to handle. Conversely, the initially impeded edge $(2, 3)$ represents a path with an obstruction the convoy is ill-equipped to handle and so the assistance of the support vehicle is largely beneficial along this edge.

convoy is unable to traverse impeded edges without the assistance of the support vehicle. We first construct a mixed-integer nonlinear programming (MINLP) formulation and use the standard big-M technique [13] to convert the MINLP into an MILP. The additional constraints are necessary to make the MILP formulation manageable for a commercial solver. The material presented in this section closely follows [16] by the author and colleagues.

2.4.1 MINLP Formulation

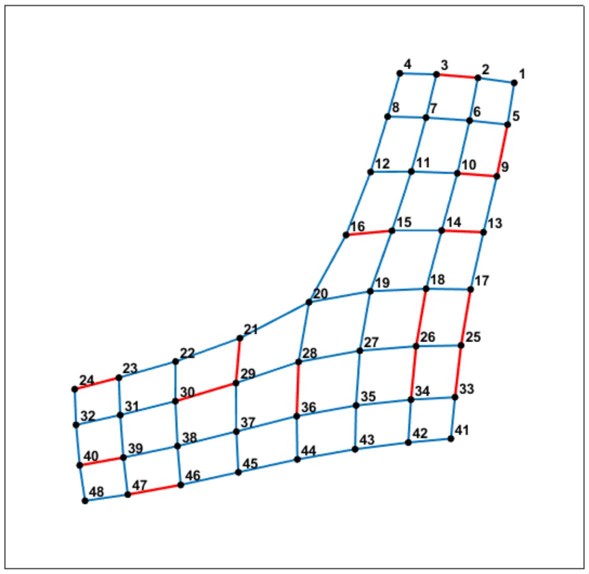
2.4.1.1 Preliminary Adjustments

For this formulation, we will treat $G = (V, E)$ as an equivalent directed graph $G' = (V, A)$ where A is the set of arcs where for each $(i, j) \in E$ we have $(i, j) \in A$ and $(j, i) \in A$. For each impeded edge $(i, j) \in K$, we say $(i, j) \in A$ and $(j, i) \in A$ are impeded arcs and denote the set of impeded arcs by K'. The arc weights are taken to be symmetric. In G', the treatment of an impeded edge is bi-directional. Therefore, when impeded arc $(i, j) \in K'$ is serviced by the support vehicle, we also say $(j, i) \in K'$ has also been serviced. The support vehicle's travel time is only affected by the arc it has taken. We add the additional requirement that the support vehicle *must take a path with no cycles*. Finally, we also assume the convoy is unable to traverse an impeded edge (arc) without the assistance of the support vehicle. This is equivalent to T_a^i being infinite for all impeded arcs $a \in K'$.

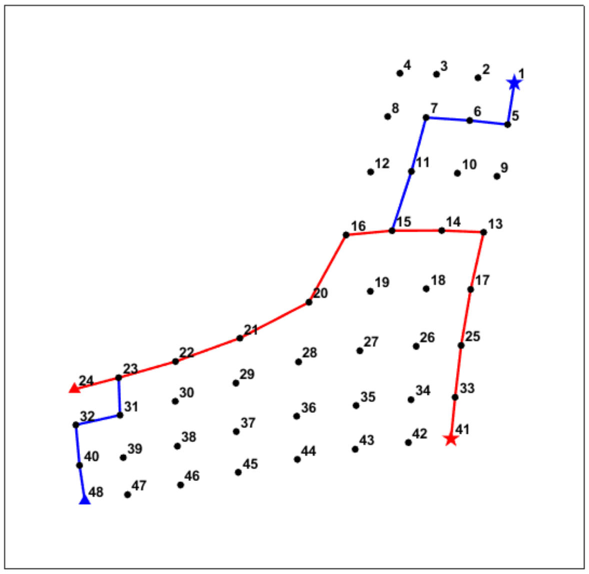
An example instance and its optimal solution is shown in Figure 2.2. In this example, both the convoy and the support vehicle take simple paths (i.e., paths with no cycles). In this particular example, the support vehicle assists the convoy by servicing the impeded edge (15, 16) (in particular, by servicing arc (16, 15) while noting servicing is bi-directional) before continuing its journey to its destination, vertex 41.

2.4.1.2 Decision Variables

For each arc $(i, j) \in A$, we introduce two binary variables x_{ij} and y_{ij}. The variable x_{ij} (resp. y_{ij}) takes on the value 1 if the convoy (resp. support vehicle) takes arc (i, j) in its path and the value 0 otherwise. For each vertex $v \in V$, we introduce two continuous variables $t_v^c \geq 0$ and $t_v^s \geq 0$,



(a)



(b)

Figure 2.2: (a) Example instance using an L-grid. Unimpeded edges are shown in blue and impeded edges are shown in red. (b) Optimal solution to the restricted ASPP for (a), with the convoy path shown in blue and the support vehicle path shown in red. In this optimal solution, neither vehicle takes any cycles. Both vehicles start from the vertex indicated by a triangle and terminate at the vertex indicated by a star.

where t_v^c and t_v^s represent the time the convoy and support vehicle, respectively, first *arrive* to vertex v. By definition, we have $t_p^c = 0$ and $t_q^s = 0$ for any instance.

2.4.1.3 Degree Constraints

We require the convoy and support vehicle take continuous paths. This is modeled using flow constraints, also known as degree constraints. Let $N^c = V \setminus \{p, d\}$ and $N^s = V \setminus \{q, d'\}$. For the convoy we have

$$\sum_{j\,:\,(i,j)\in A} x_{ij} - \sum_{j\,:\,(j,i)\in A} x_{ji} = \begin{cases} 1, & i = p, \\ -1, & i = d, \\ 0, & \forall i \in N^c \end{cases} \tag{2.1}$$

Similarly, for the support vehicle we have

$$\sum_{j\,:\,(i,j)\in A} y_{ij} - \sum_{j\,:\,(j,i)\in A} y_{ji} = \begin{cases} 1, & i = q, \\ -1, & i = d', \\ 0, & \forall i \in N^s \end{cases} \tag{2.2}$$

Constraints (2.1) and (2.2) state the convoy and support vehicle must leave their respective starting points and terminate at their respective destinations while taking a continuous path.

2.4.1.4 Time Updates

The next set of constraints correspond to how the arrival time at a vertex is updated when a vehicle reaches that vertex. For the convoy, we have

$$x_{ij}(t_i^c + T^u_{(i,j)} - t_j^c) \leq 0, \qquad \forall (i,j) \in A \tag{2.3}$$

Similarly, for the support vehicle we have

$$y_{ij}(t_i^s + \tau_{(i,j)}^i - t_j^s) \leq 0, \quad \forall (i,j) \in A \tag{2.4}$$

Constraint (2.3) says the time the convoy arrives at j is *at least* the time the convoy arrived at i plus the convoy's unimpeded travel time from i to j. Similarly, constraint (2.4) says the time the support arrives at j is *at least* the time the support arrived at i plus the support's unimpeded travel time from i to j and the service time. If the convoy or support vehicle does not take arc $(i, j) \in A$, we simply get the redundant constraint $0 \leq 0$ for that corresponding vehicle. An inequality is used instead of equality since we have made no restriction on the vehicles' ability to wait at a vertex. With the coordination constraints presented later, the convoy may need to wait at i before departing to j. We note that constraints (2.3) and (2.4) are bilinear.

2.4.1.5 Dynamic Time Window Constraints

The next set of constraints we introduce are *dynamic time window constraints*. We define a time window to be an interval of time representing the earliest and latest a vehicle may *arrive* at a vertex. We first discuss the intuition behind how these constraints are constructed. We then present the dynamic time window constraints used for this formulation.

Suppose the earliest and latest the convoy may arrive at vertex $j \in V$ is a_j and b_j, respectively, and so the convoy's time window for j is $[a_j, b_j]$. The corresponding time window constraint for the convoy arriving at j would then be expressed as

$$a_j \leq t_j^c \leq b_j.$$

For the problem at hand, we have $b_j = +\infty$ for both the convoy and the service vehicle, as there is no restriction on the latest either vehicle may arrive at any vertex. Next, suppose (i, j) is an impeded arc and the convoy intends to use this arc. We note a_j will depend on the time (i, j) or (j, i) has been treated. Since treatment of impeded arcs is bi-directional, we will need to consider two cases.

In the first case, a_j will depend on the time the support vehicle has treated (i,j) by taking (i,j); in this case the support vehicle travels in the same direction as the convoy when servicing (i,j). In the second case, a_j will depend on the time the support vehicle serviced (i,j) by taking (j,i); in this case, the support vehicle travels in the opposite direction as the convoy since servicing is bi-directional. We note that if (i,j) is an unimpeded arc, then $a_j = t_i^c + T_{(i,j)}^u$, which was already captured by constraints (2.3). Therefore, the time window constraints only correspond to the set of impeded arcs, K. We also note the support vehicle does not have any restrictions on the time it arrives to any vertex in the graph.

We now present the dynamic time window constraints used in this formulation. As previously mentioned, we need only focus on impeded arcs. Suppose the convoy is to use an impeded arc $(i,j) \in K$. We have two cases to consider. First, suppose the support vehicle's path is such that (i,j) is treated by the support vehicle using arc (i,j) (i.e., the support treats (i,j) in the same direction the convoy is traveling). The convoy must wait until the support vehicle has arrived at j before using arc (i,j) and the convoy must take at least an additional $T_{(i,j)}^u$ units of time before reaching j. We then have

$$(x_{ij} + y_{ij} - 1)(t_j^s + T_{(i,j)}^u) \leq t_j^c, \quad \forall (i,j) \in K. \tag{2.5}$$

In the second case, suppose the support's path is such that (i,j) is serviced by the support using arc (j,i) (i.e., the support services (i,j) in the opposite direction the convoy is traveling). The convoy must wait until the support has arrived at i before using arc (i,j) and the convoy must take at least an additional $T_{(i,j)}^u$ units of time before reaching j. We then have

$$(x_{ij} + y_{ji} - 1)(t_i^s + T_{(i,j)}^u) \leq t_j^c, \quad \forall (i,j) \in K. \tag{2.6}$$

Constraints (2.5) and (2.6) are the dynamic time window constraints. In the event at least one of the binary decision variables in (2.5) and (2.6) is zero, we will have t_j^c is greater than or equal to some negative number. However, we defined t_j^c to be non-negative for all nodes in the graph, so this constraint would be effectively redundant.

2.4.1.6 Coordination Constraints

The last set of constraints required are coordination constraints given by

$$x_{ij} \le y_{ij} + y_{ji}, \quad \forall (i,j) \in K \tag{2.7}$$

These constraints along with constraints (2.5) and (2.6) prevent the convoy from taking an impeded arc before the arc (and hence the corresponding edge due to bi-directional servicing) has been serviced by the support vehicle.

2.4.2 MILP Using Big-M

The previous constraints will lead to a mixed-integer non-linear program (MINLP), which will prove to be difficult for commercial solvers to solve. Constraints (2.3) through (2.7) are non-linear (bi-linear to be more precise) and so these constraints can be rewritten using the standard big-M [13] approach to convert the MINLP into an equivalent MILP. Doing so yields the following MILP for the restricted ASPP (denoted by ASPPr):

$$(ASPPr) \qquad \text{minimize } t^c_d \tag{2.8a}$$

$$\text{subject to} \quad \sum_{j\,:\,(i,j)\in A} x_{ij} - \sum_{j\,:\,(j,i)\in A} x_{ji} = \begin{cases} 1, & i = p, \\ -1, & i = d, \\ 0, & \forall i \in N^c \end{cases} \tag{2.8b}$$

$$\sum_{j\,:\,(i,j)\in A} y_{ij} - \sum_{j\,:\,(j,i)\in A} y_{ji} = \begin{cases} 1, & i = q, \\ -1, & i = d', \\ 0, & \forall i \in N^s \end{cases} \tag{2.8c}$$

$$M(x_{ij} - 1) + t^c_i + T^u_{(i,j)} \le t^c_j, \quad \forall (i,j) \in A \tag{2.8d}$$

$$M(y_{ij} - 1) + t^s_i + \tau^i_{(i,j)} \leq t^s_j, \quad \forall (i,j) \in A \tag{2.8e}$$

$$M(x_{ij} + y_{ij} - 2) + t^s_i + T^u_{(i,j)} \leq t^c_j, \quad \forall (i,j) \in K \tag{2.8f}$$

$$M(x_{ij} + y_{ji} - 2) + t^s_i + T^u_{(i,j)} \leq t^c_j, \quad \forall (i,j) \in K \tag{2.8g}$$

$$x_{ij} \leq y_{ij} + y_{ji}, \quad \forall (i,j) \in K \tag{2.8h}$$

$$x_{ij} \in \{0,1\}, \quad \forall (i,j) \in A \tag{2.8i}$$

$$y_{ij} \in \{0,1\}, \quad \forall (i,j) \in A \tag{2.8j}$$

$$t^c_i \geq 0, \quad \forall i \in V \tag{2.8k}$$

$$t^s_i \geq 0, \quad \forall i \in V \tag{2.8l}$$

In (2.8), M is a sufficiently large constant. For this formulation, the value of M was chosen as follows. First, construct an auxiliary graph $H = (V, A)$ where H has the same set of nodes and arcs as the original graph G, but the weight w_{ij} of each arc $a = (i,j)$ is set to $w_{ij} = T^u_a + \tau^i_a$. That is, the weight of each arc is the sum of the unimpeded travel cost of the convoy and the impeded cost of the support vehicle. Denote the maximum spanning tree of H by T_{max}. The constant M was then chosen to be

$$M = 2 \sum_{a=(i,j) \in T_{max}} (T^u_a + \tau^i_a). \tag{2.9}$$

This choice of M will be sufficiently large since each vehicle may only take an elementary path and so each edge in the maximum spanning tree can only be taken at most once by each vehicle. With this choice of M, it can be seen that if any binary decision variable is zero in constraints (2.8d)-(2.8g), the resulting negative term on the left-hand side will be large enough to make the left-hand side of each constraint negative overall, making these equations redundant due to (2.8k) and (2.8l). Constraints (2.8d)-(2.8g) then only come into effect when the terms being multiplied by M are equal to zero due to the corresponding binary variables taking on positive values. However, as is common when using a big-M approach, this choice of M is relatively loose for many instances. This may lead to poor computational performance in some cases.

2.4.3 Computational Results

Two sets of randomly generated graph instances were constructed. These two sets of instances will be referred to as Set 1 and Set 2, respectively. Each set consisted of 200 randomly generated instances. The 200 instances were split into 10 groups of 20 instances each. The instances in each group were given a fixed number of vertices, $|V|$. For both Set 1 and Set 2, the first 10 groups were made so that $|V|$ was 8, 10, 12, 14, 16, 18, 20, 22, 24, and 26, respectively. For each instance, the network was generated by randomly placing the vertices of the graph at integer-valued coordinates (x, y) in a $[0, 50] \times [0, 50]$ region. Two vertices were connected by a forward arc and an inverse arc if the Euclidean distance between the two vertices was at most 20 units. After arcs were added, if the resulting network was not connected then additional arcs were added to connect the next closest vertices until the resulting network was connected. The unimpeded travel cost of the convoy, T_a^u, and the unimpeded travel cost of the support vehicle, τ_a^u, for each arc $a = (i, j) \in A$ were both set to be randomly selected integers in the range $[5, 20]$. The same values were assigned to the inverse arcs $(j, i) \in A$. The number of initially impeded arcs, $|A|$, was set in advance for each instance. For Set 1, 25 percent of the arcs in the network were chosen to be impeded arcs. For Set 2, 50 percent of the arcs in the network were chosen to be impeded arcs. Arcs were randomly selected to be an impeded arc. Since $|A|$ is not known in advance due to how the networks were constructed, it is possible $0.25|A|$ or $0.5|A|$ is not an integer value or is odd. When this occurred, $|K|$ was taken to be the nearest even integer below the computed value of $|K|$. When arc (i, j) was set to be an impeded arc, the inverse arc (j, i) was also set to be an impeded arc with the same service cost. The service cost for each impeded arc was set to be a randomly chosen integer in the range $[1, 5]$. The origin vertices p and q were set to be the vertices nearest $(0, 0)$ and $(0, 50)$, respectively, for each instance. Similarly, destination vertices d and d' were set to be the vertices nearest $(50, 50)$ and $(50, 0)$, respectively, for each instance. This scheme was selected in an attempt to avoid trivial instances corresponding to the convoy taking very few arcs to reach its destination without the assistance of or with minimal assistance from the support.

For each instance, MILP (2.8) was solved on an Inspiron 5676 (3.2 GHz, 8 GB RAM) in the Julia programming language [21] using version 1.4.2. The JuMP package [22] was used to implement the MILP formulation and the Gurobi solver [23] was used to solve the MILP. Each instance was given a time limit of 1 hour to solve. For each group of 20 instances, the minimum, average, and maximum run time needed to solve the MILP was recorded. If an instance reached the 1 hour time limit, it was not considered to be solved and the corresponding run time was not included in the determination of the reported average run times for that group. Instances that were found to be infeasible were considered to be solved and their corresponding run times were included in the averages reporte for that group. Table 2.1 and Table 2.2 contain the results for the first set and second set of 200 instances, respectively. In both tables, the column labeled "Solved" shows the number of instances either solved to optimality or shown to be infeasible within the 1 hour time limit. Run times with an asterisk indicate the run time only corresponds to instances that were solved within the 1 hour time limit. Since run times for infeasible instances were included, the run times shown in both tables are biased towards lower values.

In Table 2.1 it can be seen the run times are dependent on the particular instance being solved. For example, for $|V| = 14$ the average run time was 5.18 seconds, but the maximum run time for the group was 93.4 seconds. Furthermore, as the number of vertices increases the MILP becomes significantly more computationally expensive to solve, which is to be expected. After $|V| = 20$, many instances were unable to be solved within the 1 hour time limit, resulting in skewed average run times. It should be noted that for small instances ($|V| \leq 12$), many of these instances were found to be infeasible, which is a direct consequence of how the graphs were constructed. If more arcs were included for these smaller instances, the MILP would likely be feasible more often. These infeasible solutions become common for the smaller instances because the MILP formulation only admits elementary paths for the convoy and support vehicle, making it more difficult for the MILP to be feasible when there is a small number of available arcs. The same trends can be seen in Table 2.2. For both sets, the low minimum run times correspond to infeasible instances. In both sets, typically only one instance was infeasible for $|V| \geq 12$. The lowest run time for feasible instances was typically in the range of 1 to 15 seconds in nearly all of the groups with $|V| \geq 14$.

Table 2.1: Computational Results for Set 1 ($|K| \approx 0.25|A|$)

$\|V\|$	Avg. $\|A\|$	$\|K\|/\|A\|$	Min. Run Time (s)	Avg. Run Time (s)	Max. Run Time (s)	Solved
8	22	0.25	0.0003	0.00513	0.0215	20/20
10	32	0.25	0.0004	0.0108	0.0531	20/20
12	48	0.25	0.0005	0.0972	1.10	20/20
14	64	0.25	0.0007	0.198	1.10	20/20
16	82	0.25	0.0009	0.302	0.891	20/20
18	96	0.25	0.001	0.765	4.44	20/20
20	126	0.25	0.391	117.3	2053.8	20/20
22	162	0.25	0.0021*	127.9*	1201*	**19/20**
24	184	0.25	0.321*	193.7*	1600.5*	**17/20**
26	212	0.25	0.002*	199.3*	1606.4*	**15/20**

2.5 Approximation Algorithm for the Trailing Convoy ASPP

In this section, a $(2 + \varepsilon)$-approximation algorithm for the ASPP with a trailing convoy is presented. That is, the convoy is unable to traverse impeded edges, but the convoy may take an impeded edge as the support vehicle is servicing the edge but must remain behind the support vehicle. For such a case, the travel cost of the convoy would then be the maximum unimpeded travel cost between the two vehicles plus the time required for servicing the impeded edge. Note that, unlike for the MILP formulation, both vehicles are permitted to take cycles. This is relevant for the support vehicle, as there may be situations that calls for the support vehicle to re-use an edge after depending on the structure of the graph.

2.5.1 Additional Notation

Recall X_{av} denotes a path from $a, v \in V$ for the convoy and similarly $\bar{X}_{bw}$ denotes a path from $b, w \in V$ for the support vehicle. Two convoy paths X_{av} and X_{vs} being joined to create a new convoy path X_{as} will be denoted by $X_{av} + X_{vs}$. A similar notation will be employed for the support vehicle. We say a path is a *totally impeded path* if the cost of each edge in the path is taken to be the impeded travel cost for the corresponding vehicle, irrespective of whether the edge has been

Table 2.2: Computational Results for Set 2 ($|K| \approx 0.50|A|$)

$\|V\|$	Avg. $\|A\|$	$\|K\|/\|A\|$	Min. Run Time (s)	Avg. Run Time (s)	Max. Run Time (s)	Solved
8	24	0.5	0.0003	0.008	0.051	20/20
10	32	0.5	0.0005	0.027	0.159	20/20
12	42	0.5	0.0005	0.039	0.166	20/20
14	64	0.5	0.0008	5.18	93.4	20/20
16	78	0.5	0.001	6.67	121.5	20/20
18	104	0.5	0.001	10.5	151.6	20/20
20	118	0.5	0.001	159.5	2950.4	20/20
22	156	0.5	0.002*	157.4*	1041.2*	**16/20**
24	178	0.5	0.005*	31.8*	130*	**11/20**
26	220	0.5	0.008*	192.1*	927.2*	**7/20**

previously serviced in the path. The *totally impeded cost* of a path is the sum of the edge costs of a path if it is treated as a totally impeded path and is denoted $c_I(\cdot)$. We say a path is a *totally unimpeded path* if the cost of each edge in the path is taken to be the unimpeded travel cost. The *totally unimpeded cost* of a path is the sum of the edge weights of a path if is treated as a totally unimpeded path and is denoted by $c_U(\cdot)$. For impeded edges $e \in K$, let $\rho_e = \tau_e^i - \tau_e^u$ be the cost of servicing. The *boundary of K*, denoted by $\delta(K)$, is the set of all ends of impeded edges. The vertices making up $\delta(K)$ are referred to as *boundary vertices*. As before, the words cost and time elapsed will be used interchangeably – the following work can be modified to handle more complex cost schemes, but the corresponding approximation ratio will differ as a result. If a path between two vertices does not exist, the path is set to be the empty set and the cost of that path is taken to be $+\infty$.

2.5.2 Motivating Structure for the Approximation Algorithm

Before presenting the proposed approximation algorithm for the trailing convoy ASPP, we first examine a typical solution structure to this problem. We will then use this solution structure as a guide to developing the proposed approximation algorithm.

Consider an abstracted representation of a solution to the trailing convoy ASPP shown in Figure

2.3. In Figure 2.3, a solid line represents a path for a corresponding vehicle in the graph defined for the trailing convoy ASPP. The gray region in Figure 2.3 called the "Region of Conflict“ (denoted ROC for brevity) is an abstract representation of all possible paths using at least one initially impeded edge with the first edge in the path being an initially impeded edge. The dashed border of the ROC represents all vertices that are incident with an initially impeded edge. These vertices are precisely the boundary vertices $\delta(K)$ as previously defined. A path strictly outside the ROC represents a path in the graph that does not use any impeded edges (i.e., totally unimpeded paths). In Figure 2.3, the abstract representation of a solution to the trailing convoy ASPP can be viewed as follows. Initially, the convoy and support vehicle start at vertices p and q. The convoy, taking the green path from p to the left-most red node shown in Figure 2.3, initially takes a path using only unimpeded edges until it eventually encounters a vertex that is incident with an impeded edge. This initial totally unimpeded path is represented by the green path lying outside the ROC. Once the convoy reaches this first boundary vertex, if the next edge to be taken by the convoy is an initially impeded edge, the convoy must wait for the support vehicle's assistance. With this in mind, the support vehicle's initial path goes from q to the same boundary vertex encountered by the convoy. Such a path for the support vehicle is represented by the purple path in Figure 2.3. Due to the support vehicle's capabilities, this initial path for the support vehicle need not be a totally unimpeded path. This is represented by the purple path passing through the ROC before reaching the first boundary vertex encountered by the convoy. Once the two vehicles have met at the first boundary vertex, they may then travel together using a sequence of edges, unimpeded or impeded, until they reach the final boundary vertex to be used by the convoy. The two vehicles traveling together with the support vehicle assisting the convoy by servicing edges along the way is represented by the blue path in Figure 2.3. Note that the support vehicle may have already serviced some of the impeded edges to be used by the convoy in the initial portion of the solution (purple path). This may be represented by the purple path intersecting the blue path in Figure 2.3. This was not shown in Figure 2.3 in order to reduce potential confusion with the abstraction. Once the convoy has reached the final boundary vertex used in its solution, the support vehicle's assistance is no longer required and

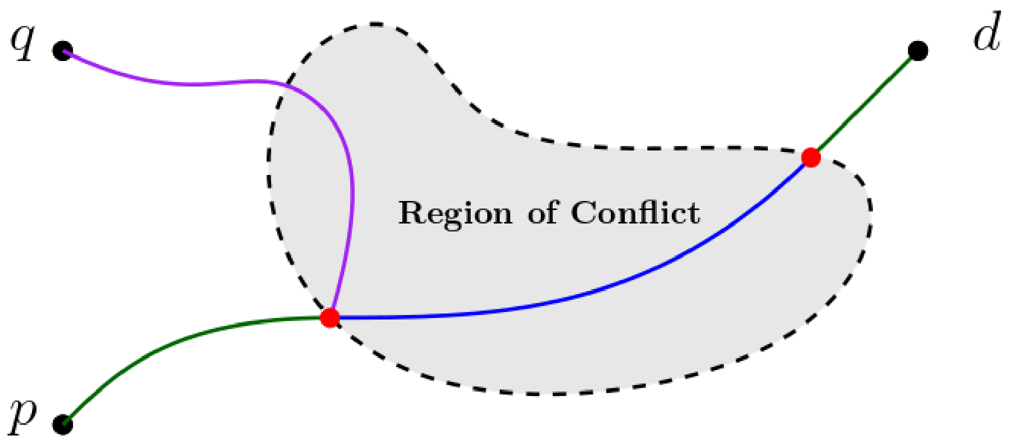


Figure 2.3: Motivating solution structure to the ASPP using an abstracted representation of the possible paths for the convoy and support vehicle.

so the support vehicle may terminate. The convoy may then take the remaining totally unimpeded path from this final boundary vertex to the destination, shown in green once again in Figure 2.3.

The motivating solution structure previously described was found to occur in many of the optimal solutions of the trailing convoy ASPP for simpler instances that could be manually solved. Additionally, this structure was seen in many of the optimal solutions to the previously described restricted ASPP from Section 2.4 (see Figure 2.2 for example). Because of this, the proposed approximation algorithm was constructed with this solution structure in mind.

2.5.3 Approximation Algorithm

The proposed algorithm (which will be referred to as Algorithm 1) is as follows.

1) Define an auxiliary graph $G_U = (V, E \setminus K)$ where the weights of the edges are set to T_e^u. Find the shortest path from p to d on G_U and store this path as X_{pd}^0.

2) Find the shortest totally unimpeded path in G_U for the convoy from p to each boundary vertex $v \in \delta(K)$ and store each such path as X_{pv}.

3) Find the shortest totally impeded path in G for the support vehicle from q to each boundary vertex $v \in \delta(K)$ and store each such path as $\bar{X}_{qv}$.

4) Find the shortest totally unimpeded path in G_U for the convoy from each boundary vertex $w \in \delta(K)$ to the destination d and store each such path as X_{wd}.

5) Create an auxiliary graph G_{max} where the weights of the edges are set to $\max\{T_e^i, \tau_e^i\}$. Then, for each pair of boundary vertices $v, w \in \delta(K)$, $v \neq w$, find the shortest path on G_{max} from v to w and store these paths as P_{vw}.

6) Define

$$t_1(v) = \max\{c_U(X_{pv}),\ c_I(\bar{X}_{qv})\} \tag{2.10}$$

and

$$c_*(P_{vw}) = \sum_{e \in P_{vw}} \left(\max\{T_e^u, \tau_e^u\} + \rho_e\right) \tag{2.11}$$

Compute

$$UB_{\bar{v},\bar{w}} = \min_{v,w \in \delta(K)} \{t_1(v) + c_*(P_{vw}) + c_U(X_{wd})\} \tag{2.12}$$

where $\bar{v}$ and $\bar{w}$ are the boundary vertices for which this minimum is attained.

7) If $c_U(X_{pd}^0) \leq UB_{\bar{v},\bar{w}}$, then set $X_{pd} = X_{pd}^0$ and do not deploy the support vehicle. Otherwise, set

$$X_{pd} = X_{p\bar{v}} + P_{\bar{v}\bar{w}} + X_{\bar{w}d} \tag{2.13}$$

to be the convoy's path and

$$\bar{X}_{q\bar{w}} = \bar{X}_{q\bar{v}} + P_{\bar{v}\bar{w}} \tag{2.14}$$

to be the support vehicle's path.

Algorithm 1 can be understood as follows. Step 1 finds the shortest path the convoy is able to take from p to d without the assistance of the support vehicle (noting we assume the convoy is unable to traverse impeded edges). This path is used as a default feasible solution. If no such path exists, it is simply set to be the empty set. Step 2 finds the shortest totally unimpeded paths for the convoy to each obstruction, which is indicated by reaching a boundary vertex $v \in \delta(K)$, without using any initially impeded edges. These paths represent the fastest the convoy may reach any obstruction without the assistance of the support vehicle. Similarly, Step 3 finds the shortest totally impeded paths for the support vehicle to each obstruction. Since all costs are positive, there will be no cycles in these shortest paths and so these paths represent the fastest the support vehicle may reach each obstruction. Step 4 finds the shortest path for the convoy from each obstruction to the destination without using any initially impeded edges. These paths represent the best-case scenario for the convoy to depart from an obstruction and reach the destination in minimum time without encountering any impeded edges along the way. For any boundary vertex without such a path, we simply set the corresponding path to be the empty set. Step 5 finds the shortest paths going from one obstruction to another obstruction where the edge weights provide a conservative estimate of the case where the convoy and support vehicle travel together on the same path P_{vw}. Since the convoy must trail behind the support vehicle on any impeded edges they may be traveling on simultaneously, the edge weight is taken to be the worst case of $\max\{T_e^u, \tau_e^u\} + \rho_e$, though impeded edges may actually be serviced before the convoy reaches them. For Step 6, suppose the convoy and support vehicle took the paths X_{pv} and $\bar{X}_{qv}$ given by Step 2 and Step 3, respectively, to the boundary vertex $v \in \delta(K)$. In such a case, the value $t_1(v)$ defined in Step 6 represents the time needed for both vehicles to be present at v, noting both vehicles are permitted to wait at any vertex in the ASPP. Next, suppose the convoy and support vehicle begin at $v \in \delta(K)$ and travel to $w \in \delta(K)$ using the same path P_{vw}. When the convoy and support vehicle are traversing an impeded edge e simultaneously, the travel cost is given by $\max\{T_e^u, \tau_e^u\} + \rho_e$. With this in mind, $c_*(P_{vw})$ represents an upper bound on the time elapsed for the convoy when both vehicles take path P_{vw}. This cost is an upper bound as an impeded edge may be serviced by the support vehicle before the convoy reaches it along this path depending on the various travel costs. We may then consider the paths $X_{pd} = X_{pv} + P_{vw} + X_{wd}$ and $\bar{X}_{qv} + P_{vw}$ for the convoy and support vehicle, respectively. That is, the vehicles both travel to the same boundary vertex v, travel together on the same path from v to another boundary vertex w, then the support vehicle terminates at w and the convoy travels from w to d without taking any initially impeded edges. The value $UB_{\bar{v}\bar{w}}$ then represents the smallest of the conservative estimates of the travel cost of the paths for all combinations of v and w. Finally, the feasible solution corresponding to the upper bound is compared to the convoy traveling without the assistance of the support vehicle and the better of the two is outputted.

If the solution returned by Algorithm 1 is not the default solution, X^0_{pd}, then this solution will have the same structure as the desired solution structure shown in Figure 2.3. The boundary vertices $\bar{v}$ and $\bar{w}$ from the returned solution of Algorithm 1 correspond to the red boundary nodes indicated in Figure 2.3. The paths $X_{p\bar{v}}$ and $\bar{X}_{q\bar{v}}$ correspond to the initial green and purple paths shown in Figure 2.3. The path $P_{\bar{v}\bar{w}}$ that both the convoy and support vehicle take corresponds to the blue path shown in Figure 2.3. Finally, the path $X_{\bar{w}d}$ corresponds to the final green path from the right-most boundary node ($\bar{w}$) to d in Figure 2.3. The computation of $UB_{\bar{v}\bar{w}}$ in Step 6 corresponds to selecting a combination of the green, purple, and blue paths in Figure 2.3 that correspond to various shortest paths as computed in Steps 2 through 4 that result in the least total cost.

2.5.4 Proof of Approximation Ratio

Let $c(X_{pd}, \bar{X}_{qw})$ be the cost (i.e., time elapsed for the convoy) of the solution outputted by Algorithm 1, where the special case $w = q$ corresponds to the support vehicle not being deployed. Let OPT denote the cost of the optimal solution to the trailing convoy ASPP. It will then be shown

$$c(X_{pd}, \bar{X}_{qw}) \leq (2 + \varepsilon)OPT \tag{2.15}$$

where

$$\varepsilon = \max_{e \in E} \frac{\tau^u_e}{T^u_e} \tag{2.16}$$

To do this, we will make use of three lemmas.

Lemma 2.5.1. By construction, we have

$$c(X_{pd}, \bar{X}_{qw}) \leq UB_{\bar{v}\bar{w}}. \tag{2.17}$$

Proof. If Algorithm 1 outputs $X_{pd} = X^0_{pd}$, then the inequality is clear from Step 7. Suppose instead

Algorithm 1 outputs the paths given by (2.13) and (2.14) in Step 7. The convoy and the support vehicle take the same path $P_{\bar{v}\bar{w}}$ from $\bar{v}$ to $\bar{w}$. Along this path, the most time required for the convoy to reach $\bar{w}$ is precisely

$$\sum_{e \in P_{\bar{v}\bar{w}}} \left(\max\{T_e^u, \tau_e^u\} + \rho_e \right)$$

which represents the every edge $e \in P_{\bar{v}\bar{w}}$ being an impeded edge and the convoy trailing behind the support vehicle on each edge as they traverse these edges simultaneously. Therefore, $c_*(P_{\bar{v}\bar{w}})$ is an upper bound the time elapsed when the two vehicles take the path $P_{\bar{v}\bar{w}}$. By construction, $t_1(\bar{v})$ is an upper bound on the earliest the convoy is able to leave $\bar{v}$ after taking the path $X_{p\bar{v}}$. The inequality then immediately follows. □

Lemma 2.5.2. Let X_{pd}^* be the convoy's path in the optimal solution to the ASPP with cost OPT. Suppose X_{pd}^* uses at least one initially impeded edge. Denote the first and last boundary vertex used in X_{pd}^* by v^* and w^*, respectively. Then

$$t_1(v^*) + \sum_{e \in X_{pd}^*} T_e^u + c_U(X_{pd}^*) \leq OPT.$$

Proof. The term on the left-hand side of the inequality represents the time elapsed if all the impeded edges on X_{pd}^* were serviced before the convoy reached them and is therefore the least cost possible for a path using impeded edges. □

Lemma 2.5.3. Let X_{pd}^* be the convoy's path in the optimal solution to the ASPP with cost OPT. Suppose X_{pd}^* uses at least one initially impeded edge and denote the first and last boundary vertex used in X_{pd}^* by v^* and w^*, respectively. Let $X_{v^*w^*}^*$ denote the sub-path from v^* to w^* in X_{pd}^*. Then,

$$\sum_{e \in X_{v^*w^*}^*} \rho_e \leq OPT.$$

Proof. OPT includes the service times along X_{pd}^*, which only occur in the sub-path $X_{v^*w^*}^*$. □

We are now prepared to prove the approximation ratio of the proposed algorithm.

Theorem 1. Let X_{pd} and $\bar{X}_{qw}$ be the convoy and support vehicle paths, respectively, that have been outputted by Algorithm 1, where $\bar{X}_{qw} = \emptyset$ is used to represent the support vehicle not being deployed. Let the cost of the optimal solution to the trailing convoy ASPP be denoted by OPT. Define

$$\varepsilon = \max_{e \in E} \frac{\tau_e^u}{T_e^u}$$

Then

$$c(X_{pd}, \bar{X}_{qw}) \leq (2 + \varepsilon) OPT$$

Proof. Let X_{pd}^* and $\bar{X}_{qw}^*$ denote the convoy and support vehicle paths, respectively, in the optimal solution to the trailing convoy ASPP with corresponding cost OPT. If the support vehicle is not deployed in the optimal solution, set $\bar{X}_{qw}^* = \emptyset$.

Suppose X_{pd}^* does not use any initially impeded edges. Then $X_{pd}^* = X^0$ and so this case is captured in Step 7 of the algorithm.

Suppose instead X_{pd}^* uses at least one initially impeded edge. Denote the first and last boundary vertex used in X_{pd}^* by v^* and w^*, respectively. By construction, we have

$$UB_{\bar{v}\bar{w}} \leq t_1(v^*) + c_*(P_{v^* w^*}) + c_U(X_{w^* d})$$

Let $P_{v^* w^*}^*$ denote the sub-path taken by the convoy from v^* to w^* in X_{pd}^*. Similarly, let $X_{w^* d}^*$ denote the sub-path taken by the convoy from w^* to d in X_{pd}^*. Expanding $c_*(P_{v^* w^*})$ and noting $X_{w^* d}$ and $P_{v^* w^*}$ are shortest paths (for graphs with appropriate edge weights outlined in Steps 4 and 5), we find

$$UB_{\bar{v}\bar{w}} \leq t_1(v^*) + \sum_{e \in P_{v^* w^*}^*} \left(\max\{T_e^u, \tau_e^u\} + \rho_e \right) + c_U(X_{w^* d}^*)$$

We further note

$$\begin{aligned}\sum_{e \in P^*_{v^*w^*}} \max\{T^u_e, \tau^u_e\} &\leq \sum_{e \in P^*_{v^*w^*}} T^u_e + \sum_{e \in P^*_{v^*w^*}} \tau^u_e \\ &\leq (1+\varepsilon) \sum_{e \in P^*_{v^*w^*}} T^u_e\end{aligned}$$

From Lemmas 2 and 3, we then have

$$UB_{\bar{v}\bar{w}} \leq (2+\varepsilon)OPT$$

and so by Lemma 1

$$c(X_{pd}, \bar{X}_{qw}) \leq (2+\varepsilon)OPT \tag{2.18}$$

as desired. □

2.5.5 Computational Results

Algorithm 1 was implemented in Python 3.7 on a Dell Inspiron (i7-8565U processor @ 1.80 GHz, 16 GB RAM). Three sets of instances were created for testing. Each instance consists of a network that was randomly generated on a 50×50 grid. For large problem instances ($|V| = 50$), vertices were connected by an edge if their Euclidean distance was less than 20 units. For smaller instances ($|V| = 25$), vertices were connected by an edge if their Euclidean distance was less than 30 units. In both cases, additional edges were added if the generated network was not originally connected. These additional edges were chosen at random from the possible edges connected separated connected components. For each instance, the convoy's unimpeded travel cost on each edge of the network was chosen to be a random integer from 5 to 20. The number of initially impeded edges for each instance was set in advance and the impeded edges were randomly selected among all edges available. The time to service each impeded edge was chosen to be a random integer from 1 to 5. The support vehicle's unimpeded travel time is specified in a different manner for each set in order to capture different aspects of Algorithm 1. This information is presented along with the corresponding set information that follows. The convoy's origin p was set to be the vertex closest to the point $(0,0)$ and the convoy's destination d was set to be the vertex closest to the point $(50,50)$. The support vehicle's origin q was set to be the vertex closest to $(0,50)$. This was done for each instance in an attempt to avoid trivial solutions corresponding to the convoy using only a few edges to reach its destination.

In addition to the three sets of instances that were created, 30 additional simulations were initially ran for the general heterogeneous case with the support vehicle's unimpeded travel time being a randomly chosen integer from 5 to 20 for each edge in the network. The results for this additional simulations are shown in Table 2.3. The LB column contains the cost of a lower bound for the trailing convoy ASPP corresponding to the shortest path in the network from p to d with all edges being treated as unimpeded or serviced edges with edge weights set to T_e^u for each edge. That is, the lower bound LB is precisely the shortest path from p to d assuming the obstructions were instantly removed for no additional cost. The c column contains the cost Algorithm 1's output for each instance. The $UB_{\bar{v}\bar{w}}$ column contains the upper bound that was computed in Step 6 of Algorithm 1. The ε simply contains the value of ε computed for each instance using (2.16). The lower bound previously described will also be used for the remaining three sets of instances that follow.

2.5.5.1 Set 1 - Heterogeneous Case

In Set 1, the support vehicle's unimpeded travel cost τ_e^u for each edge was set to a random integer from 5 to 20. Set 1 consists of 800 instances in total, with the first 400 corresponding to networks with $|V| = 50$ and the remaining 400 corresponding to networks with $|V| = 25$. For the networks with 50 vertices, the instances were split into 4 groups of 100 instances with the number of initially impeded edges set to 250, 200, 150, and 100, respectively. For the networks with 25 vertices, the instances were split into 4 groups of 100 instances with the number of initially impeded edges set to 100, 80, 50, and 40, respectively The number of initially impeded edges were chosen so that the percentage of edges requiring servicing were roughly comparable between the two instance sizes. Algorithm 1 was applied to each instance and the results were compiled into Table 2.4. In Table 2.4, the c/LB column contains the average ratio of the cost of Algorithm 1's output versus

Table 2.3: Results for 30 instances - General heterogeneous case

Instance No.	$\lvert V\rvert$	$\lvert K\rvert$	$\lvert E\rvert$	LB	c	$UB_{\bar{v},\bar{w}}$	Run Time (s)	ε
1	50	250	472	30	31	33	0.98	4.0
2	50	250	421	30	40	45	0.77	4.0
3	50	250	463	36	43	48	0.82	3.8
4	50	250	379	33	49	52	0.65	4.0
5	50	250	409	32	40	41	0.68	4.0
6	50	250	429	39	48	51	0.76	4.0
7	50	250	406	28	28	33	0.69	3.8
8	50	250	391	38	53	55	0.81	3.8
9	50	250	380	31	41	46	0.69	4.0
10	50	250	406	28	39	41	0.73	4.0
11	50	250	430	33	48	55	0.75	4.0
12	50	250	454	28	44	44	0.74	3.8
13	50	250	467	23	27	38	0.82	4.0
14	50	250	358	33	46	49	0.70	4.0
15	50	250	350	24	45	57	0.67	4.0
16	50	250	421	28	36	41	0.73	4.0
17	50	250	405	33	40	44	0.70	4.0
18	50	250	430	38	41	51	0.70	4.0
19	50	250	370	35	49	50	0.65	4.0
20	50	250	430	29	42	45	0.83	4.0
21	50	250	444	30	42	42	0.81	3.8
22	50	250	403	33	49	52	0.71	3.8
23	50	250	403	24	25	35	0.71	3.8
24	50	250	403	34	71	76	0.70	3.8
25	50	250	381	29	47	47	0.68	3.8
26	50	250	535	23	44	44	0.94	4.0
27	50	250	426	28	87	90	0.77	4.0
28	50	250	440	34	45	45	0.93	3.8
29	50	250	398	37	59	59	0.74	4.0
30	50	250	421	26	72	79	0.73	4.0

Table 2.4: Computational Results for Set 1 - General Heterogeneous Case

$\lvert V\rvert$	$\lvert K\rvert$	$\lvert K\rvert/\lvert E\rvert$	ε	c/LB	$UB_{\bar{v},\bar{w}}/c$	sec.
50	250	0.609	3.95	1.46	1.09	0.759
50	200	0.490	3.95	1.31	1.11	0.758
50	150	0.371	3.94	1.23	1.11	0.754
50	100	0.245	3.95	1.12	1.16	0.733
25	100	0.564	3.80	1.51	1.21	0.077
25	80	0.443	3.83	1.29	1.26	0.081
25	50	0.284	3.78	1.12	1.31	0.077
25	40	0.225	3.84	1.10	1.34	0.068

the computed lower bound for each set of 100 instances. The $UB_{\bar{v}\bar{w}}$ column contains the average ratio of the computed upper bound in Step 6 versus the cost of Algorithm 1's output for each set of 100 instances. The final column contains the average run time of Algorithm 1 for each set of 100 instances in seconds.

2.5.5.2 Set 2 - Homogeneous Case

In Set 2, the support vehicle's unimpeded travel cost τ_e^u for each edge was set to be equal to the convoy's unimpeded travel cost T_e^u. Set 2 consists of 800 instances in total, with the first 400 corresponding to networks with $|V| = 50$ and the remaining 400 corresponding to networks with $|V| = 25$. For the networks with 50 vertices, the instances were split into 4 groups of 100 instances with the number of initially impeded edges set to 250, 200, 150, and 100, respectively. For the networks with 25 vertices, the instances were split into 4 groups of 100 instances with the number of initially impeded edges set to 100, 80, 50, and 40, respectively. As with Set 1, the number of impeded edges were chosen so that the percentage of edges requiring servicing were roughly comparable between the two instance sizes. Algorithm 1 was applied to each instance and the results were compiled into Table 2.5. The data listed in Table 2.5 correspond to averages for each set of 100 instances and the columns are defined in the same manner as those in Table 2.4.

Table 2.5: Computational Results for Set 2 - Homogeneous Case

$\lvert V\rvert$	$\lvert K\rvert$	$\lvert K\rvert/\lvert E\rvert$	ε	c/LB	$UB_{\bar{v},\bar{w}}/c$	sec.
50	250	0.614	1.0	1.45	1.04	0.750
50	200	0.500	1.0	1.26	1.04	0.747
50	150	0.368	1.0	1.14	1.06	0.761
50	100	0.248	1.0	1.10	1.06	0.695
25	100	0.554	1.0	1.41	1.13	0.079
25	80	0.444	1.0	1.25	1.13	0.077
25	50	0.283	1.0	1.15	1.13	0.075
25	40	0.225	1.0	1.09	1.21	0.071

2.5.5.3 Set 3 - Scalar Multiple Case

In Set 3, the support vehicle's unimpeded travel cost for each edge was set to a specified scalar multiple of the convoy's unimpeded travel cost, i.e., $\tau_e^u = \varepsilon T_e^u$ for each $e \in E$. This scalar multiple, ε, was set to 0.5, 2, 3, and 4 for 400 instances each, totaling to 1600 instances for the entire set. Each set of 400 instances was split into 4 sets of 100 instances with the number of impeded edges set to 250, 200, 150, and 100, respectively. Algorithm 1 was applied to each instance the results were compiled into Table 2.6. The data listed in Table 2.6 corresponds to averages for each set of 100 instances and the columns are defined in the same manner as those in Table 2.4. All instances in Set 3 have 50 vertices.

2.5.5.4 Discussion of Results

In Tables 2.3 through 2.6, it can be seen Algorithm 1 performs well for relatively small ($|V| = 25$) and relatively large ($|V| = 50$) instances, even as the percentage of impeded edges vary. Additionally, the run time for the algorithm. Additionally, the algorithm's run time appears to be at an acceptable level for all sets. The results for Set 3 (see Table 2.6) seem to indicate the relatively quality of the output of Algorithm 1 begins to worsen as the value of ε increases. In all instances, as the percentage of impeded edges decreases, the quality of the algorithm's output increases. This

Table 2.6: Computational Results for Set 3 - Scalar Multiple Case

$\|V\|$	$\|K\|$	$\|K\|/\|E\|$	ε	c/LB	$UB_{\bar{v},\bar{w}}/c$	sec.
50	250	0.616	0.5	1.29	1.08	0.771
50	200	0.484	0.5	1.15	1.05	0.767
50	150	0.369	0.5	1.13	1.04	0.761
50	100	0.241	0.5	1.05	1.03	0.743
50	250	0.617	2.0	1.58	1.28	0.762
50	200	0.495	2.0	1.38	1.33	0.761
50	150	0.371	2.0	1.19	1.45	0.748
50	100	0.245	2.0	1.12	1.49	0.722
50	250	0.619	3.0	1.63	1.58	0.744
50	200	0.498	3.0	1.43	1.64	0.757
50	150	0.370	3.0	1.25	1.73	0.768
50	100	0.248	3.0	1.16	1.79	1.00
50	250	0.614	4.0	1.72	1.84	1.016
50	200	0.497	4.0	1.49	2.00	1.006
50	150	0.370	4.0	1.21	2.05	1.024
50	100	0.250	4.0	1.10	2.14	0.987

is expected, as the convoy's path in the optimal solution is more likely to be a totally unimpeded path (i.e., contain no impeded edges) as the number of impeded edges in the network decreases. Algorithm 1 is able to output such a solution.

2.6 Asynchronous Generalized Permanent Labeling Algorithm

2.6.1 Section Structure

In this section, a generalized version of the ASPP is considered and an asynchronous generalized permanent labeling algorithm is presented to solve this generalized ASPP. First, we consider the case where the support vehicle does not wait at any vertex unless it has chosen to terminate its journey. This variation of the ASPP occurs in situations where the support vehicle is a vehicle that is not able to easily wait at a vertex, such as in the case of a fixed-wing UAV. Afterwards, we consider the case where the support vehicle is capable of waiting at any vertex.

2.6.2 Generalized ASPP with an Unyielding Support

2.6.2.1 Problem Statement

Much of the notation and definitions directly match those given in Section 2.2. For the sake of being self-contained, this information is repeated here in addition to the small changes leading to the more general form of the ASPP. This form can be further generalized and it will be pointed out where these further generalizations may be accounted for in the presented labeling algorithm.

Let $G = (V, E)$ be an undirected, connected graph representing an impeded environment, where V is the set of vertices and E is the set of undirected edges. The vertices and edges are chosen by a designer *a priori* and are assumed to reasonably represent the environment. The convoy and support vehicle start at vertices p and q, respectively, where p and q need not be distinct. Let $d \in V$ be the destination of the convoy. The support vehicle is permitted to terminate at any vertex (a destination may be specified for the support vehicle and the presented algorithm may be modified accordingly). For the remainder of this section, cost and time will be used interchangeably, though in general a cost structure may be imposed that is not directly equal to time elapsed and the algorithm may be modified to reflect this cost structure. Let $K \subseteq E$ denote the set of impeded edges in the graph. An edge (corresponding to a path in the real environment) may be impeded due to a physical obstruction (debris, broken road, etc.) or an abstract obstruction (permission required, inspection required, etc.). The impeded edges K are assumed to be known *a priori* by the designer. The remaining edges (i.e., $E \setminus K$) are referred to as the unimpeded edges of the graph. We say an initially impeded edge has been *serviced* if either the convoy or support vehicle has completely traversed that edge. An impeded edge being serviced is equivalent to the relevant obstruction (physical or abstract) being removed in the impeded environment. It will be assumed a serviced edge remains serviced for the duration of the vehicles' journey (i.e., obstructions are permanently removed). The term impeded travel cost will refer to the cost for a vehicle to take an impeded edge that has not yet been serviced. The term unimpeded travel cost will refer to the cost for a vehicle to taken an unimpeded or serviced edge. Each edge $e \in E$ has four positive edge weights of the form $(T_e^u, T_e^i, \tau_e^u, \tau_e^i)$. Edge weights T_e^u and T_e^i are the unimpeded and impeded travel cost for the convoy, respectively, when taking edge e. Edge weights τ_e^u and τ_e^i are defined similarly for the support vehicle. The costs T_e^u, τ_e^u, and τ_e^i are taken to be finite, but T_e^i may be infinite in general. A positive infinite cost T_e^i corresponds to the convoy being unable to take the impeded edge e without the assistance of the support vehicle. In this section, we will only consider finite values of T_e^i. The presented algorithm can be easily modified to account for positive infinite cost T_e^i for select (or all) impeded edges. For unimpeded edges, we have $T_e^i = T_e^u$ and $\tau_e^i = \tau_e^u$. For impeded edges, we require $T_e^i > T_e^u$ and $\tau_e^i > \tau_e^u$, i.e., servicing an edge (removing an obstruction) always reduces the cost of taking that edge. All edges are undirected and so the cost in both directions are taken to be identical. To avoid ambiguity, if a vehicle begins to take an initially impeded edge as another vehicle is currently traversing the same edge, the travel cost of the former (i.e., the vehicle that began taking the impeded edge at a later time) will be taken to be the impeded travel cost.

The following assumptions are made: (i) the convoy and support vehicles may share vertices and edges without conflict, (ii) the two vehicles start at the same time, (iii) the two vehicles communicate at all times and information is shared in a negligible amount of time, and (iv) once the support vehicle stops moving at a vertex it will remain at that vertex for the remainder of the journey.

Assumption (iv) will only be needed for the unyielding support vehicle variation of the ASPP that is considered in this section. In the following section, assumption (iv) will be removed.

Let X_{pa} be a path from p to $a \in V$ for the convoy and $\bar{X}_{qb}$ from q to $b \in V$ for the support vehicle. A vehicle's path may also include waiting at one or more vertices. The two paths are coupled by the vehicles' interactions (i.e., servicing impeded edges) previously described. Note the support vehicle is permitted to remain at q, which is equivalent to the support vehicle never being deployed as the convoy travels to its destination. The accumulated cost of these two paths is taken to be the sum of the time elapsed for the convoy to reach a and the time elapsed for the support vehicle to reach b, including costs incurred by waiting (noting the support vehicle waiting is said to incur no additional cost out of mathematical convenience by assumption (iv)), while adhering to the previously described travel cost rules. If the support vehicle never initially leaves q, the cost of $\bar{X}_{qq}$ is zero. The accumulated cost of the coupled paths is denoted by $C(X_{pa}, \bar{X}_{qb})$. The problem considered is then to find two coupled paths X_{pd} and $\bar{X}_{qv}$, where v is any vertex in V, that minimizes $C(X_{pd}, \bar{X}_{qv})$.

2.6.2.2 Definitions and Algorithm

The principal idea behind the upcoming labeling algorithm is to use abstract objects called labels (which represent paths for the two vehicles) to generate new labels (corresponding to valid extensions of the vehicles' paths) and repeat this process until all possible paths have been exhausted. We then reduce the number of labels (paths) to be explored through additional mechanisms (dominance, filtering, and an A^*-like exploration procedure) to improve the computational performance of the algorithm.

We first define an abstract object called a label to store relevant information on the *decisions* made by both vehicles. It is important to note the label considers the decisions made, rather than the state (i.e., position) of the vehicles.

Definition 2.6.1 (Label)**.** Suppose the convoy and support vehicle have taken paths X_{pi} and $\bar{X}_{qj}$ to i and j in G in times t_i^c and t_j^s, respectively, including waiting, while accumulating a total cost C_{ij}. For each edge e that has been serviced by either the convoy or the support vehicle in their respective paths, create a tuple (e, s_e) where s_e is the time at which edge e was serviced. Let S_{ij} be the collection of such tuples (e, s_e), where $e \in K$. A label λ_{ij} is then defined as

$$\lambda_{ij} = (i, j, t_i^c, t_j^s, C_{ij}, S_{ij}). \tag{2.19}$$

□

The cost C_{ij} in Definition 2.6.1 is simply shorthand for $C(X_{pi}, \bar{X}_{qj})$ for ease of notation. For each label λ_{ij}, we define $K_S(\lambda_{ij}) = \{e \in S_{ij}\}$ to be the set of serviced edges in S_{ij} and $\hat{K}(\lambda_{ij}) = K \setminus K_S(\lambda_{ij})$ to be the set of impeded edges that have yet to be serviced by the convoy or support vehicle after taking paths X_{pi} and $\bar{X}_{qj}$, respectively.

We next introduce resource extension functions (REFs) [14] associated with each label. REFs describe how a label λ_{ij} is extended to create a new label λ_{lm} given a valid action (a valid pair of decisions by the convoy and support vehicle). Extending a label corresponds to feasible extensions of the paths X_{pi} and $\bar{X}_{qj}$ from *decisions* involving edges (i, l) and (j, m), respectively. We allow for $i = l$ or $j = m$ to represent the corresponding vehicle's position not changing, but we do not allow these cases simultaneously.

Definition 2.6.2 (REFs)**.** Suppose convoy and support vehicle paths X_{pi} and $\bar{X}_{qj}$ associated with label λ_{ij} are extended by edges $e_c = (i, l)$ and $e_s = (j, m)$, respectively, to create a new label λ_{lm}, where we allow for $i = l$ or $j = m$ but not simultaneously. Then the REFs are defined as

$$t_l^c = \begin{cases} t_i^c + T_{e_c}^u, & e_c \in E \setminus K \\ t_i^c + \min(T_{e_c}^i, T_{e_c}^u + \max(0, s_{e_c} - t_i^c)), & e_c \in K_S(\lambda_{ij}) \\ t_i^c + T_{e_c}^i, & e_c \in \hat{K}(\lambda_{ij}) \\ \max(t_i^c, t_m^s), & i = l \ \wedge \ j \neq m \end{cases} \tag{2.20}$$

$$t_m^s = \begin{cases} t_j^s + \tau_{e_s}^i, & e_s \in \hat{K}(\lambda_{ij}) \\ t_j^s + \tau_{e_s}^u, & e_s \in E \setminus \hat{K}(\lambda_{ij}) \\ \max(t_j^s, t_l^c), & j = m \ \wedge \ i \neq l \end{cases} \tag{2.21}$$

$$C_{lm} = \begin{cases} C_{ij} + (t_l^c - t_i^c) + (t_m^s - t_j^s), & j \neq m \\ C_{ij} + (t_l^c - t_i^c), & j = m \end{cases} \tag{2.22}$$

$$S_{lm} = \begin{cases} S_{ij}, & e_c, e_s \in E \setminus \hat{K}(\lambda_{ij}) \\ S_{ij} \cup \{(e_c, t_l^c)\}, & e_c \in \hat{K}(\lambda_{ij}) \ \wedge \ e_s \in E \setminus \hat{K}(\lambda_{ij}) \\ S_{ij} \cup \{(e_s, t_m^s)\}, & e_c \in E \setminus \hat{K}(\lambda_{ij}) \ \wedge \ e_s \in \hat{K}(\lambda_{ij}) \\ S_{ij} \cup \{(e_c, t_l^c), (e_s, t_m^s)\}, & e_c, e_s \in \hat{K}(\lambda_{ij}) \end{cases} \tag{2.23}$$

□

The REFs in Definition 2.6.2 encode the travel cost rules outlined in Section 2.6.2.1. REF (2.20) encodes how much time will elapse for the convoy when taking the path X_{pi} followed by edge (i, l), noting the case $i = l$ is also captured in the REF. Similarly, REF (2.21) encodes how much time will elapse for the support vehicle when taking the path $\bar{X}_{qj}$ followed by the edge (j, m), noting the case for $j = m$ is also captured in the REF. It is important to note that while t_l^c and t_m^s represent the time elapsed for the convoy and support vehicle, respectively, they need not be equal. This is a direct consequence of the fact these REFs correspond to *decisions* made by the two vehicles. For example, in the second case of (2.20), the time elapsed for the convoy when taking the edge (i, l) will differ based on whether the convoy has reached i before the support vehicle has serviced edge (i, l). This is directly captured by the minimum function in (2.20). The final case in (2.20), which corresponds to $i = l$, appears to correspond to the convoy waiting at a vertex, but this is not accurate. Rather, this final case for (2.20) represents two scenarios. In the scenario $t_i^c > t_m^s$, the convoy effectively does not yet make any decisions, as the clock associated with the support vehicle's path is lagging behind the convoy's clock. Because of this, there may still be impeded edges that can be serviced by the support vehicle by the time the convoy needs to make a decision in reality (i.e., when the clocks match). In the scenario $t_i^c \leq t_m^s$, the convoy is truly waiting in the environment at i until the support vehicle has finished acting out its most recent decision corresponding to taking edge (j, m). This asynchronous aspect of the labels and their REFs allows for complex path structural constraints, such as the interaction with impeded edges, to be represented by the labels while also allowing for mathematical machinery to be put in place to reduce the search space necessary for finding an optimal solution. The REFs (2.21) also exhibit this asynchronous behavior in the final case. REF (2.22) updates the total accumulated cost of the two paths, with the second case corresponding to only the convoy moving while the support vehicle pauses its decision-making at $j = m$. We note the support vehicle waiting at $j = m$ does not incur additional cost. In this variation of the ASPP, the support vehicle will never wait at a vertex and instead can only pause at a vertex if it has chosen to terminate its journey. We can treat the support vehicle's waiting time as incurring zero cost without affecting the search for the optimal solution as a result. If a more general cost structure is required for the ASPP, this complex cost structure would need to be encoded in (2.22) through modification and/or additional cases. This will be discussed in the next section when considering the general ASPP with a support vehicle capable of pausing at a vertex. REF (2.23) updates the impeded edges that have been serviced and the times they were serviced.

In general, there are an infinite number of possible labels due to cycles and waiting. To combat this, a dominance rule [14] is introduced to reduce the number of labels under consideration to a finite number.

Definition 2.6.3 (Dominance Rule)**.** Consider two labels λ_{ij} and λ'_{ij}. We say λ'_{ij} dominates λ_{ij} if

(D1) $t'^c_i \leq t^c_i$

(D2) $t'^s_j \leq t^s_j$

(D3) $K_S(\lambda'_{ij}) \supseteq K_S(\lambda_{ij})$

(D4) $s'_e \leq s_e, \quad \forall e \in K_S(\lambda_{ij})$

If we have equality for all the above conditions, then we consider $\lambda'_{ij} = \lambda_{ij}$ and one label can be discarded arbitrarily. □

Condition (D1) and (D2) state both vehicles have reached the same pair of vertices in less time in the paths of label λ'_{ij} when compared to paths of λ_{ij}. Since we have defined the total accumulated cost of a pair of paths to be the sum of time elapsed for each vehicle, we do not require an additional dominance condition for the cost as $C'_{ij} \leq C_{ij}$ directly follows from (D1) and (D2) being satisfied. If a more general cost structure is used, an additional dominance condition of $C'_{ij} \leq C_{ij}$ will be required. Condition (D3) states at least all the serviced edges in λ_{ij} have also been serviced in λ'_{ij}. Condition (D4) states each serviced edge in λ_{ij} has been serviced earlier or at the same time in λ'_{ij}. We say a label λ_{ij} is *non-dominated* if there exists no other label that dominates it according to the dominance rule given in Definition 2.6.3.

We first note the label corresponding to the optimal solution must necessarily be a non-dominated label, as otherwise would imply another label (and hence another feasible solution) has been found with lower cost. We now make use of the following theorem to significantly reduce the search space required for finding the label corresponding to the optimal solution.

Theorem 2. The extensions of only the non-dominated labels need to be considered to obtain the optimal solution.

Proof. Let λ'_{ij} and λ_{ij} be two distinct labels with λ'_{ij} dominating λ_{ij}. We need to show any feasible extension of λ_{ij} will be dominated by the same extension of λ'_{ij}.

Both λ'_{ij} and λ_{ij} must have the same feasible extensions. Let the labels be extended by $e_c = (i, l)$ and $e_s = (j, m)$ for the convoy and support vehicle, respectively, resulting in new labels λ_{lm} and λ'_{lm}. We allow for $i = l$ and $j = m$, but not simultaneously. Using Definitions 2.6.2 and 2.6.3, we observe the following:

- If $i \neq l$ and $e_c \notin K_S(\lambda_{ij}) \cup K_S(\lambda'_{ij})$, then we have $t'^c_l \leq t^c_l$.
- Suppose $i \neq l$ and $e_c \in K_S(\lambda_{ij}) \cap K_S(\lambda'_{ij})$. From $s'_{e_c} \leq s_{e_c}$,

$$t'^c_i + \min(T^i_{e_c}, T^u_{e_c} + \max(0, s'_{e_c} - t'^c_i)) \leq t^c_i + \min(T^i_{e_c}, T^u_{e_c} + \max(0, s_{e_c} - t^c_i)),$$

  which implies $t'^c_l \leq t^c_l$.
- Suppose $i \neq l$ and $e_c \in K_S(\lambda'_{ij}) \setminus K_S(\lambda_{ij})$. Then

$$t'^c_i + \min(T^i_{e_c}, T^u_{e_c} + \max(0, s'_{e_c} - t'^c_i)) \leq t^c_i + T^i_{e_c},$$

  which implies $t'^c_l \leq t^c_l$.
- Suppose $j \neq m$. Since $K_S(\lambda'_{ij}) \supseteq K_S(\lambda_{ij})$ and $t'^s_j \leq t^s_j$, from Definition 2.6.2 we must have $t'^s_m \leq t^s_m$.
- Suppose $i = l$ and $j \neq m$. Then

$$\max(t'^c_i, t'^s_m) \leq \max(t^c_i, t^s_m)$$

  which implies $t'^c_l \leq t^c_l$.
- Suppose $j = m$ and $i \neq l$. Then

$$\max(t'^s_j, t'^c_l) \leq \max(t^s_j, t^c_l),$$

  which implies $t'^s_m \leq t^s_m$.

- The same extension is used so clearly $K_S(\lambda'_{ij}) \supseteq K_S(\lambda_{ij})$ implies $K_S(\lambda'_{lm}) \supseteq K_S(\lambda_{lm})$.
- Suppose $e_c \in K$.
  - If $e_c \notin K_S(\lambda_{ij}) \cup K_S(\lambda'_{ij})$, from $t'^c_l \leq t^c_l$ it follows that $s'_{e_c} \leq s_{e_c}$.
  - If $e_c \in K_S(\lambda'_{ij}) \setminus K_S(\lambda_{ij})$, then $s'_{e_c} \leq t'^c_i \leq t'^c_l \leq t^c_l$ and so $s'_{e_c} \leq s_{e_c}$.
- Suppose $e_s \in K$.
  - If $e_s \notin K_S(\lambda_{ij}) \cup K_S(\lambda'_{ij})$, from $t'^s_j \leq t^s_j$ it follows that $s'_{e_s} \leq s_{e_s}$.
  - If $e_s \in K_S(\lambda'_{ij}) \setminus K_S(\lambda_{ij})$, then $s'_{e_s} \leq t'^s_j \leq t'^s_m \leq t^s_m$ and so $s'_{e_s} \leq s_{e_s}$.

We see λ'_{lm} will always dominate λ_{lm}. Since the optimal solution must be a non-dominated label, we then need only consider non-dominated labels. □

From Theorem 2, the general ASPP with an unyielding support can be solved by repeatedly extending non-dominated labels (including the label corresponding to the initial configuration) to generate new labels until there are no more unique non-dominated labels that can be generated. The optimal solution will then be the non-dominated label among with the convoy at the destination and with the least accumulated cost. This provides the framework for a basic labeling algorithm to solve the general ASPP. However, this basic labeling algorithm will have poor computational performance due to the number of non-dominated labels that need to be explored. We now introduce additional mechanisms to significantly reduce the number of non-dominated labels that need to be explored to find the optimal solution.

2.6.2.3 Filtering Through Knowledge of the Optimal Solution Structure

The domination rule in Definition 2.6.3 will remove dominated labels that can never lead to an optimal solution. However, a label generated by extending a non-dominated label has the potential to never lead to an optimal solution but not be immediately discarded by the dominance rule, as another label dominating this label may not have been generated yet. We can introduce additional filtering based on knowledge of the structure of the optimal solution to remove some of these undesirable labels. We note if a different cost structure and/or set of assumptions are used, the optimal solution structure may differ and hence so will the additional filtering mechanisms for the modified problem.

We first make a few trivial observations for the optimal solution to the general ASPP with an unyielding support:

(O1) The support vehicle will never begin to move again after pausing at a vertex.

(O2) The support vehicle, if deployed, will only terminate immediately after servicing an impeded edge.

(O3) The convoy will only wait at an end of an impeded edge it uses later.

(O4) If the convoy is waiting to use an impeded edge, it will wait until the support vehicle has serviced that edge.

(O5) The convoy will never wait at a vertex after the support vehicle has terminated its motion.

These observations are a direct consequence of the structure of the cost rules and the assumptions listed in Section 2.6.2.1. These observations may not hold for more general cost structures and variants of the ASPP involving differing constraints. These observations may be used to recognized a label generated by extending a non-dominated label may be discarded, even if this label is a non-dominated label. To do so, two Boolean variables (i.e., flags), denoted by SV_{term} and δ, are introduced. These Boolean variables are appended to the label tuple in Definition 2.6.1 and so a label λ_{ij} will be denoted by

$$\lambda_{ij} = (i, j, t_i^c, t_j^s, S_{ij}, SV_{term}, \delta) \tag{2.24}$$

These additional Boolean variables are not a part of the dominance rule and do not affect the validity of Theorem 2, as they will simply be used to remove additional (potentially non-dominated) labels that cannot possibly lead to an optimal solution. The first Boolean variable SV_{term} denotes whether the support vehicle has terminated. Observations (O1), (O2), and (O5) are captured by SV_{term}.

For the initial configuration label, SV_{term} is set to false. When extending a label with SV_{term} set to false, all feasible extensions are generated with SV_{term} set to false for each label resulting from these extensions. If any of the newly generated extensions corresponds to the support vehicle servicing an edge, generate a copy of that extension's resulting label with SV_{term} set to true. This copy with SV_{term} set to true corresponds to observation (O2) and potentially avoids unnecessary extensions of the support vehicle's path when searching for the optimal solution. In a similar vein, a copy of the initial configuration label with SV_{term} set to true is also created at the start of the algorithm to capture the case where the support vehicle never deploys. When extending a label with SV_{term} set to true, the support vehicle always remains at its current position and the convoy will always move to a new position (i.e., we do not consider any extensions with both vehicles remaining at their current position), and all resulting labels from these extensions will have SV_{term} set to true. This procedure corresponds to observations (O1) and (O5). The second Boolean variable δ is used to capture the interaction between the convoy and support vehicle in a way that eliminates redundant decisions by the convoy that can never lead to an optimal solution and will correspond to observations (O3) and (O4). The initial configuration label (including the copy for SV_{term} set to true) begins with δ set to false. When extending a label λ_{ij} with δ set to false, if the convoy's current position i is not incident with an impeded edge that has yet to be serviced, then all generated extensions will have δ set to false. Also, by observation (O3) we note we do not need to consider an extension with the convoy remaining at its current position. If instead the convoy's position i is incident with at least one impeded edge, then the extensions corresponding to the convoy remaining at i will have δ set to true. When extending a label with δ set to true, generate labels where the convoy takes an edge that has been serviced (i.e., an edge $e \in K_S(\lambda_{ij})$) and set δ to be false. In a sense, δ being set to true represents the convoy "waiting" to make a decision when given the option to take an impeded edge. However, due to the asynchronous nature of the clocks associated with the two vehicles in each label, δ being set to true does not necessarily correspond to the convoy physically waiting at i. Rather, it represents a pause in the extension of the sequence of decisions associated with the convoy.

2.6.2.4 Additional Filtering

In the basic labeling algorithm, non-dominated labels that have yet to be extended are collected into a list L_{open}. A non-dominated label is then removed from L_{open} and extended to produce new labels. The non-dominated labels from these new labels are then added to L_{open}. This process is then repeated until L_{open} is empty and the non-dominated label with the convoy at d and with the least cost is the optimal solution. While this algorithm will find the optimal solution, the search is in some sense a brute force approach as all non-dominated labels that have not been filtered are extended until there are no more non-dominated labels to extend. This leads to many redundant computations. The selection rule used to choose which labels from L_{open} are to be extended can be arbitrary in general, as the algorithm will always converge to the optimal solution irrespective of the choice of selection rule. In permanent labeling algorithms [14], it is common to select the label that was most recently added to L_{open} (known as last-in first-out or LIFO) or find the label in L_{open} with the least cost (known as best-first). The selection rule used can have a significant impact on the termination time of the algorithm, as the labels present to dominate newly generated labels will differ depending on the choice of selection rule.

The best-first selection rule can be used in conjunction with a simple check to significantly reduce the number of labels to be extended by allowing the algorithm to terminate as soon as the optimal solution's label has been created. This approach is based on the early termination criterion used in the A^* algorithm [24] used to solve the shortest path problem for a single vehicle. For any label λ_{ij}, define the heuristic cost h_i to be a lower bound on the cost for the convoy to reach d from i. In the presented implementation, h_i has been taken to be the cost of the least cost path from i to d while treating all impeded edges as unimpeded, i.e., the least cost path from i to d with edge weights set to T_e^u. We define the so-called f-cost [24] for λ_{ij} to be

$$f(\lambda_{ij}) = t_i^c + t_j^s + h_i = C_{ij} + h_i \tag{2.25}$$

That is, $f(\lambda_{ij})$ is a lower bound on the accumulated cost for the convoy to reach the destination from the state corresponding to λ_{ij}. Note that if the convoy is able to reach d from i without using any initially impeded edges, the f-cost of λ_{ij} is exactly the accumulated cost of the solution for the convoy and support vehicle with the support vehicle terminating at j. We now make use of this f-cost to identify when the algorithm may be terminated early.

Lemma 2.6.1. If a label λ_{ij} with the least f-cost is selected to be extended at every iteration of the algorithm, then the first such label with $i = d$ will correspond to the optimal solution.

Proof. From Definition 2.6.2, for any extension from λ_{ij} to a new label λ_{lm}, we must necessarily have $C_{ij} \leq C_{lm}$ by construction. From the definition of h_i (i.e., h_i corresponding to the shortest unimpeded path from i to d), we have

$$h_i \leq h_l + C_{lm} - C_{ij}$$

$$\Rightarrow h_i + C_{ij} \leq h_l + C_{lm}$$

$$f(\lambda_{ij}) \leq f(\lambda_{lm})$$

which implies the f-cost will never decrease with label extensions. Suppose the least f-cost label is selected to be extended at every iteration of the algorithm. Let λ_{dv} be the first label with the convoy at the destination to be selected by this selection rule. Therefore, all other labels that have yet to be extended must have an f-cost greater than or equal to the f-cost of λ_{dv}. Since the convoy is at d in λ_{dv}, the heuristic cost h_d associated with λ_{dv} is zero and so the f-cost is exactly equal to C_{dv}, which is the total accumulated cost of the solution corresponding to λ_{dv}. Since f-cost never decreases with label extensions, there must not exist any other label that has yet to be generated with a lower f-cost (total cost C_{dv}) and hence λ_{dv} is optimal. $\square$

From Lemma 2.6.1, we may use a best-first selection rule where the label with least f-cost is selected for extension in each iteration of the algorithm. As soon as a label with the convoy's position at d is selected for extension, the algorithm may terminated and this label may be returned as the optimal solution. The heuristic cost h_i at each vertex $i \in V$ can be computed before starting the algorithm, as h_i only depends on the unimpeded travel costs T_e^u.

We also make use of an upper bound to further reduce the number of labels to be considered. Let UB denote the value of an upper bound to the general ASPP. An initial value of UB is first found by finding the least-cost path for the convoy from p to d assuming no help from the support vehicle (i.e., each unimpeded edge has weight T_e^u and each impeded edge has weight T_e^i). We may pre-compute the least-cost path for the convoy from each $i \in V$ to d. For the presented implementation, this was done using Dijkstra's algorithm. Next, for the each label λ_{ij} selected for extension, the pre-computed least-cost path for the convoy from i to d is used to find a feasible solution. The cost of resulting feasible solution is computed with the cost of each impeded edge e in the pre-computed path from i to d adjusted if it has been serviced (i.e., if $e \in K_S(\lambda_{ij})$. If the cost of the new feasible solution is less than UB, the upper bound is updated and the corresponding best known feasible solution is stored. When a label is generated from the extension of a non-dominated label, before checking for dominance, the label's f-cost is compared to UB. If the label's f-cost exceeds the upper bound, the generated label may be discarded without requiring a dominance check. We are therefore able to discard many non-dominated labels that can never lead to an optimal solution that would have otherwise required extension in the basic algorithm. Additionally, since the best-known feasible solution is stored as the algorithm progresses, the algorithm may be terminated early by a user and a feasible solution may be returned. Without this additional step, the algorithm only produces a feasible solution when a label extension happens to have the convoy at d, which may require a large amount of time depending on the instance.

*2.6.2.5 Generalized Permanent Labeling Algorithm - A**

The generalized permanent labeling algorithm - A^* is as follows. Create initial labels $\lambda_{pq}^1 = (p, q, 0, 0, 0, , false, false)$ and $\lambda_{pq}^2 = (p, q, 0, 0, 0, , true, false)$. Initialize two lists L_{open} and D with λ_{pq}^1 and λ_{pq}^2 stored in these lists. L_{open} represents the list of labels yet to be extended (also called open labels) and D represents the list of all non-dominated labels generated so far. The following steps are then repeated until L_{open} is empty. In each iteration, select the label λ_{ij} with the least f-cost from L_{open}. For efficiency, L_{open} is kept sorted by the f-cost. If the selected label λ_{ij} has $i = d$, terminate the algorithm and return this label as the optimal solution (see Lemma 2.6.1). Otherwise, find a new feasible solution corresponding to λ_{ij} and update the UB if needed as described in Section 2.6.2.4. Extend the selected label λ_{ij} according to the REFs of Definition 2.6.2, with certain labels avoided according to the filtering due to SV_{term} and δ as outlined in Section 2.6.2.3. If the f-cost of a generated label is less than UB and it is non-dominated when compared to the labels in D, add this label to L_{open} and D. When inserting a label into the sorted list L_{open}, to break f-cost ties we select the label with the highest accumulated cost (following the procedure used in [25] for A^*). If a tie still persists, the most recently created label is used to break the tie. The trajectories for the optimal solution can be found by iterating through the predecessors of the label. This is accomplished in the coding implementation of the algorithm by simply storing for each label a reference to the parent label used to generated that label during the extension step, with the initial labels λ^1_{pq} and λ^2_{pq} having no such reference, indicating the end of the predecessor chain. For any given graph G, the connectivity can be verified in polynomial time using breadth-first search before running the algorithm to ensure there exists a feasible solution.

2.6.2.6 Complexity Analysis

We now find the complexity of $GPLA^*$. Define

$$D = UB - LB \tag{2.26}$$

where UB is the cost of the least-cost path for the convoy from p to d without deploying the support vehicle and LB is the cost of the least-cost path from p to d while treating all impeded edges as unimpeded (i.e., each edge has weight T^u_e). By construction, UB and LB do not depend on the support vehicle. Note that D represents the maximum additional cost the support vehicle can accumulate before the combined cost of both vehicles will always exceed the computed upper bound for any convoy path from p to d. Therefore, D can be used to find a bound on the number of labels that need to be explored to find the optimal solution. Define

$$k_1 = \left\lceil \frac{UB}{T_{min}} \right\rceil \tag{2.27}$$

and

$$k_2 = \left\lceil \frac{D}{\tau_{min}} \right\rceil \tag{2.28}$$

where $T_{min} = \min\{T_e^u \mid e \in E\}$ and $\tau_{min} = \min\{\tau_e^u \mid e \in E\}$, and $\lceil\cdot\rceil$ denotes the ceil function. Parameter k_1 is the maximum number of edges the convoy can take before exceeding the computed upper bound on its own. Similarly, parameter k_2 is the maximum number of edges the support vehicle can take before the convoy and support vehicle together exceed the computed upper bound.

For any vertex, the convoy has at most $n - 1$ neighboring vertices it can move to. Additionally, the convoy is able to take at most k_1 edges before exceeding the computed upper bound by construction. Therefore, the number of paths the convoy can take is upper bounded by

$$1 + (n-1) + (n-1)^2 + \ldots + (n-1)^{k_1} = \frac{(n-1)^{k_1+1} - 1}{n-2} \tag{2.29}$$

In the construction of $GPLA^*$, the convoy does not need to consider entering a "waiting" state (i.e., δ set to true) if the support vehicle has terminated its journey. As a result, the convoy can only enter a "waiting" state at most k_2 times in the algorithm for each convoy path. Therefore, an upper bound on the number of labels with δ set to true for any convoy path is $k_1 k_2$. Since this bound applies to all convoy paths, an upper bound on the number of sequences of *decisions* (i.e., moves and "waiting" states) for the convoy is found to be

$$\frac{(k_1 k_2 + 1)\left[(n-1)^{k_1+1} - 1\right]}{n-2} \tag{2.30}$$

For any vertex, the support vehicle has at most $n - 1$ neighboring vertices it can move to and the support vehicle is allotted at most k_2 moves. Additionally, the support vehicle is permitted to terminate at any vertex and hence the support vehicle has at most n options at each vertex. An upper bound on the number of sequences of decisions the support vehicle can make is then n^{k_2}. The support vehicle's decisions are coupled with the convoy's decisions and hence an upper bound on the number of labels (representing sequences of decisions for the two vehicles) is found to be

$$\frac{(k_1 k_2 + 1)\left[(n-1)^{k_1+1} - 1\right] n^{k_2}}{n-2} \tag{2.31}$$

These labels are effectively vertices of a higher-dimensional graph, which $GPLA^*$ effectively explores using A^*. Therefore, the complexity of the algorithm is $\mathcal{O}((k_1 k_2)^2 n^{2(k_1+k_2)})$, which is pseudo-polynomial [26]. From (2.31), the number of non-dominated labels to be explored is finite and so from Lemma 2.6.1 we can conclude $GPLA^*$ will terminate with the optimal solution in finite time.

We see k_1 and k_2 will have a direct impact on the overall complexity of $GPLA^*$. This is to be expected, as k_1 and k_2 are measures of the number of allotted moves for both vehicles. As the number of allotted moves for both vehicles increses, the complexity of the algorithm begins to explode. It should be noted that k_1 and k_2 can take rather large values depending on the instance considered. This is especially true for graphs where the least-cost path for the convoy from p to d without deploying the support vehicle is long and involves many initially impeded edges (i.e., UB is large and LB is relatively small in comparison).

2.6.2.7 Computational Results

$GPLA^*$ was implemented in Python 3.6 on an MSI laptop (8 core Intel i7-7700HQ processor @ 2.80 GHz with 16 GB RAM). For the analysis, a grid graph (see Figure 2.4) was used as grid graphs are typically used to represent a real-world environment, such as a warehouse, and grid graphs are easier to reproduce for verification. For all instances, the origin and destination of the convoy, p and d, were chosen to be at diagonally opposite ends of the grid to avoid trivial cases. Impeded edges can be chosen randomly or strategically so that the convoy has to traverse through at least one impeded edge to reach the destination in the optimal solution. This strategic selection can be achieved by choosing the impeded edges such that they make a cut between p and d. A cut between p and d for an instance is defined as the set of edges $\{(a,b) \in E \mid a \in A,\ b \in B\}$ for any

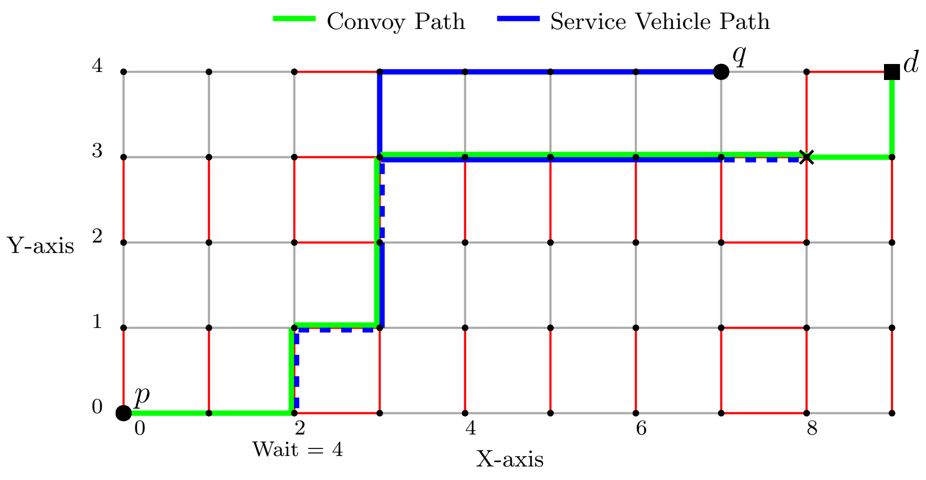


Figure 2.4: Example grid graph instance.

two disjoint sets $A, B \subset V$ such that $A \cup B = V$, $p \in A$, and $q \in B$. To illustrate the effectiveness of $GPLA^*$, it is compared with centralized A^* [27], an exact algorithm for cooperative multi-agent path planning problems modeled using MA-STRIPS [28].

The remainder of this section is as follows. First, centralized A^* is compared with $GPLA^*$. Next, $GPLA^*$ is analyzed by implementing using $GPLA^*$ for three classes of instances of the general ASPP. In Class 1, the computational limits of $GPLA^*$ as the graph size and number of impeded edges increases is examined. In Class 2, the significance of unavoidable impeded edges on the optimal solution cost is studied. Finally, in Class 3 the instances were designed to understand the impact of the support vehicle's starting position on the optimal solution cost.

2.6.2.7.1 Comparison Between GPLA* and Centralized A* To compare $GPLA^*$ and centralized A^*, instances with varied grid sizes and cuts were created. Table 2.7 shows the results for these instances. The starting position for the support vehicle was chosen randomly for each instance. For the convoy, the unimpeded travel cost, T_e^u, was randomly chosen from the range $[10, 15]$ for all edges and the impeded travel cost, T_e^i, was randomly chosen from the range $[40, 50]$ for the impeded edges. For the support vehicle, the unimpeded travel cost, τ_e^u, was set to be 1 for all edges and the impeded travel cost, τ_e^u, was randomly chosen from the range $[2, 6]$. For unimpeded edges, we set $T_e^u = T_e^i$ and $\tau_e^i = \tau_e^u$. This cost structure was chosen to encourage collaboration between the vehicles while keeping the decision-making non-trivial.

The results in Table 2.7 highlight the effectiveness of $GPLA^*$ over the centralized A^* method. Each row represents the averages over 50 randomly generated instances using the previously described structure. The columns labeled T_{GPLA^*} and T_{A^*} represent the average computational times for $GPLA^*$ and centralized A_c^*, respectively. The columns labeled O_{GPLA^*} and $O_{A_c^*}$ shows the average number of extended states for both algorithms. From Table 2.7, it can be seen $GPLA^*$ outperforms centralized A^* both in the computational time needed to find an optimal solution and in the number of states that needed to be explored to do so. The key difference between these two algorithms lies in the extension step. Centralized A^* only considers a single agent's action for each extension step. Conversely, $GPLA^*$, using the REFs presented in Definition 2.6.2, considers the actions of both agents *simultaneously*, resulting in fewer label extensions required. This in combination with the additional filtering, domination checks, and avoiding generating redundant states, all leads to better performance.

Table 2.7: Computational time comparison: $GPLA^*$ vs centralized A^*

Grid-Size	Cuts	T_{GPLA^*}	$T_{A_c^*}$	O_{GPLA^*}	$O_{A_c^*}$
6×6	2	0.05 s	0.59 s	914	2300
8×8	2	0.60 s	3.52 s	6008	15768
10×10	2	2.45 s	22.9 s	15723	46404
3×15	2	0.10 s	0.42 s	1623	4738
3×15	3	1.80 s	42.2 s	7787	28351
3×15	4	7.96 s	149 s	16972	66160

2.6.2.7.2 Class 1 Instances To test the computational limits of $GPLA^*$, we generated instances from 6×6 to 10×10 grid sizes. For each instance, the impeded edges were chosen randomly and the same cost structure as in Section 2.6.2.7.1 was used. The fraction of impeded edges is denoted by $|K|/|E|$. For each grid size, the fraction of impeded edges were varied to be 0.1, 0.2, 0.3, 0.4, and 0.5. We generated 50 random instances for each grid size and fraction of impeded edges. The starting position for the support vehicle was chosen randomly for each instance. An instance is said to be *successful* if it terminated with the optimal solution within 900 seconds. The success rate, γ, is defined as the fraction of all successful instances over the total number of instances. For a majority of the instances with $|K|/|E| = 0.1$ and 0.2, we observed that the impeded edges did not form a cut and so the convoy had at least one unimpeded path from p to d that was found to be optimal. Therefore, we do not include those results in the discussion.

The results for Class 1 are shown in Table 2.8. In Table 2.8, each entry in the time columns represents the average computation time of successful instances along with the standard deviation in parentheses. The success rate decreases with increasing grid size and increasing fraction of impeded edges. The average computation time and the standard deviation increase with increasing grid size and fraction of impeded edges. For the cases with a relatively low success rate ($\gamma < 0.5$), the average computation time may not follow the same trend as the samples are skewed. From this set of instances, we see the computation time is significantly affected by the number of impeded edges. This is to be expected, as $GPLA^*$ must produce additional labels whenever a label is such that the convoy's position is an end of an impeded edge. Similarly, an additional label is also generated whenever an extension is such that the support vehicle services an impeded edge. As the number of impeded edges increases, the number of additional labels generated will begin to explode. The upper bound cost filter introduced for $GPLA^*$ significantly reduces the number of redundant labels generated. However, as the grid size increases, the upper bound becomes less tight and so the algorithm begins to fail to discard redundant labels early on. This behavior can be seen by noticing as the fraction of impeded edges increases for a fixed grid size, the computation time begins to grow at an exponential rate. Conversely, for a fixed fraction of impeded edges, the computation time grows more slowly as the grid size increases.

2.6.2.7.3 Class 2 Instances For this set of instances, we wish to determine how the presence of unavoidable impeded edges affects the optimal solution cost. To do this, we used a fixed grid

Table 2.8: Class 1 Results

	$\lvert K\rvert/\lvert E\rvert = 0.3$		$\lvert K\rvert/\lvert E\rvert = 0.4$		$\lvert K\rvert/\lvert E\rvert = 0.5$	
Grid Size	time (sec.)*	γ	time (sec.)*	γ	time (sec.)*	γ
6×6	0.1 ± 0.2	1.00	5.6 ± 21	1.00	27 ± 96	0.96
7×7	2.3 ± 11	1.00	39 ± 142	0.94	93 ± 180	0.66
8×8	10 ± 48	0.98	55 ± 146	0.84	100 ± 184	0.36
9×9	39 ± 96	0.98	72 ± 147	0.68	128 ± 255	0.16
10×10	51 ± 154	0.88	112 ± 223	0.48	127 ± 132	0.12

* time indicates the average computation time ± standard deviation.

size of 3×15 with the same cost structure defined in Section 2.6.2.7.1 for each instance. We then generated cuts to introduce unavoidable impeded edges for the convoy. The narrow grid structure was chosen as it is easier to produce cuts with fewer edges, which keeps the computational time reasonable as we increase the number of cuts. The support vehicle's starting position was chosen randomly for each instance. The number of cuts was varied from 1 to 5. We generated 50 random instances for each cut size.

The computational results for this set of instances is shown in Table 2.9. Each row in Table 2.9 represents an average value over the 50 instances for a set number of cuts. Column $|K|/|E|$ indicates the fraction of impeded edges. An upper bound, UB, was computed by having the convoy take the shortest path from p to d without any assistance from the support vehicle. The gap between the cost of the optimal solution to the ASPP and UB gives an indicator of the benefit of deploying the support vehicle. Similarly, a lower bound, LB, was computed by treating all impeded edges as unimpeded and finding the shortst path for the convoy from p to d only using the unimpeded cost for all edges. Columns OPT/UB and OPT/LB represents the ratio of the optimal cost, OPT, against the upper and lower bound costs, respectively. We see as the number of cuts increases, the OPT/UB ratio decreases. This shows the effectiveness/benefit of the support vehicle in an impeded environment, especially as the number of unavoidable impeded edges increases. Column $\sigma(OPT)$ shows the standard deviation of the optimal cost. The increasing trend of $\sigma(OPT)$ indicates the optimal solution cost is sensitive to the number of unavoidable impeded edges.

Table 2.9: Class 2 Results

Cuts	$\|K\|/\|E\|$	OPT	OPT/UB	OPT/LB	$\sigma(OPT)$
1	0.05	191	0.89	1.05	6.19
2	0.10	196	0.82	1.08	6.48
3	0.14	201	0.77	1.10	6.94
4	0.18	205	0.71	1.12	6.90
5	0.22	207	0.68	1.14	7.56

2.6.2.7.4 Class 3 Instances For this set of instances, we examine the impact of the starting position of the support vehicle on the optimal solution. For each instance, we again use a 3×15 grid, but we impose a fixed cost structure for all instances. All edges were assigned costs $T_e^u = 10$ and $\tau_e^u = 1$ and the impeded edges were all assigned costs $T_e^i = 40$ and $\tau_e^i = 6$. The convoy's starting position and destination were fixed at diagonally opposite ends with positions $(0, 0)$ and $(14, 2)$ (see Figure 2.5). This cost structure and choice of positions were chosen to encourage collaboration between the convoy and the support vehicle. We generated 50 different instances by randomly generating 3 cuts for each instance. For each instance, we computed the optimal solution cost for each of the 45 possible starting positions for the support vehicle. We then computed the average optimal cost across all 50 instances for each of the 45 support vehicle starting positions.

Figure 2.5 shows the average cost across 50 instances for each starting position in the fixed grid. The convoy starting position and destination are marked at $(0, 0)$ and $(14, 2)$ in Figure 2.5. Gaussian interpolation was used to generate a continuous map. We observe that the minimum and maximum average costs are achieved at vertex $(3, 1)$ and $(14, 0)$, respectively. Figure 2.5 shows the starting position of the support vehicle will have a noticeable impact on the optimal solution. The magnitude of the impact will depend on the numerical values of the costs for the edges. We

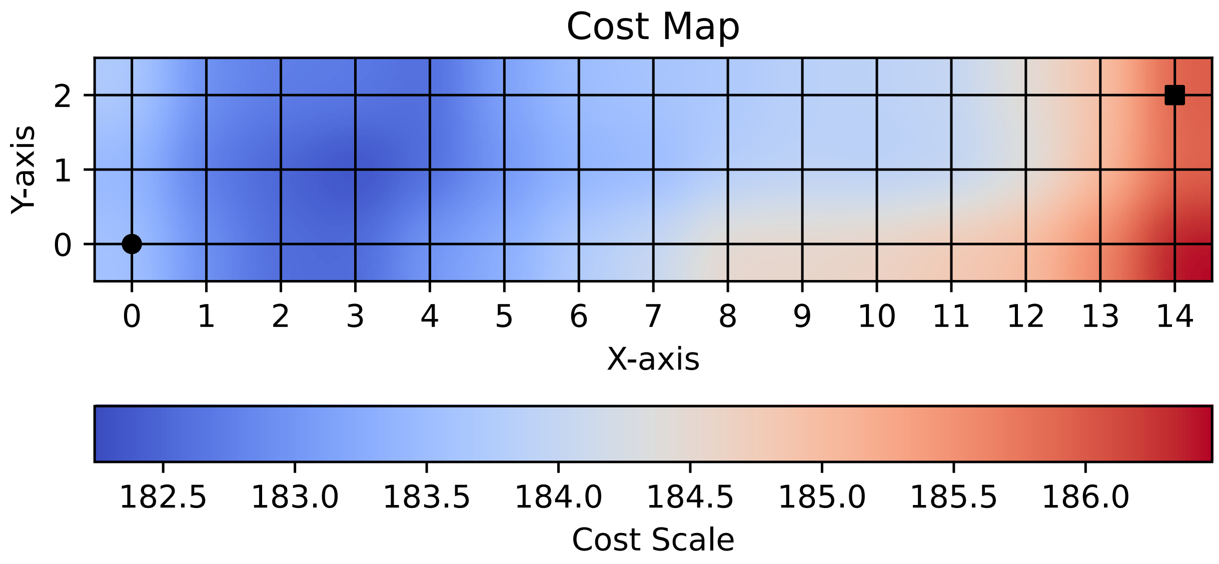


Figure 2.5: Average cost map as a function of the starting position of the support vehicle.

also note the best starting position for the support vehicle in this set of instances does not coincide with the starting position of the convoy. This is to be expected. If the support vehicle starts some distance away from the convoy, it may be able to attend to more significant impeded edges early on before the convoy is able to reach them.

2.6.3 Generalized ASPP

2.6.3.1 Problem Statement

We now consider the generalized ASPP with a support vehicle that is capable of waiting at a vertex for any duration of time. All other notation and definitions from Section 2.6.2.1 introduced for the unyielding support variation will apply. As before, we will assume (i) the convoy and support vehicles may share vertices and edges without conflict, (ii) the two vehicles start at the same time, and (iii) the two vehicles communicate at all times and information is shared in a negligible amount of time. Unlike in the unyielding support variation, we will assign a cost for the support vehicle waiting at a vertex. For simplicity, the waiting cost will be the time elapsed. A more complex cost structure can be considered and the following REFs may be modified accordingly.

2.6.3.2 Extending Previous Results

This variation of the generalized ASPP can be solved in a manner similar to the unyielding support variation. The REFs in Definition 2.6.2 would need to be modified by adjusting how the support vehicle's time, t^s_m, changes by considering the case where the support vehicle waits for the convoy to service an edge. In doing so, the support vehicle's time update will mirror the convoy's time update in Definition 2.6.2. By modifying the REFs Definition 2.6.2, all possible feasible solutions to the generalized ASPP can be generated. Afterward, an equivalent result to Theorem 2 can be shown to reduce the number of labels that need to be considered to find the optimal solution to the ASPP.

As of writing this dissertation, results for this variation of the generalized ASPP are not yet available and so discussion on this variation is limited in this dissertation to avoid being unintentionally misleading. It is important to note the additional label filtering techniques shown in Section 2.6.2.3 and Section 2.6.2.4 do not all immediately apply to this variation of the generalized ASPP. For example, the labeling algorithm presented for the unyielding support vehicle variation does not permit a label with the support vehicle waiting at a vertex to be extended in a manner that results in the support vehicle moving to a new vertex (i.e., once the support vehicle pauses, it remains at that vertex for the remainder of the journey).

3. GLOBAL OPTIMIZATION ALGORITHM FOR MIXED-INTEGER NONLINEAR PROGRAMS WITH TRIGONOMETRIC FUNCTIONS

3.1 Introduction

Optimization problems in many applications such as chemical process networks [29, 30, 31, 32], energy systems [33], and wastewater treatment [34] to name a few, are traditionally modeled as Mixed Integer Nonlinear Programs (MINLPs). MINLPs are mathematical programs that include both continuous and discrete decision variables and the objective function and/or the constraints may be nonlinear and possibly nonconvex in general. Algorithms to solve this class of optimization problems to global optimality has garnered extensive attention from both the academia and the industry and has resulted in the development of both open-source (Couenne [12] and SCIP [35]) and commercial solvers (BARON [36], LINDOGlobal [11] and ANTIGONE [37]) for the same. Each of these solvers are specialised and solve a subset of factorable MINLPs [38] where nonlinearities arise due to a certain class of functions such as multilinear, logarithmic, exponential, etc. Since the focus of this article is on MINLPs with trigonometric functions, we remark that among all these global optimization solvers, to the best of our knowledge, only Couenne and LINDOGLOBAL implement global optimization algorithms that are equipped to solve MINLPs with trigonometric terms. The method presented in this chapter differs from the methods used by these solvers.

All algorithmic approaches to solving MINLPs to global optimality contain two main features: convex relaxations and search. For instance, any of the aforementioned solvers for MINLPs first isolate each non-convex term in the problem and construct convex over- and under-estimators for these terms. Doing so for every non-convex term in the MINLP yields a convex relaxation of the MINLP which in turn is solved to optimality to provide a bound to the optimal objective value of the MINLP. This process is typically combined with a standard search procedure such as spatial branch-and-bound (sBB) [39, 40] to explore the full space of feasible solutions to the MINLP. Furthermore, at each node of the sBB tree, a local solve or heuristics are applied to keep generating feasible solutions to the MINLPs and to keep track of the optimality gap. Mathematical properties of sBB such as consistency and exhaustiveness [39, 41] in turn provide a guarantee of convergence of the algorithm to global optimality. Each solver differs in how the convex relaxations are constructed, the strength of the respective convex relaxations, and the support for the different types of non-convex terms it provides, resulting in disparate computational performance for the same MINLP.

More recently, global optimization algorithms for MINLPs that rely on solving a Mixed Integer Linear Program (MILP) at each iteration have gained considerable interest [42, 43, 44, 45, 46]. The main motivation for developing MILP-based algorithms for MINLPs has been the meteoric improvement in the speed of off-the-shelf MILP solvers [47]. While the sBB-based algorithms rely on constructing and solving convex relaxations of non-convex structures, MILP-based algorithms rely on constructing and solving piecewise convex or polyhedral relaxations of non-convex structures [48]. The search procedure for MILP-based methods are domain partitioning schemes which partition the domains of the variables involved in the non-convex terms [46]. Similar to the sBB search procedure, the domain partitioning schemes require the mathematical properties of exhaustiveness and consistency to ensure convergence. It has been shown in recent work [46, 49] that the MILP-based methods perform comparably with sBB-based methods and sometimes even outperform the sBB counterpart for MINLPs with multilinear and quadratic functions. In this chapter, we develop piecewise polyhedral relaxations for non-convexities arising from trigonometric terms and embed these relaxations in an MILP-based algorithmic framework for global optimization of MINLPs. To the best of our knowledge, this is the first work to develop a MILP-based global optimization algorithm for MINLPs with trigonometric functions. This framework can be implemented into existing MINLP solvers to handle trigonometric terms which originally could not be considered. Additionally, the presented framework can be extended to factorable MINLPs with differentiable, periodic terms, with trigonometric terms falling into a subset of this class of MINLPs.

This chapter is divided into five major parts: (i) development of polyhedral and piecewise polyhedral relaxations for univariate trigonometric functions, (ii) integration of these relaxations into an MILP-based algorithmic framework for global optimization of MINLPs, (iii) exploration of different adaptive partitioning schemes for variable domain partitioning, (iv) introduction of a reformulation for periodic functions with bounded inputs to reduce the overall search space of the MILP relaxations, and (v) illustration of the effectiveness of the overall algorithm to solve a MINLP that models a path planning problem for a single fixed-wing vehicle.

3.1.1 Problem Statement

We consider the problem of finding the globally optimal solution* of an MINLP composed of linear, trigonometric, and bilinear terms that has been factored into the form

$$
\begin{aligned}
(\mathcal{F}) \qquad \text{minimize} \quad & c^T x + d_1^T y + d_2^T z && (3.1a)\\
\text{subject to} \quad & A_1 x + A_2 y + A_3 z \leq 0 && (3.1b)\\
& y_i = f_i(x), \qquad i = 1, \dots, n_t && (3.1c)\\
& z_j = g_j(x, y), \quad j = 1, \dots, n_b && (3.1d)\\
& x \in X \subseteq \mathbb{Z}^{n_i} \times \mathbb{R}^{n-n_i} && (3.1e)
\end{aligned}
$$

In (3.1), the vector x consists of continuous and integer-valued variables and represents the variables of the original MINLP. The vectors c, d_1, and d_2 and the matrices A_1, A_2, and A_3 are of appropriate size and are the result of the factoring procedure used to convert the original MINLP into the form of (3.1). The functions $f_i : I_i \subset \mathbb{R} \mapsto \mathbb{R}$ for $i = 1, \dots n_t$, are univariate, trigonometric functions that are differentiable and bounded over a closed interval I_i, where n_t is the number of unique trigonometric terms after factoring. The functions $g_j : R_j \subset \mathbb{R}^2 \mapsto \mathbb{R}$ for $j = 1, \dots, n_b$ are bilinear terms which may consist of the original variables x for the MINLP and/or auxiliary variables y that have been added after factoring. The functions g_j are each defined over a closed rectangle R_j and n_b represents the number of unique bilinear terms after factoring. It is assumed the feasible space of (3.1) is non-empty. Note that multilinear terms may also be considered by recursively defining bilinear variables. In doing so, g_j would also be a function of the introduced z variables.

*Global optimality is defined numerically by a specified tolerance, ϵ.

The proposed approach to finding the globally optimal solution to $\mathcal{F}$ is by first introducing MILP relaxations for the trigonometric terms (3.1c) and the bilinear terms (3.1d). For the univariate trigonometric functions f_i with domain I_i, we partition I_i using information of the convexity of f_i over sub-intervals of I_i. An MILP relaxation of each f_i is then constructed using its corresponding partition where the MILP relaxation is the disjunctive union of triangles containing f_i. For the bilinear terms g_j with domain R_j, we partition R_j by dividing the domain of one variable and leaving the second variable's domain untouched. An MILP relaxation of each g_j is then constructed using its corresponding partition where the MILP relaxation is the disjunctive union of tetrahedrons containing g_j. All the relaxations are formulated using a so-called "incremental formulation" [50]. Once all trigonometric and bilinear terms have been relaxed, the resulting MILP is solved to find a lower bound. This lower bound is successively tightened by refining some or all of the partitions used to construct the MILP relaxations of the trigonometric and bilinear terms. This process is repeated until the gap is sufficiently small. It is assumed a method exists for determining an upper bound given a lower bound. If no such method exists, a standard local nonlinear programming (NLP) solver can be used to find a feasible solution and hence, an upper bound to the optimal objective value. This algorithm can be integrated into existing solvers to handle a larger class of MINLPs than those considered in this paper.

3.1.2 Structure of Chapter

The remainder of this chapter is organized as follows. Section 3.2 presents a simplified flowchart of the proposed algorithm. This overview is meant to serve as a guide for the reader. Section 3.3 presents preliminary definitions which will be used throughout the remainder of the chapter. Section 3.4 presents the MILP relaxations for trigonometric and bilinear terms. Section 3.5 presents an adaptive partition refinement scheme that will be used to successively tighten the MILP relaxation until global optimality is reached. Section 3.6 presents a procedure for representing a periodic function with a bounded input using an alternative, equivalent formulation that reduces the overall search space required of a MILP relaxation with such a function in its constraints. Section 3.7 presents a motivating example in the form of the path planning of a vehicle moving in the plane through a specified sequence of points subject to kinematic constraints. Section 3.8 presents computational results.

3.1.3 Relevance to the Assisted Shortest Path Problem

Before continuing, it should be noted the work presented in this chapter may be directly related to the ASPP by noting the problem of determining appropriate edge weights for the graph chosen by a designer for the ASPP can be posed as a MINLP of the kind considered in this chapter. Despite this, the majority of this chapter is presented as independent material to highlight the effectiveness of the proposed algorithm for a larger class of problems. The example problem shown in Section 3.7 directly relates to the problem of determining appropriate edge weights for the ASPP.

3.2 Initial Overview of the Algorithm

A simplified flowchart of the algorithm is shown in Figure 3.1. The algorithm first takes an MINLP composed of linear, trigonometric, and bilinear terms in the factored form (3.1). An initial partition is created for each trigonometric and bilinear term in the factored MINLP. The procedure for constructing these initial partitions is the subject of Sections 3.3, 3.4, and 3.5. Using these initial partitions, an MILP relaxation for each trigonometric and bilinear term in the factored MINLP is constructed. The procedure for constructing MILP relaxations of trigonometric and bilinear terms given a set of partitions is the subject of Section 3.4. The MILP relaxation is then solved using a standard MILP solver (such as CPLEX [51] or Gurobi [23]) to obtain a lower bound on the original MINLP's optimal solution. It is assumed a method is available to find a feasible solution to the MINLP. This method is then used to find a feasible solution, the best known upper bound is updated, and the resulting relative gap is computed. If the gap is sufficiently small (using a user-specified gap tolerance ε), the algorithm terminates. If the gap is too large, some or all partitions are selected for refinement to tighten the relaxation. The procedure used for selecting partitions for further refinement is referred to as a refinement strategy. Refinement strategies are the subject of Section 3.5.3. Each of the selected partitions are then refined. The procedure used for refining a given partition is referred to as a refinement scheme. Refinement schemes are the subject of Section

3.5.2. Once each of the selected partitions are further refined, a new, tighter MILP relaxation is constructed using these partitions. The tighter MILP relaxation is solved and this process is repeated until the gap is less than ε.

3.3 Preliminaries

Terminology that will be used throughout the remainder of the chapter is presented in this section. The terminology used closely follows the work of [48].

Definition 3.3.1. Given a closed interval $[l, u] \subset \mathbb{R}$, a *partition* p of $[l, u]$ is an ordered sequence of distinct real numbers $(x_0, x_1, \dots, x_m)$ such that $l = x_0 < x_1 < \dots < x_m = u$. We denote the set of partition points of p by $\mathcal{P}(p) = \{x_0, x_1, \dots, x_m\}$.

Definition 3.3.2. Given a nonlinear, bounded, and differentiable function
$f : [l, u] \mapsto \mathbb{R}$, we say $b \in [l, u]$ is a *break point* of f if at b the function changes from being convex to concave or vice-versa.

Definition 3.3.3. Given $f : [l, u] \mapsto \mathbb{R}$, we say a partition p is *admissible* if p contains all break points of f in $[l, u]$.

Definition 3.3.4. Given a function $f : [l, u] \mapsto \mathbb{R}$, the partition p^0 of $[l, u]$ is referred to as a *base partition* of $[l, u]$ for f if (i) it is admissible, (ii) for any sub-interval $[x_i, x_{i+1}]$ defined by the partition, $f'(x_i) \neq f'(x_{i+1})$, and (iii) $|\mathcal{P}(p^0)|$ is minimum.

Definition 3.3.5. Given $f : [l, u] \mapsto \mathbb{R}$ and admissible partition p of $[l, u]$ for f, we say the partition q is a *valid refinement* of p if $\mathcal{P}(p) \subset \mathcal{P}(q)$ and for each sub-interval $[x_i, x_{i+1}]$ defined by q we have $f'(x_i) \neq f'(x_{i+1})$.

As an example, consider the function $f(x) = \sin x$ for $x \in [0, 2\pi]$ and the partition $p = (0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}, 2\pi)$. The break points of f in the domain $[0, 2\pi]$ are 0, π, and 2π and so p is admissible. We also see the slope conditions (ii) in Definition 3.3.4 are satisfied. However, $\mathcal{P}(p)$ satisfying conditions (i) and (ii) in Definition 3.3.4 is not of minimum size and so p is not a base partition. The partition $p^0 = (0, \pi, 2\pi)$ is a base partition of $[0, 2\pi]$ for f and so p is a valid refinement of

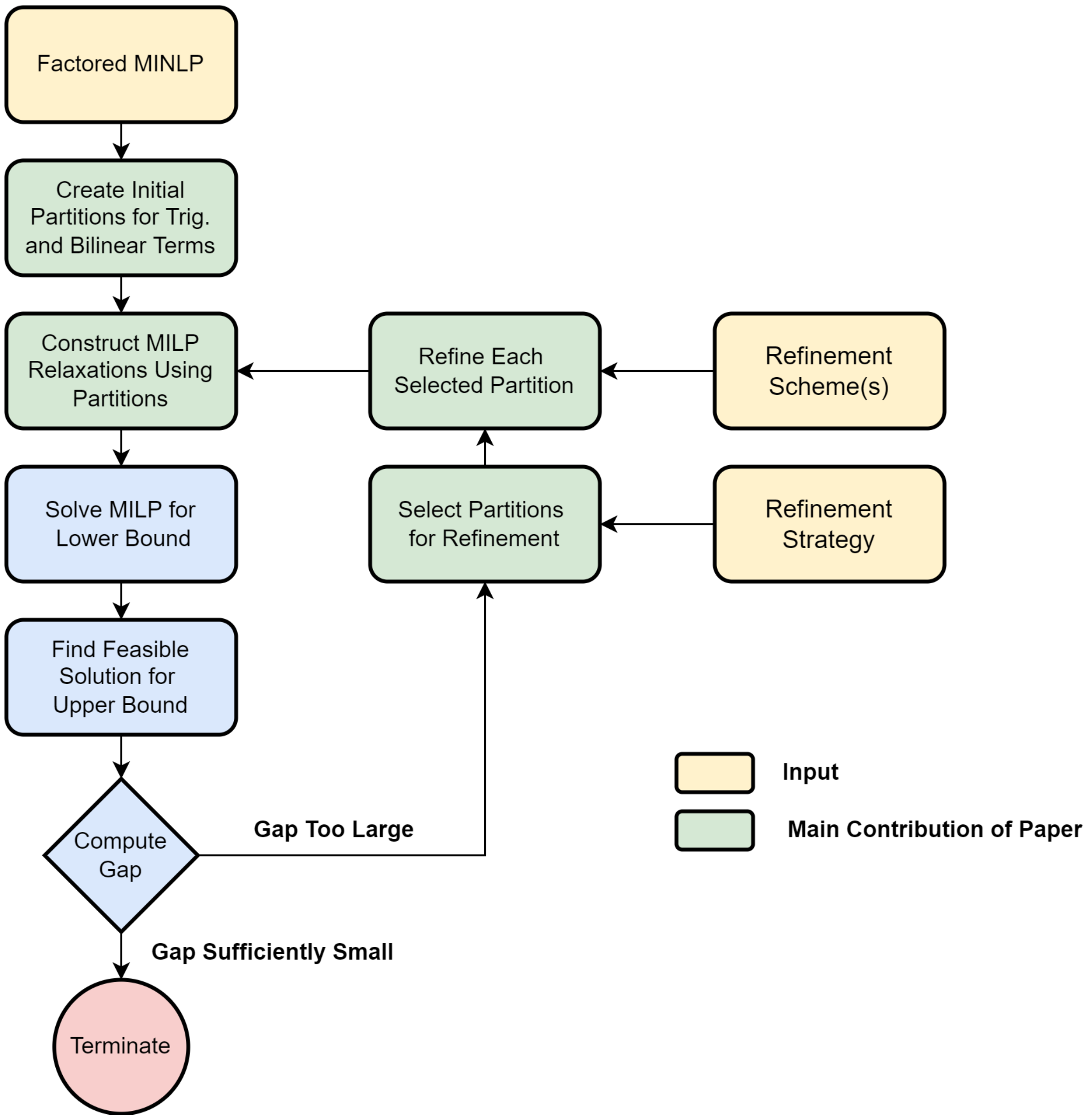

Factored MINLP
Create Initial Partitions for Trig. and Bilinear Terms
Construct MILP Relaxations Using Partitions
Solve MILP for Lower Bound
Find Feasible Solution for Upper Bound
Compute Gap
Gap Too Large
Gap Sufficiently Small
Terminate
Refine Each Selected Partition
Select Partitions for Refinement
Refinement Scheme(s)
Refinement Strategy
Input
Main Contribution of Paper

Figure 3.1: Simplified flowchart for the algorithm.

p^0. In general, to construct a base partition we start with all break points and only add additional points to the partition when the slope condition (ii) in Definition 3.3.4 for a sub-interval $[x_i, x_{i+1}]$ is not satisfied. As a result, a base partition is not unique in general.

3.4 MILP Relaxations

In this section a method for constructing MILP relaxations of univariate trigonometric terms (3.1c) and bilinear terms (3.1d) in $\mathcal{F}$ is presented. Once these nonlinearities have been relaxed, the resulting MILP can be solved using a standard MILP solver.

3.4.1 Trigonometric Terms

Consider a univariate, trigonometric function $f : [x^L, x^U] \mapsto \mathbb{R}$, where f is differentiable and bounded over $[x^L, x^U] \subset \mathbb{R}$. We would like to construct a MILP relaxation of the constraint $y = f(x)$ with $x \in [x^L, x^U]$, which can then be used for the constraints (3.1c) in $\mathcal{F}$. Define an admissible partition $p = (x_0, x_1, \dots, x_m)$ of $[x^L, x^U]$ for f where p is a valid refinement of a base partition p^0 or is p^0 itself. Let $[x_i, x_{i+1}]$ be a sub-interval defined by p. Define the following:

$$h_i(x) = f(x_i) + f'(x_i) \cdot (x - x_i) \tag{3.2}$$

$$h_{i+1}(x) = f(x_{i+1}) + f'(x_{i+1}) \cdot (x - x_{i+1}) \tag{3.3}$$

$$t_{i+1}(x) = \begin{cases} \max\{h_i(x),\ h_{i+1}(x)\}, & \text{if } f \text{ is convex in } [x_i, x_{i+1}] \\ \min\{h_i(x),\ h_{i+1}(x)\}, & \text{if } f \text{ is concave in } [x_i, x_{i+1}] \end{cases} \tag{3.4}$$

$$s_{i+1}(x) = f(x_i) + \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} \cdot (x - x_i) \tag{3.5}$$

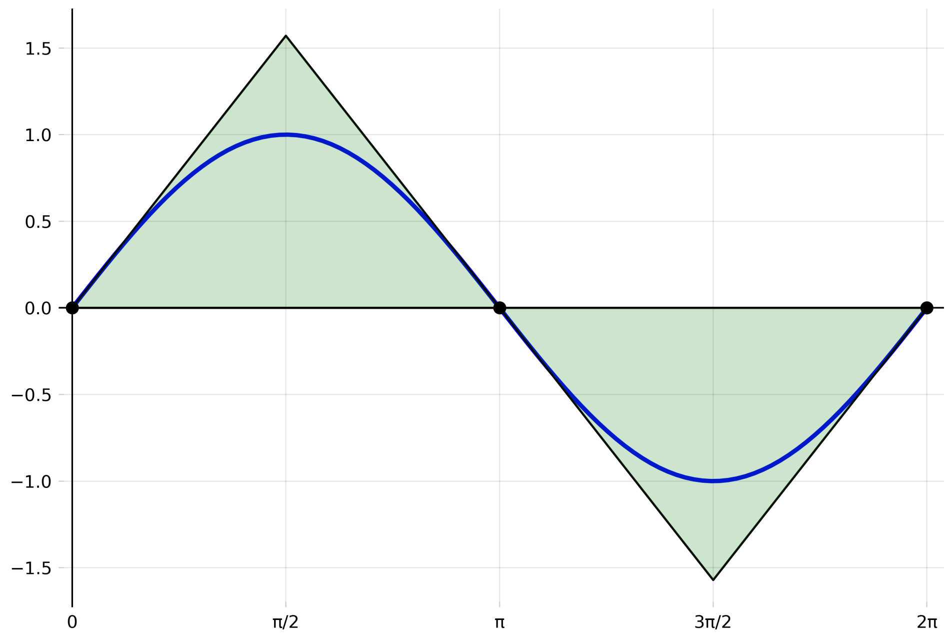
Equations (3.2) and (3.3) define tangent lines at $f(x_i)$ and $f(x_{i+1})$, respectively. When f is convex (resp. concave) the tangent lines lie below (resp. above) the function and so a simple underestimate (resp. overestimate) of f over the sub-interval $[x_i, x_{i+1}]$ is obtained by taking the maximum (resp. minimum) of these two tangents, corresponding to (3.4). Equation (3.5) defines a secant line from $f(x_i)$ to $f(x_{i+1})$, which is above (resp. below) f when it is convex (resp. concave) in the sub-interval $[x_i, x_{i+1}]$. For each partition point $x_i \in \mathcal{P}(p)$, let $v_i = f(x_i)$ be the corresponding point on the curve. For each sub-interval $[x_i, x_{i+1}]$, let $v_{i,i+1}$ be the point of intersection for the tangent lines defined by (3.2) and (3.3).

We now make a few key observations. For any sub-interval $[x_i, x_{i+1}]$ defined by p, the points v_i, $v_{i,i+1}$, and v_{i+1} form a triangle which contains the curve f over this sub-interval. This can be seen by noting the partition p is admissible and so f is either convex or concave in each sub-interval. The intersection point $v_{i,i+1}$ is guaranteed to exist for each sub-interval as p satisfies the slope condition in Definitions 3.3.4 and 3.3.5. The number of triangles formed by p is $m = |\mathcal{P}(p)| - 1$. If q is a valid refinement of p, the number of triangles will increase as $\mathcal{P}(p) \subset \mathcal{P}(q)$. Additionally, by the nature of the construction of the triangles, the triangles themselves will decrease in size and approach the curve itself. This can be seen by the simple example $f(x) = \sin x$ with $x \in [0, 2\pi]$. The base partition (which is unique in this case) is $p^0 = (0, \pi, 2\pi)$. Define $p = (0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}, 2\pi)$, which is simply the result of bisecting each sub-interval of p^0. The partition p is a valid refinement of p^0 and so once again triangles containing the curve over each sub-interval can be constructed using the points v_i, $v_{i,i+1}$, and v_{i+1} as previously described. This is shown in Figure 3.2. It can be clearly seen the triangles approach the curve itself as the number of partition points increases with valid refinements, as expected.

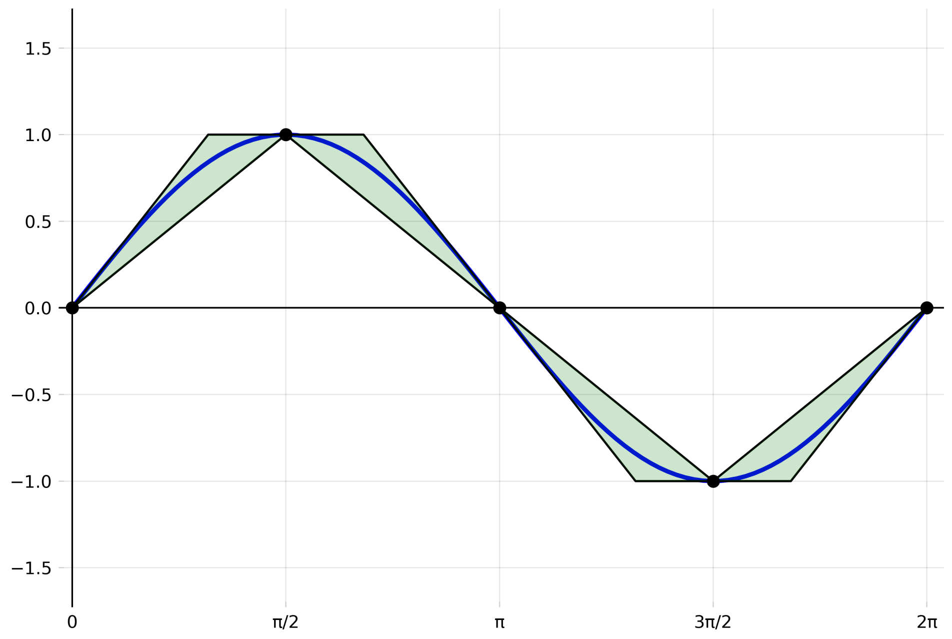
The main idea behind the MILP relaxation is then as follows: For each sub-interval $[x_i, x_{i+1}]$ of p, any point in the triangle defined by the points v_i, $v_{i,i+1}$, and v_{i+1} provides a relaxation of the constraint $y = f(x)$ when $x \in [x_i, x_{i+1}]$. Therefore, the disjunctive union of triangles formed by p using (3.2)-(3.5) provides a relaxation of $y = f(x)$ with $x \in [x^L, x^U]$. As valid refinements are iteratively constructed, this relaxation is tightened and will ultimately converge to the original function in the limit.

3.4.1.1 Incremental Formulation

We are now prepared to construct the MILP relaxation for the constraint $y = f(x)$ with $x \in [x^L, x^U] \subset \mathbb{R}$ where f is a univariate, trigonometric function that is differentiable and bounded over $[x^L, x^U]$. Let $p = (x_0, x_1, \dots, x_m)$ be a valid refinement of a base partition p^0 of $[x^L, x^U]$ for f or be



(a) Using base partition.



(b) Using valid refinement.

Figure 3.2: Example of triangles formed by overestimates, underestimates, and secant lines of $f(x) = \sin x$ with $x \in [0, 2\pi]$ and admissible partition p. The points on the curve corresponding to the partition points are shown with black markers. The green triangles correspond to the points v_i, $v_{i,i+1}$, and v_{i+1} for each sub-interval $[x_i, x_{i+1}]$ of p. (a) Triangles formed when using base partition $p^0 = (0, \pi, 2\pi)$. (b) Triangles formed when using the admissible partition $p = (0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}, 2\pi)$, which is a valid refinement of the base partition p^0.

p^0 itself. As before, let $v_i = f(x_i)$ and let $v_{i,i+1}$ be the intersection of the two tangent lines defined by (3.2) and (3.3) for each sub-interval $[x_i, x_{i+1}]$. Let v_i^x and v_i^y be the x- and y-coordinate of v_i and similarly for the vertex $v_{i,i+1}$. Define binary variables u_i for $i = 1, \dots, m-1$ and non-negative continuous variables δ_1^i and δ_2^i for $i = 1, \dots, m$. The disjunctive union of triangles formed by v_i, $v_{i,i+1}$, and v_{i+1} for each sub-interval $[x_i, x_{i+1}]$ provides a relaxation of $y = f(x)$ with $x \in [x^L, x^U]$ and can be expressed using a standard incremental formulation [50, 48] as

$$x = v_0^x + \sum_{i=1}^{m} \left(\delta_1^i (v_{i-1,i}^x - v_{i-1}^x) + \delta_2^i (v_i^x - v_{i-1}^x) \right) \tag{3.6a}$$

$$y = v_0^y + \sum_{i=1}^{m} \left[\delta_1^i (v_{i-1,i}^y - v_{i-1}^y) + \delta_2^i (v_i^y - v_{i-1}^y) \right] \tag{3.6b}$$

$$\delta_1^1 + \delta_2^1 \le 1 \tag{3.6c}$$

$$\delta_1^i + \delta_2^i \le u_{i-1} \le \delta_2^{i-1}, \quad \forall i \in \{2, \dots, m\} \tag{3.6d}$$

$$0 \le \delta_1^i,\, \delta_2^i \le 1, \quad \forall i \in \{1, \dots, m\} \tag{3.6e}$$

$$u_i \in \{0, 1\}, \quad \forall i \in \{1, \dots, m-1\} \tag{3.6f}$$

Though there are many ways to formulate the disjunctive union of triangles with theoretical properties similar to that of the incremental formulation as shown in [50, 52], the incremental formulation given can be shown to have computational superiority to other means of formulating the disjunctive union [53]. Nevertheless, comparing the different formulations for relaxing trigonometric functions is an interesting problem in its own right and will be delegated to potential future work.

The relaxation (3.6) can be understood as follows. The terms v_0^x and v_0^y represent a starting point for x and y. For partition $p = (x_0, x_1, \dots, x_m)$, we refer to the triangle corresponding to the sub-interval $[x_i, x_{i+1}]$ as the $(i+1)$-th triangle and so the triangles are ordered. Let (x, y) denote the relaxation value of $(x, f(x))$. The binary variable u_i takes value 1 if the i-th triangle must be passed to reached the triangle containing the point (x, y) and takes value 0 otherwise. It can be shown [50] that (3.6) has the ordering property $u_1 \ge u_2 \ge \dots \ge u_{m-1}$. Therefore, if $u_{i^*} = 1$ we must have $u_i = 1$

for all $i < i^*$, which simply states triangles must be considered in sequential order before reaching the triangle containing (x, y). The variables δ_1^i and δ_2^i for the i-th triangle have two purposes which are most easily understood through example. Suppose (x, y) lies in the first triangle. In this case, we must have $u_i = 0$ for all $i \in \{1, \dots, m-1\}$, which implies $\delta_1^i = 0$ and $\delta_2^i = 0$ for all $i \in \{2, \dots, m\}$ by (3.6d) and (3.6e). Relaxation (3.6) then becomes

$$x = v_0^x + \delta_1^1(v_{0,1}^x - v_0^x) + \delta_2^1(v_1^x - v_0^x) \tag{3.7a}$$

$$y = v_0^y + \delta_1^1(v_{0,1}^y - v_0^y) + \delta_2^1(v_1^y - v_0^y) \tag{3.7b}$$

$$\delta_1^1 + \delta_2^1 \leq 1 \tag{3.7c}$$

$$0 \leq \delta_1^1, \delta_2^1 \leq 1 \tag{3.7d}$$

From (3.7), we see δ_1^1 and δ_2^1 are used to capture any (x, y) in the first triangle defined by v_0, $v_{0,1}$, and v_1. In the case where (x, y) lies in the i-th triangle, variables δ_1^i and δ_2^i hold similar meaning. Next, suppose (x, y) lies in the second triangle. In this case, we must have $u_1 = 1$ and $u_i = 0$ for all $i \in \{2, \dots, m-1\}$. From (3.6d) we then must have $\delta_1^1 = 0$ and $\delta_2^1 = 1$. Similarly, we must also have $\delta_1^i = 0$ and $\delta_2^i = 0$ for all $i \in \{3, \dots, m\}$. Relaxation (3.6) then becomes

$$x = v_0^x + (v_1^x - v_0^x) + \delta_1^2(v_{1,2}^x - v_1^x) + \delta_2^2(v_2^x - v_1^x) \tag{3.8a}$$

$$y = v_0^y + (v_1^y - v_0^y) + \delta_1^2(v_{1,2}^y - v_1^y) + \delta_2^2(v_2^y - v_1^y) \tag{3.8b}$$

$$\delta_1^2 + \delta_2^2 \leq 1 \tag{3.8c}$$

$$0 \leq \delta_1^2, \delta_2^2 \leq 1 \tag{3.8d}$$

We now note that $\delta_2^1 = 1$ is the coefficient of $(v_1^x - v_0^x)$ and $(v_1^y - v_0^y)$, which corresponds to the secant line connecting v_0 and v_1. Therefore, δ_2^1 simply shifts the starting point from v_0 to v_1 or, equivalently, shifts the search for (x, y) from the first triangle to the second triangle. After this shift, we again see δ_1^2 and δ_2^2 are used to capture any (x, y) in the second triangle, as expected. The variable δ_1^1 is zero as we do not need it to shift from v_0 to v_1 by construction. In the case where (x, y) lies in the i^*-th triangle, variables δ_1^i and δ_2^i for all $i < i^*$ hold similar meaning.

3.4.1.2 Convergence Guarantee

The reader is referred to [48] for a proof the previously described MILP relaxation converges to the original function as all sub-intervals are refined in the limit. This convergences relies on the refinement strategy used on the partition. This is addressed in Section 3.5.2.

3.4.2 Bilinear Terms

This section presents an incremental formulation to obtain a MILP relaxation of the constraint $z = xy$ where $x \in [x^L, x^U] \subset \mathbb{R}$ and $y \in [y^L, y^U] \subset \mathbb{R}$. Such a relaxation can then be used for the bilinear terms (3.1d) in $\mathcal{F}$ found after factoring. The relaxation presented has a similar construction as the relaxation for trigonometric terms. Using MILP relaxations for bilinear terms is not new and has been dealt with explicitly in the literature [54, 46]. Typically the MILP relaxations use either a big-M reformulation or a convex hull representation of a disjunctive union [46]. The proposed approach instead uses an incremental formulation similar to the relaxation used for trigonometric terms. Furthermore, it is assumed that only one of the variables involved in the bilinear term, i.e., either x or y is partitioned. To the best of the author's knowledge, the following incremental formulation is not explicitly presented in the literature and so, for the sake of completeness it is presented here.

Consider the standard convex envelope [38] commonly used for relaxing bilinear terms.

$$z = w \tag{3.9a}$$

$$w \geq x^L y + x y^L - x^L y^L \tag{3.9b}$$

$$w \geq x^U y + x y^U - x^U y^U \tag{3.9c}$$

$$w \leq x^U y + x y^L - x^U y^L \tag{3.9d}$$

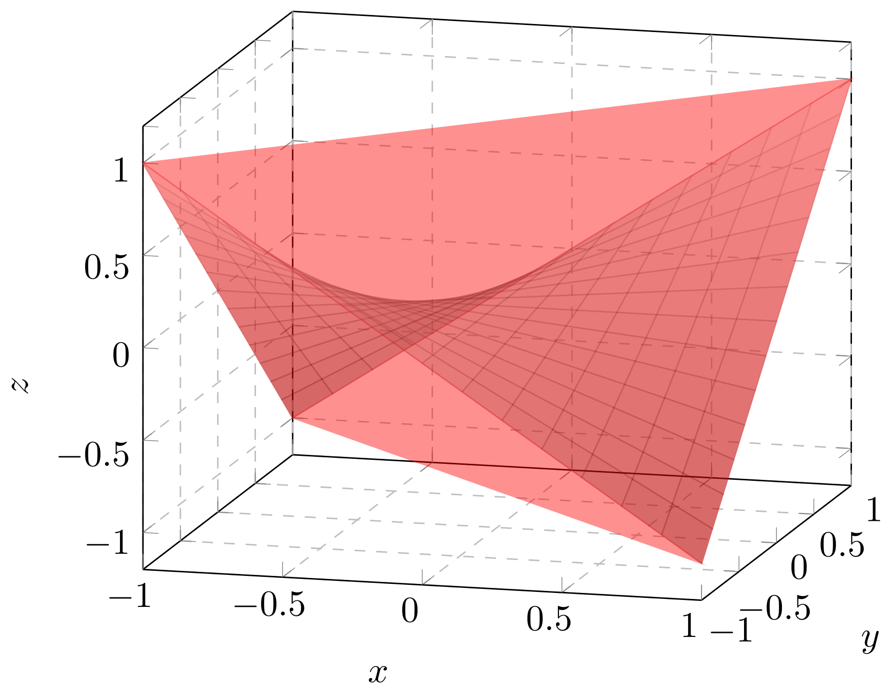


Figure 3.3: Convex hull of $z = xy$ with domain $[-1, 1] \times [-1, 1]$.

$$w \leq xy^U + x^L y - x^L y^U \tag{3.9e}$$

The convex envelope defined by (3.9) is precisely the convex hull of $z = xy$ over the domain $[x^L, x^U] \times [y^L, y^U]$. This convex hull is the tightest tetrahedron containing the graph of the bilinear term (see Figure 3.3). Consider a partition $p_x = (x_0, x_1, \ldots, x_m)$ of $[x^L, x^U]$. Let $R_{i+1} = [x_i, x_{i+1}] \times [y^L, y^U]$ be the $(i + 1)$-th rectangle defined by sub-interval $[x_i, x_{i+1}]$ of p_x. We can describe the graph over R_{i+1} by

$$z = xy \tag{3.10a}$$

$$x_i \leq x \leq x_{i+1} \tag{3.10b}$$

$$y^L \leq y \leq y^U \tag{3.10c}$$

Here (3.10) is once again a bilinear term with box constraints on x and y, so we can again construct

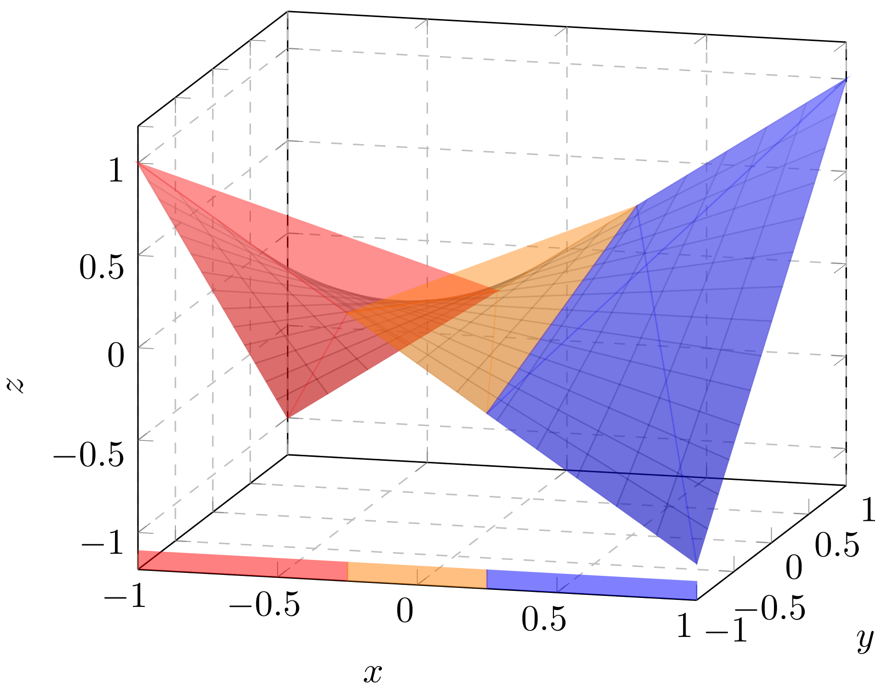


Figure 3.4: Disjunctive union of three tetrahedrons containing $z = xy$ over the domain $[-1, 1] \times [-1, 1]$. The variable x has been partitioned using $p_x = (-1, -0.25, 0.25, 1)$, shown by the colored bars along the x-axis with tetrahedrons being colored accordingly.

a convex envelope of (3.10) to get a tetrahedron containing the graph $z = xy$ over sub-domain $\boldsymbol{R}_{i+1}$. Figure 3.4 shows an example of this procedure for three sub-domains.

The main idea behind the MILP relaxation is then as follows: For each sub-domain $\boldsymbol{R}_{i+1}$ defined by partition p_x, construct the tightest tetrahedron (i.e., convex envelope) containing $z = xy$ over $\boldsymbol{R}_{i+1}$. Any point in the corresponding tetrahedron provides a relaxation of $z = xy$ with $(x, y) \in \boldsymbol{R}_{i+1}$. Therefore, the disjunctive union of tetrahedrons formed by p_x provides a relaxation of $z = xy$ over the original domain $[x^L, x^U] \times [y^L, y^U]$.

3.4.2.1 Incremental Formulation

We are now prepared to construct the MILP relaxation for the bilinear term $z = xy$ with $x \in [x^L, x^U] \subset \mathbb{R}$ and $y \in [y^L, y^U] \subset \mathbb{R}$. Let $p_x = (x_0, x_1, \ldots, x_m)$ be a partition of $[x^L, x^U]$. Define binary variables u_i for $i = 1, \dots, m-1$ and non-negative continuous variables δ_1^i, δ_2^i, and δ_3^i for $i = 1, \dots, m$. The MILP relaxation is the disjunctive union of tetrahedrons constructed from p_x and can be expressed [50] as

$$x = x^L + \sum_{i=1}^{m} \left[\delta_2^i(x_i - x_{i-1}) + \delta_3^i(x_i - x_{i-1})\right] \tag{3.11a}$$

$$y = y^L + \sum_{i=1}^{m} \left[\delta_1^i(y^U - y^L) + \delta_2^i(y^U - y^L)\right] \tag{3.11b}$$

$$z = x^L y^L + \sum_{i=1}^{m} \begin{pmatrix} \delta_1^i \\ \delta_2^i \\ \delta_3^i \end{pmatrix}^T \begin{pmatrix} x_{i-1}y^U - x_{i-1}y^L \\ x_i y^U - x_{i-1}y^L \\ x_i y^L - x_{i-1}y^L \end{pmatrix} \tag{3.11c}$$

$$\delta_1^i + \delta_2^i + \delta_3^i \le u_{i-1} \le \delta_3^{i-1}, \quad \forall i \in \{2, \dots, m\} \tag{3.11d}$$

$$\delta_1^1 + \delta_2^1 + \delta_2^1 \le 1 \tag{3.11e}$$

$$0 \le \delta_1^i,\, \delta_2^i,\, \delta_3^i \le 1, \quad \forall i \in \{1, \dots, m\} \tag{3.11f}$$

$$u_i \in \{0, 1\}, \quad \forall i \in \{1, \dots, m-1\} \tag{3.11g}$$

Relaxation (3.11) can be understood as follows. For the given partition p_x, we will refer to the tetrahedron corresponding to sub-domain R_{i+1} as the $(i+1)$-th tetrahedron and so the tetrahedrons are ordered. Note the $(i+1)$-th tetrahedron over R_{i+1} has the four extreme points

$$v_{i+1}^0 = (x_i, y^L, x_i y^L) \tag{3.12a}$$

$$v_{i+1}^1 = (x_i, y^U, x_i y^U) \tag{3.12b}$$

$$v_{i+1}^2 = (x_{i+1}, y^U, x_{i+1} y^U) \tag{3.12c}$$

$$v_{i+1}^3 = (x_{i+1}, y^L, x_{i+1} y^L) \tag{3.12d}$$

The terms x^L, y^L, and $x^L y^L$ represent starting points for x, y, and z, respectively. In particular,

$(x^L, y^L, x^L y^L)$ is the extreme point v_1^0. Let (x, y, z) denote the relaxation value of (x, y, xy). The binary variable u_i takes value 1 if the i-th tetrahedron must be passed to reach the tetrahedron containing the point (x, y, z) and takes value 0 otherwise. It can be shown [50] that (3.11) has the ordering property $u_1 \geq u_2 \geq \ldots \geq u_{m-1}$. Therefore, if $u_{i^*} = 1$ then we must have $u_i = 1$ for all $i < i^*$, which simply states we must pass the tetrahedrons in order before reaching the tetrahedron containing (x, y, z). The variables δ_1^i, δ_2^i, and δ_3^i for the i-th tetrahedron have similar meaning as the δ variables in the trigonometric case.

3.4.2.2 Convergence Guarantee

It can be shown [55] the largest gap between the convex envelope (3.9) over a sub-domain $R_{i+1} = [x_i, x_{i+1}] \times [y^L, y^U]$ is at most

$$\max_{(x,y)\in R_{i+1}} |w_{i+1} - xy| = \frac{(x_{i+1} - x_i)(y^U - y^L)}{4} \tag{3.13}$$

where w_{i+1} denotes the relaxation value. This bound holds for each sub-domain and so as all sub-domains are further refined the MILP relaxation (3.11) converges to the original bilinear function in the limit. This convergences relies on all sub-domains being further refined. This will be addressed in Section 3.5.2.

3.5 Partitions

This section focuses on the schemes used for partitioning selected variable domains which in turn are used to construct the MILP relaxations previously described. It is first shown how to reduce the total number of variables needed across MILP relaxations of different nonlinear functions by sharing partitions between functions when multiple functions use the same variable. This is especially important when using trigonometric functions, as many applications will have several functions sharing the same variable (for example, $\sin x$ and $\cos x$ may appear in pairs when looking at signals or geometric constraints). Then several refinement schemes are introduced, which are responsible for refining a given partition after finding an optimal solution to the MILP relaxation of $\mathcal{F}$ to further tighten the relaxation. The refinement scheme chosen will directly impact the convergence rate of the overall algorithm and so three options are presented - bisection, direct, and non-uniform. It is entirely possible to use alternative refinement schemes not discussed in this paper to great success and the author does not claim to have exhaustively examined all possible refinement schemes. When refining the partitions for the MILP relaxation of $\mathcal{F}$, we may either choose to refine all available partitions or elect to refine a subset of partitions subject to some criteria (under certain conditions so convergence to a globally optimal solution is ensured). These two possibilities are briefly discussed and a simple criteria for refining a subset of all partitions is given as a motivating example.

3.5.1 Sharing Partitions

For ease of discussion, we will consider two univariate trigonometric functions $f_1(x)$ and $f_2(x)$ which share the same variable x and have been relaxed using the MILP relaxation presented in Section 3.4. The same procedure presented applies to bilinear terms that have been relaxed as well and may be extended to more than two functions. The variable x has a box constraint $[x^L, x^U]$.

Let p_1^0 and p_2^0 be base partitions of $[x^L, x^U]$ for f_1 and f_2, respectively. Rather than constructing two separate partitions, we can use information from both base partitions to construct a single partition that can be used for both functions simultaneously in the MILP relaxation. Define a partition as the union of points in p_1^0 and p_2^0. If the slope condition in Definition 3.3.4 is not initially satisfied (applied to both functions), add additional partition points so that the slope condition is satisfied and the resulting partition, p^0, is of minimum size. The resulting partition is in a sense a base partition of the collection of functions. We note that for any sub-interval $[x_i, x_{i+1}]$ defined by p^0, the functions f_1 and f_2 are either convex or concave over the sub-interval (the functions need not have the same convexity). This allows the use the incremental formulation presented in Section 3.4 and so we can refine p^0 to tighten all parts of the formulation containing f_1 and f_2 rather than refining two separate partitions. In doing so, we reduce the number of binary variables needed for f_1 and f_2 in the incremental formulation by half (or a factor of M when M functions are considered instead of two). This will significantly reduce the computation time needed.

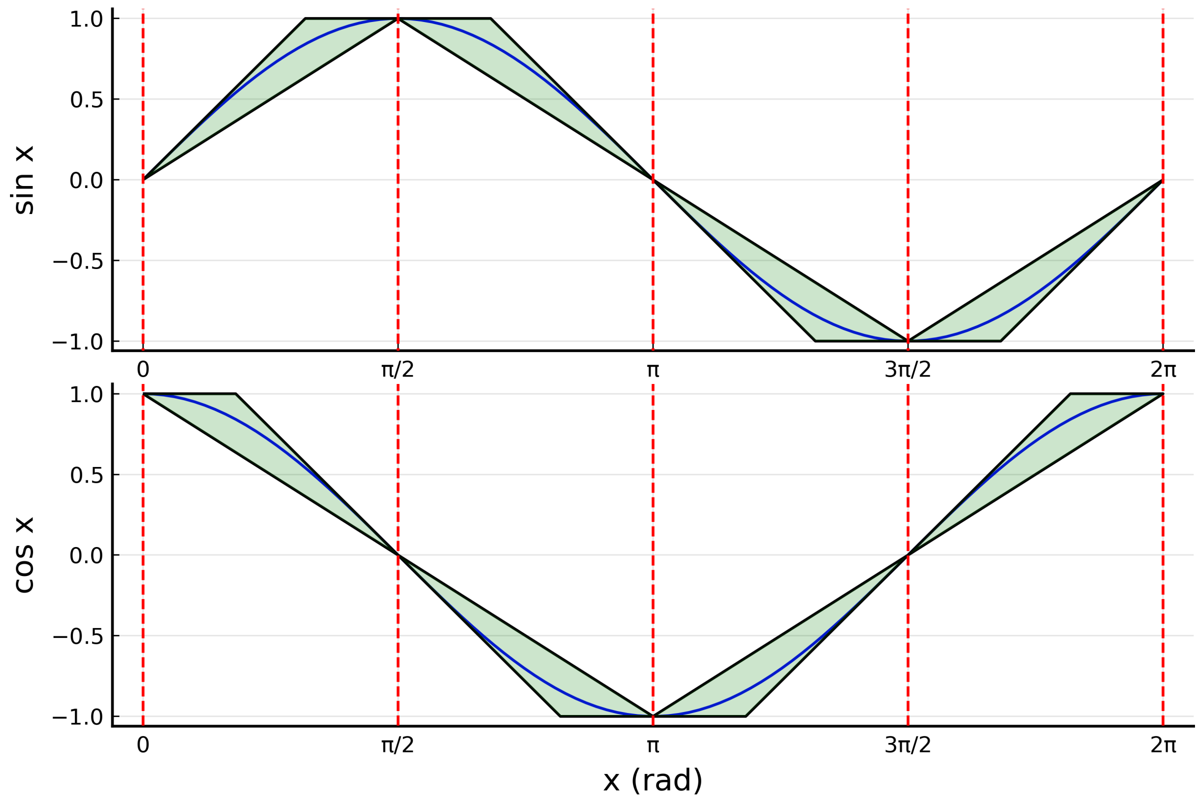


Figure 3.5: Example of using a shared partition, $p^0 = (0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}, 2\pi)$, for $f_1(x) = \sin x$ and $f_2(x) = \cos x$ over the closed interval $[0, 2\pi]$. It can be seen the i-th triangle for f_1 and the i-th triangle for f_2 are defined over the same sub-interval, so they can be linked by sharing the same binary variables in the MILP relaxations. The binary variables then indicate which sub-interval the solution is in, rather than which triangle.

3.5.1.1 Sharing Partitions Example

As a simple example, consider the two functions $f_1(x) = \sin x$ and $f_2(x) = \cos x$ where $x \in [0, 2\pi]$. The (unique) base partitions for f_1 and f_2 are $p_1^0 = (0, \pi, 2\pi)$ and $p_2^0 = (0, \frac{\pi}{2}, \frac{3\pi}{2}, 2\pi)$, respectively. We can then define a new partition $p^0 = (0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}, 2\pi)$. We see the slope condition in Definition 3.3.4 is satisfied for both f_1 and f_2 with p^0 and so p^0 is effectively a base partition for the set of functions. We can then use relaxation (3.6) for both f_1 and f_2 using the same partition. The i-th triangle for f_1 and the i-th triangle for f_2 are linked since they are defined over the same sub-interval. This is readily seen in Figure 3.5. With this in mind, the binary variables in relaxation (3.6) for both functions may be shared.

3.5.2 Refinement Schemes - Method of Partition Refinement

The following definition is first introduced.

Definition 3.5.1. Given an admissible partition p of $[x^L, x^U]$ for a function f, the procedure used to generate a valid refinement of p is referred to as a *refinement scheme*.

A refinement scheme is for a *single* partition and it is entirely possible to use various refinement schemes depending on the partition considered. As a simple example, consider the base partition $p^0 = (0, \pi, 2\pi)$ of $[0, 2\pi]$ for $f(x) = \sin x$ (see Figure 3.2). We can construct the valid refinement $q = (0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}, 2\pi)$, where the refinement scheme used is: For each sub-interval defined by p^0 we add a point to bisect that sub-interval. We note that in this particular example the refinement scheme used is indifferent of any known solution to the MILP relaxation. This may be considered inefficient, as some sub-intervals are further refined despite them potentially not being useful to the solution of the original problem. More targeted refinement schemes using information of the optimal solution to the MILP relaxation are soon considered.

For the remainder of this section, the following notation will be used. Suppose $\mathcal{F}$ has been relaxed using the previously described MILP relaxations. For a given term (trigonometric or bilinear) that has been relaxed, let x be the variable that has been partitioned with $p = (x_0, \dots, x_m)$ and let x^* denote its value in the optimal solution of the MILP. Only a single partition p needs to be associated with x because multiple relaxed terms using x may share the same partition as previously discussed. It will be assumed x^* is strictly in the interior of a sub-interval $[x_i, x_{i+1}]$ defined by p. In the case where $x^* = x_i$ for some $x_i \in \mathcal{P}(p)$, we do not refine partition p *for that particular iteration of the overall algorithm.* In a later iteration we may have x^* strictly in the interior of a sub-interval defined by p and so we may refine p for that iteration. Three refinement schemes are now presented — bisection, direct, and non-uniform.

3.5.2.1 Bisection Refinement Scheme

The first refinement scheme is the bisection refinement scheme. Given $x^* \in (x_i, x_{i+1})$, the bisection refinement scheme is to simply add a new partition point $x' = \frac{1}{2}(x_i + x_{i+1})$ to p, giving the valid refinement $q = (x_0, \ldots, x_i, x', x_{i+1}, \ldots, x_m)$.

It is important to note that while this refinement scheme is simple, it may not *always* improve the lower bound to $\mathcal{F}$. This can be seen by considering $f(x) = \sin x$ with domain $[0, \pi]$. We may first start with the base partition $p = (0, \pi)$ and use relaxation (3.6) for f using p, where we introduce a variable y for the relaxation value of $f(x)$. This will result in a single triangle containing the curve (green triangle in Figure 3.6). After relaxing f (and other constraints), we may solve the resulting MILP, yielding values (x^*, y^*). It is possible for (x^*, y^*) to lie in the *interior* of the triangle defined by p. Since x^* is in the interior of $[0, \pi]$, we may refine p using bisection to get the valid refinement $q = (0, \frac{\pi}{2}, \pi)$. Using relaxation (3.6) with q results in two triangles containing f (blue triangles in Figure 3.6). Because (x^*, y^*) was in the interior of the original triangle, it is possible this point is also in one of the two newly created triangles. This case is shown in Figure 3.6. In the event this happens, solving the new MILP relaxation *may not result in a tighter lower bound*, though this will largely depend on any other constraints present in the problem. We also note a similar situation can occur when (x^*, y^*) is on an *edge* of a triangle, which can be seen by noting the overlapping edges in Figure 3.6.

It is important to note that while the lower bound may not improve for a particular iteration using bisection, the MILP's solution will still converge to the optimal solution of $\mathcal{F}$ in the limit. Once again consider the example shown in Figure 3.6 and suppose (x^*, y^*) remains optimal when relaxing using q. We then apply the bisection refinement scheme to q, resulting in $r = (0, \frac{\pi}{4}, \frac{\pi}{2}, \pi)$ and so two triangles are created over $[0, \frac{\pi}{2}]$. Visually, it should be clear from Figure 3.6 that the two triangles will likely not contain (x^*, y^*). When this happens, the lower bound will continue to improve. This same idea holds in the case of bilinear terms as well. We are guaranteed this will always eventually happen because the MILP relaxations used approach the original functions in the limit. As such, any relaxation solution that is not on the original function will eventually become infeasible as partitions are further refined.

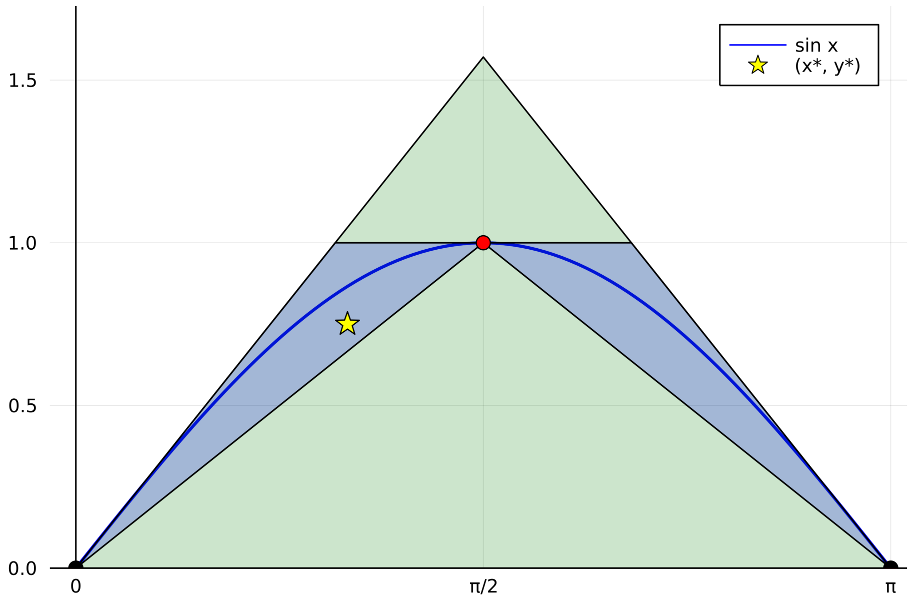


Figure 3.6: MILP relaxation of $f(x) = \sin x$ over $[0, \pi]$ using $p = (0, \pi)$ (green) and the valid refinement $q = (0, \frac{\pi}{2}, \pi)$ (blue), which was constructed using a bisection refinement scheme. The optimal solution (yellow star) of the resulting MILP, (x^*, y^*), when using p may still lie in the triangles constructed using q. If this happens, (x^*, y^*) remains a possible solution to the MILP when using q and so the lower bound may not improve for that particular iteration of the algorithm.

3.5.2.2 Direct Refinement Scheme

We next consider the direct refinement scheme, which is another natural choice. Given $x^* \in (x_i, x_{i+1})$, the direct refinement scheme is to simply add x^* to p, giving the valid refinement $q = (x_0, \dots, x_i, x^*, x_{i+1}, \dots, x_m)$.

Unlike bisection, the direct refinement scheme is guaranteed to remove the previous MILP optimal solution from the relaxation's feasible space except in the extremely rare (and favorable) case the optimal solution lies on the original curve. Once again consider $f(x) = \sin x$ with domain $[0, \pi]$. As before, we start with the base partition $p = (0, \pi)$, relax f using relaxation (3.6) (introducing variable y for the relaxation value of $f(x)$), and solve the resulting MILP giving optimal solution (x^*, y^*). Suppose (x^*, y^*) lies in the interior of the triangle constructed using p. Since x^* is in the interior of $[0, \pi]$, we may refine p using the direct refinement scheme to get the valid refinement $q = (0, x^*, \pi)$. Suppose we now use relaxation (3.6) using q. Since y^* lies on the vertical line

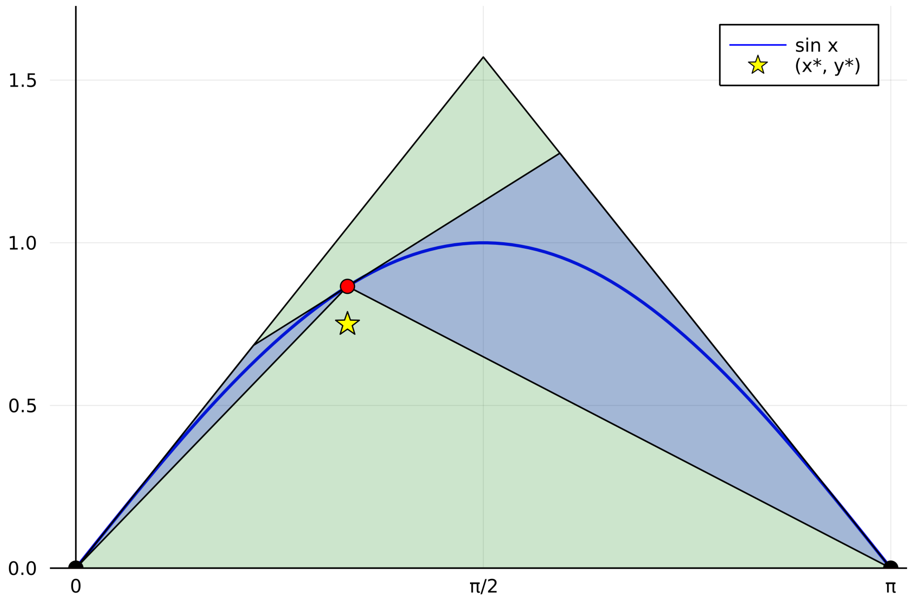


Figure 3.7: MILP relaxation of $f(x) = \sin x$ over $[0, \pi]$ using $p = (0, \pi)$ (green) and the valid refinement $q = (0, x^*, \pi)$ (blue), which was constructed using a direct refinement scheme. The previous optimal solution to the MILP using p (yellow star) is no longer feasible when constructing the relaxation of f using q.

$x = x^*$, we see the tangent lines and secant line defined in (3.6) will only contain (x^*, y^*) if either (i) $y^* = f(x^*)$ or (ii) a tangent line is vertical, which cannot happen as we require the function to be differentiable. This can be seen in Figure 3.7. A similar discussion holds for bilinear terms that have been relaxed using (3.11).

3.5.2.3 Non-Uniform Refinement Scheme

Another refinement scheme to consider is the non-uniform refinement scheme. In bisection, the location of the added partition point only considered which sub-interval contained x^* and was indifferent to the exact location of x^*. In contrast, the direct refinement scheme only considered the exact location of x^*. The non-uniform refinement scheme attempts to strike a balance between these two refinement schemes and is as follows. Let $\Delta_1, \Delta_2 > 1$ be user-defined constants and let $x^* \in (x_i, x_{i+1})$ as before. Define

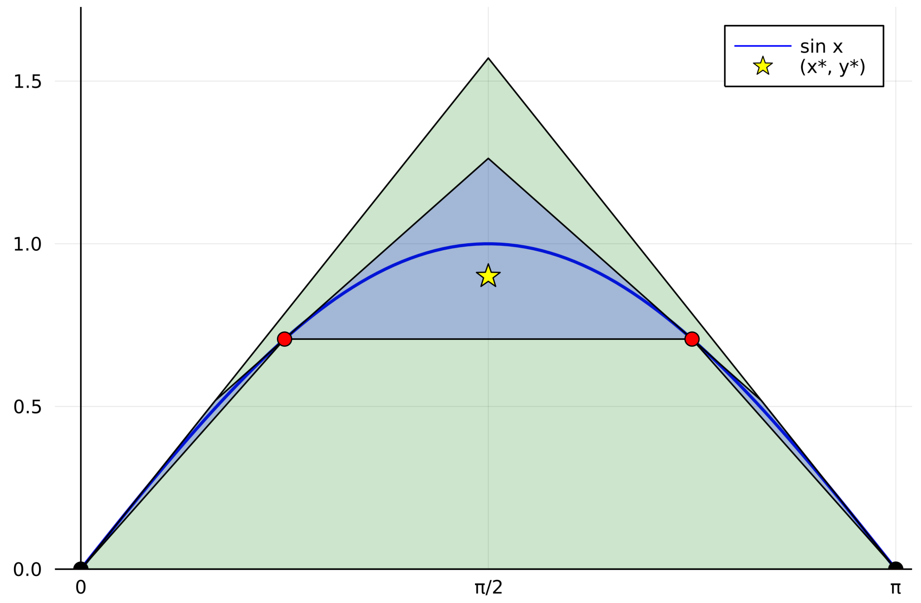
$$x_1' = x^* - \frac{x^* - x_i}{\Delta_1} \tag{3.14}$$

$$x_2' = x^* + \frac{x_{i+1} - x^*}{\Delta_2} \tag{3.15}$$

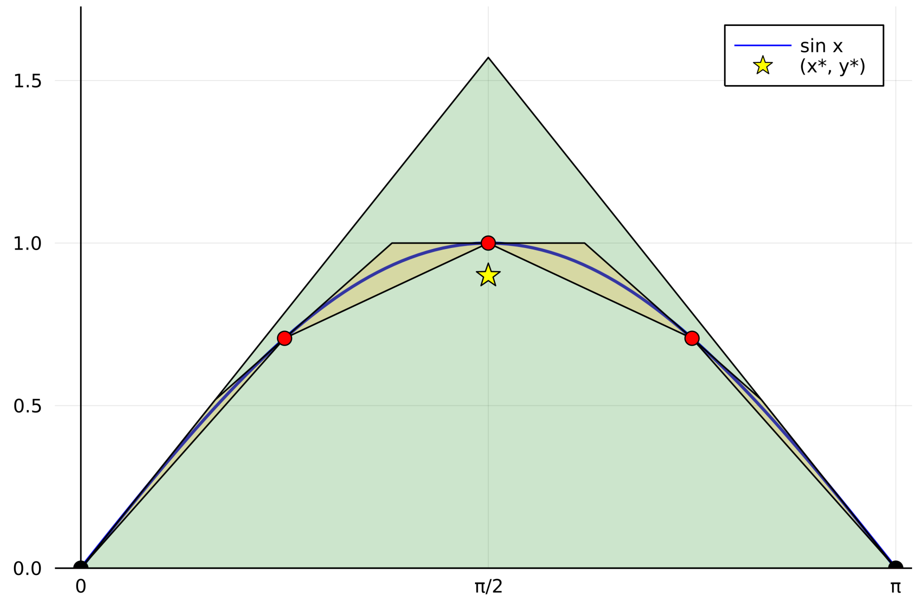
In the *two-point non-uniform refinement scheme*, points x_1' and x_2' are added to p to yield the valid refinement $q = (x_0, \dots, x_i, x_1', x_2', x_{i+1}, \dots, x_m)$. In a sense, the two-point refinement scheme attempts to remove large amounts of the relaxation space, like in bisection, while still being influenced by the exact location of x^*. Larger values of Δ_1 and Δ_2 will produce a smaller polyhedron (triangle for trigonometric or tetrahedron for bilinear) containing the curve or surface near x^*. This refinement scheme has been proposed previously in the literature with $\Delta_1 = \Delta_2$ in the context of adaptive partitioning schemes [46]. In the *three-point non-uniform refinement scheme*, the point x^* is also added to the partition to yield the valid refinement $q = (x_0, \dots, x_i, x_1', x^*, x_2', \dots, x_m)$. Adding point x^* will ensure the previous MILP's optimal solution will be removed from the feasible space, just like in the direct refinement scheme. In the two-point refinement scheme, two binary variables are added for each partition refined using this scheme. In the three-point refinement scheme, three binary variables are added.

To better visualize both the two-point and three-point non-uniform refinement schemes, once again consider $f(x) = \sin x$ with domain $[0, \pi]$. As before, start with the initial base partition $p = (0, \pi)$, apply relaxation (3.6) for f using p (introducing variable y for the relaxation value of $f(x)$), and solve the resulting MILP giving optimal solution (x^*, y^*). Suppose (x^*, y^*) lies in the interior of the triangle constructed using p. As in the bisection refinement scheme, the two-point refinement scheme may not remove (x^*, y^*) from the feasible space after refinement. However, as in the direct refinement scheme, the three-point refinement scheme will always remove (x^*, y^*) from the feasible space except for in the rare case $y^* = f(x^*)$. These observations directly follow from the discussion of the two previous refinement schemes. This can be seen in Figure 3.8.

Note in Figure 3.8, the three-point refinement scheme results in a significantly tighter relaxation, but has four triangles instead of three in the two-point refinement. This appears to make the three-point refinement scheme the better choice of the two. However, suppose the domain was larger than $[0, \pi]$. The base partition would necessarily consist of more points, leading to more initial



(a)



(b)

Figure 3.8: Example showing partition $p = (0, \pi)$ being refined using (a) the two-point non-uniform refinement scheme and (b) the three-point non-uniform refinement scheme. The initial MILP relaxation using p is shown in green. In (a) the optimal solution (yellow star) for the previous iteration remains feasible, while in (b) it is no longer feasible after refinement. In both cases we have set $\Delta_1 = \Delta_2 = 2$.

triangles. It is possible both refinement schemes may tighten the relaxation over $[0, \pi]$ enough for the optimal solution to move to a different sub-interval and never return to $[0, \pi]$. If this happens, the three-point refinement scheme will have generated an additional binary variable that no longer contributes to the overall convergence of the algorithm. With this in mind, the choice between using two points or three points is not clear in general and would require computational experiments to make an informed decision.

3.5.2.4 Consistent Refinement Scheme

Consider the function f (trigonometric or bilinear) that has been relaxed using one of the previously described MILP relaxations using partition p for variable x. As p is refined using a chosen refinement scheme, the MILP relaxation of f is tightened. When all sub-intervals defined by p are further refined, the MILP relaxation will approach f in the limit. However, the refinement schemes presented only further refine a single sub-interval to limit the number of additional variables generated at each iteration of the algorithm. An additional mechanism so the MILP relaxation approaches f in the limit is required. Motivated by [56, 39] for global optimization using branch-and-bound, the following definition is introduced:

Definition 3.5.2. A refinement scheme is said to be *consistent* if at every step any sub-interval defined by the current partition is capable of further refinement and the length of every sub-interval approaches zero (or a small $\varepsilon > 0$) in the limit.

Note the use of the word "capable" in Definition 3.5.2. It is not *required* that every sub-interval be further refined. This concept is similar to the concept of a consistent branch-and-bound algorithm, where the feasible space may be subdivided (branching) and all sub-domains may be explored unless it is known a sub-domain cannot contain the optimal solution (pruning).

The previous three refinement schemes are not consistent. In order to make these refinement schemes consistent, the following simple rule is added to the algorithm.

Consistency Rule 3.5.1. Let p be a given partition and let $[x_i, x_{i+1}]$ be a sub-interval defined by p to be further refined by adding one or more partition points according to a refinement scheme. If

$|x_i - x_{i+1}| < \varepsilon$ for small $\varepsilon > 0$, bisect the largest sub-interval defined by p instead.

This rule ensures the algorithm does not get stuck refining the same initial sub-interval when the optimal solution lies in a different sub-interval. Using the largest sub-interval ensures all sub-intervals initially defined by a base partition are *capable* of further refinement. The choice of bisection is for simplicity and alternative refinement schemes could be used. Note that if the optimal solution to the MILP relaxation results in x being equal to some $x_i \in \mathcal{P}(p)$, we do not refine p for that iteration and so we would not need to invoke this rule.

Using any of the previously described refinement schemes along with Consistency Rule 3.5.1 guarantees the MILP relaxation of a function (trigonometric or bilinear) will approach the function in the limit. This only concerns a single variable in $\mathcal{F}$ and so additional criteria is needed to ensure *all* relaxations of functions approach their original functions in the limit. This is the subject of Section 3.5.3.

3.5.3 Refinement Strategies - Selecting Partitions for Refinement

The following definition is first introduced.

Definition 3.5.3. Let $\mathscr{P}$ be the set of partitions used to construct the MILP relaxation of $\mathcal{F}$. The procedure used to choose a subset $\mathscr{P}_1 \subseteq \mathscr{P}$ of partitions to be refined is referred to as a *refinement strategy*.

Let $\mathcal{R}(\mathcal{F}, \mathscr{P})$ denote the relaxation of $\mathcal{F}$ using the previously described MILP relaxations using a set of partitions $\mathscr{P}$ (after sharing partitions). Consider a partition $p \in \mathscr{P}$ corresponding to function f (a similar discussion holds for when p is shared among a set of functions) with partitioned variable x. In Section 3.5.2, it was noted the MILP relaxation of $y = f(x)$ will approach the original curve if a consistent refinement scheme is used. This only considers a single partition and so the purpose of this section is extend this to the set $\mathscr{P}$. Doing so will ensure the optimal solution of $\mathcal{R}(\mathcal{F}, \mathscr{P})$ approaches the optimal solution of $\mathcal{F}$ in the limit as further refinements are performed. Similar to the concept of a consistent refinement scheme, the concept of a consistent refinement strategy is introduced to accomplish this.

The remainder of this section is organized as follows. The concept of a consistent refinement strategy is introduced to guarantee the optimal solution of $\mathcal{R}(\mathcal{F}, \mathscr{P})$ approaches the optimal solution of $\mathcal{F}$ in the limit. Two consistent refinement strategies are then provided. The first refinement strategy is the most natural refinement strategy where all partitions are refined at each iteration (i.e., $\mathscr{P}_1 = \mathscr{P}$). The second refinement strategy considers a subset of partitions based on the quality of the relaxations associated with those partitions.

3.5.3.1 Consistent Refinement Strategy

The concept of a consistent refinement strategy is now introduced.

Definition 3.5.4. A refinement strategy is said to be *consistent* if at every iteration of the algorithm every partition used for the MILP is capable of being selected for refinement and in the limit every partition will be refined such that the corresponding relaxations converge to the original functions.

Note the use of the word "capable" in Definition 3.5.4. Similar to consistent refinement scheme, it is not *required* that every partition to be refined. In fact, it is possible for one or many initial partitions to never be refined and still have the optimal value of the relaxation of $\mathcal{F}$ approach the optimal solution to $\mathcal{F}$. This can happen when the optimal solution has a point that corresponds to one or many initial partition points. This is similar to the case in branch-and-bound when a feasible solution is found at a node in the branch-and-bound tree and so no branching is needed for that node. The condition a refinement strategy be consistent is equivalent to requiring every relaxation (trigonometric or bilinear) be capable of being tightened at every iteration of the overall algorithm until the gap is sufficiently small.

3.5.3.2 Complete Refinement Strategy

The most obvious consistent refinement strategy is to refine all partitions $\mathscr{P}$ at each iteration of the overall algorithm. In doing so, all relaxations (trigonometric and bilinear) are tightened at each iteration until the gap is sufficiently small. This refinement strategy will be referred to as the complete refinement strategy.

The complete refinement strategy has advantages and disadvantages. It is easy to implement and avoids additional computations and sorting that may be used in an alternative refinement strategy that selects a subset of partitions. However, a major disadvantage of this strategy is the number of variables (with emphasis on binary variables) will grow at a faster rate than any other refinement strategy that only uses a subset of partitions. As a result, the complete refinement strategy's effectiveness will depend on the number of iterations needed in the overall algorithm (i.e., the number of times the refinement strategy is invoked) before the gap is sufficiently small. If the number of iterations to reach optimality is small, then the growth in the number of variables will be manageable.

3.5.3.3 k-Worst Refinement Strategy

An alternative consistent refinement strategy that selects a subset of partitions $\mathscr{P}_1 \subseteq \mathscr{P}$ for further refinement at each iteration of the overall algorithm is now presented. To do so, the following definition is introduced.

Definition 3.5.5. Let $p_x \in \mathscr{P}$ be a partition for $x \in [x^L, x^U]$ corresponding to functions $\{f_1(x), \dots, f_M(x)\}$ with relaxation values $y_1, \dots, y_M$ in $\mathcal{R}(\mathcal{F}, \mathscr{P})$. Let (x^*, y_i^*) denote the value of x and y_i in the optimal solution to $\mathcal{R}(\mathcal{F}, \mathscr{P})$. The *measure of* p_x, denoted by $\mu(p_x)$, is then

$$\mu(p_x) = \max_{1 \le i \le M} |f_i(x^*) - y_i^*|. \tag{3.16}$$

Note if $p_x \in \mathscr{P}$ only corresponds to a single function f, we simply have $\mu(p_x) = |f(x^*) - y^*|$ in Definition 3.5.5. The measure of a partition p_x provides a means of quantifying the quality of the MILP relaxations of the corresponding functions (with lower values being desirable). Suppose $\mathcal{R}(\mathcal{F}, \mathscr{P})$ has been solved to optimality and the measure of all partitions in $\mathscr{P}$ have been computed. The proposed alternative consistent refinement strategy is then to choose the k partitions with the k highest measures (with ties broken arbitrarily) for further refinement. This refinement strategy will be referred to as the k-worst refinement strategy. The integer k is set in advance by the user and requires knowledge of the number of functions to be relaxed in $\mathcal{F}$. The user may also choose to modify k as the algorithm is running, either by increasing or decreasing k to a valid non-zero integer. For the purposes of this paper, k will be kept constant when using this refinement strategy. Note $k = |\mathscr{P}|$ results in the complete refinement strategy.

The k-worst refinement strategy has the advantage of reducing the number of variables (with emphasis on binary variables) added each time partitions are refined to tighten the relaxation. However, a major disadvantage of this refinement strategy is the number of iterations needed to get a sufficiently small gap may increase as a result. This may lead to a longer overall computational time and will largely depend on the maximum gap allowed before terminating the algorithm.

3.6 Principal Domains for Periodic Functions

In this section we discuss a simple reformulation for bounded, periodic functions with period T where the corresponding variable has a domain that does not align with a known, more convenient domain of width T. We will refer to this more convenient domain as a *principal domain*. A few simple examples of such principal domains for $T = 2\pi$ are $[0, 2\pi]$, $[-\pi, \pi]$, and more generally $[\theta, \theta + 2\pi]$ for any $\theta \in \mathbb{R}$. For sine and cosine, these example principal domains capture all possible values and so if a variable's domain contains one of these principal domains (i.e., the domain's width is at least T), then the points outside a principal domain of interest are in some sense redundant. A similar observation holds for when the variable's original domain can be shifted to contain a principal domain of interest. Noting this, we aim to reduce the number of binary variables introduced in the MILP relaxation of trigonometric functions, as these binary variables will likely have a significant impact on the overall computational time needed to solve the original MINLP. We also show the presented reformulation gives the ability to effectively tighten multiple regions of the original domain simultaneously.

3.6.1 Reformulation Using Principal Domains

Let $f(x)$ be a periodic, bounded, univariate function with period T and with $x \in [x^L, x^U] \subset \mathbb{R}$ where $x^U - x^L \geq T$. Let $[\hat{x}^L, \hat{x}^U] \subset \mathbb{R}$ be a principal domain of interest for f where $\hat{x}^U - \hat{x}^L = T$. For brevity, we will denote $[x^L, x^U]$ and $[\hat{x}^L, \hat{x}^U]$ by I and $\hat{I}$, respectively. We will focus our attention to the case $\hat{I} \subseteq I$. The procedure for $\hat{I} \not\subseteq I$ can be easily deduced based on the following discussion and will only differ from $\hat{I} \subseteq I$ by considering where the endpoints of I are relative to the endpoints of $\hat{I}$. If $I = \hat{I}$, clearly no additional work needs to be done. In the case $\hat{I} \subset I$, define

$$\alpha^L = \left\lfloor \frac{x^L - \hat{x}^L}{T} \right\rfloor \tag{3.17}$$

$$\alpha^U = \left\lceil \frac{x^U - \hat{x}^U}{T} \right\rceil \tag{3.18}$$

where $\lfloor \cdot \rfloor$ and $\lceil \cdot \rceil$ denote the floor and ceil function, respectively. We note α^L and α^U are integers by construction and may take on negative values. We may then reformulate $y = f(x)$ with $x \in [x^L, x^U]$ by

$$y = f(\hat{x}) \tag{3.19a}$$

$$\hat{x} = x - \alpha T \tag{3.19b}$$

$$\hat{x} \in [\hat{x}^L, \hat{x}^U] \tag{3.19c}$$

$$x \in [x^L, x^U] \tag{3.19d}$$

$$\alpha \in \{\alpha^L, \dots, \alpha^U\} \tag{3.19e}$$

We have introduced a new variable $\hat{x}$ and we observe in (3.19a) the function takes in $\hat{x}$ as its argument. This is valid because f is T-periodic and bounded and the principal domain $[\hat{x}^L, \hat{x}^U]$ has width T, so we may make use of the mapping (3.19b) to accomplish this. The trade-off for doing this is the introduction of the integer variable α which takes on an integer value between α^L and α^U. The variable α represents the number of integer steps of width T a point $x \in [x^L, x^U]$ is from an equivalent point $\hat{x} \in [\hat{x}^L, \hat{x}^U]$. Depending on the original domain I, this reformulation may not be more efficient than using the original formulation. However, as we will see in Section 3.8, this reformulation may lead to noticeable improvements in the computational time needed to solve the original MINLP.

3.6.2 Impact on MILP Relaxation

In order to see how reformulation (3.19) impacts the MILP relaxations of trigonometric terms as described in Section 3.4.1.1, consider the case where the MINLP has constraints $y_1 = \sin(\theta)$ and $y_2 = \cos(\theta)$ with $\theta \in [-4\pi, 4\pi]$. From Section 3.5, a single partition p may be used for relaxing y_1 and y_2 simultaneously. We will use the union of the base partitions of y_1 and y_2 as the shared partition, which will consist of 17 points, noting that we require a point for every multiple of $\frac{\pi}{2}$ in $[-4\pi, 4\pi]$. The corresponding initial MILP relaxation of y_1 and y_2 using this partition will then consist of 15 binary variables and 32 non-negative continuous variables. We will choose $[0, 2\pi]$ to be the principal domain of interest. For this principal domain, we may rewrite these constraints as

$$y_1 = \sin(\hat{\theta}) \tag{3.20a}$$

$$y_2 = \cos(\hat{\theta}) \tag{3.20b}$$

$$\hat{\theta} = \theta - 2\pi\alpha \tag{3.20c}$$

$$\hat{\theta} \in [0, 2\pi] \tag{3.20d}$$

$$\theta \in [-4\pi, 4\pi] \tag{3.20e}$$

$$\alpha \in \{-2, -1, 0, 1\} \tag{3.20f}$$

From (3.20), we have changed the domain of the trigonometric functions from $[-4\pi, 4\pi]$ to $[0, 2\pi]$. For this new domain, we can use the smaller shared partition $p = (0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}, 2\pi)$, which only has 5 points (see Figure 3.5) and so the corresponding MILP relaxation will consist of 3 binary variables and 8 non-negative continuous variables. In exchange, we have introduced an integer variable α which may take on 4 possible values. Despite the number of binary variables being reduced from 17 to 3, the introduction of α may not lead to improved computational times. This is because α may require multiple branching steps when solving the MILP, which may prove to be too computationally expensive. Even so, this example clearly illustrates the potential benefit of using this reformulation.

In addition to reducing the number of variables in the resulting MILP relaxation, the presented reformulation effectively allows the relaxation over multiple sub-domains to be tightened simultaneously. For example, once again consider (3.20). In the original formulation, the functions are relaxed over the domain $[-4\pi, -2\pi] \cup [-2\pi, 0] \cup [0, 2\pi] \cup [2\pi, 4\pi]$. Conversely, in the principal domain reformulation the functions are relaxed over the single principle domain $[0, 2\pi]$. Now, consider the shared (effective base) partition $p = (0, \frac{\pi}{2}, \pi, \frac{3\pi}{2}, 2\pi)$ of $[0, 2\pi]$. If we are now to refine p using any of the previously described refinement schemes, the refinement will be equivalent to adding a partition point to each of the previous sub-domains simultaneously at the expense of a single partition point and the use of the integer variable α in (3.20). More generally, for T-periodic functions, adding a partition point in a principle domain is equivalent to adding a partition point in every sub-domain of width T that has been shifted by αT from the principal domain. In the original formulation, adding multiple partition points to any of the sub-domains may prove to be expensive if the optimal solution lies in a different sub-domain. However, in the principal domain reformulation any added partition point will be beneficial as it will impact *all* of the sub-domains simultaneously.

Even with the benefits of reformulating the problem to use principal domains, computational studies need to be conducted to determine if this formulation will lead to reduced computational times for the user-specific application at hand. The need for the integer variables α may ultimately lead to longer solve times despite these benefits due to the branching needed to identify the optimal value of α.

3.6.3 Choice of Principal Domain

As previously mentioned, there are many choices for the principal domain to be used for the reformulation in general. This is because we may always shift an interval of width T by any amount to produce another principal domain (assuming the function remains properly defined). The choice of principal domain has a direct impact on the performance of the reformulation. To see this, consider $y = \sin(\theta)$ with $\theta \in [-\pi, 3\pi]$. For the principal domain $[0, 2\pi]$, we find $\alpha^L = -1$ and

$\alpha^U = 1$, and so $\alpha \in \{-1, 0, 1\}$. However, for the principal domain $[-\pi, \pi]$, we find $\alpha^L = 0$ and $\alpha^U = 1$ and so $\alpha \in \{0, 1\}$. Clearly the principal domain $[-\pi, \pi]$ is superior in this case since α only has two possible values, which will likely reduce the computational effort in solving the resulting MILP relaxation. Therefore, care should be taken when choosing a principal domain for reformulation.

3.6.4 Relating Principal Domain Variables

Consider two T-periodic functions $f_1(\theta_1)$ and $f_2(\theta_2)$ with the same principal domain $[\hat{\theta}^L, \hat{\theta}^U]$ and with original domains $I_1 = [\theta_1^L, \theta_1^U]$ and $I_2 = [\theta_2^L, \theta_2^U]$. We will denote by α_1 and α_2 the integer variables introduced for θ_1 and θ_2 when implementing the principal domain reformulations for f_1 and f_2. Suppose θ_1 and θ_2 are related by a linking constraint $\theta_2 = h(\theta_1)$. We will assume h is continuous and bounded with lower and upper bounds K_1 and K_2, respectively. We will assume $I_1 \subseteq I_2$, i.e., I_2 is the result of expanding I_1 a finite amount. Such a case occurs when θ_1 and θ_2 both represent the accumulation of some quantity (time, angle, cost, etc.) in one stage of a process for θ_1 and a subsequent stage for θ_2. We will also assume $K_1 \in I_2$ and $K_2 \in I_2$. In the case $K_1 \notin I_2$ (resp. $K_2 \notin I_2$), we will be unable to improve the lower bound (resp. upper bound) of α_2. Define

$$\gamma^U = K_2 - \theta_1^U \tag{3.21}$$

$$\gamma^L = \theta_1^L - K_1 \tag{3.22}$$

where γ^U (resp. γ^L) is the maximum amount θ_2 can deviate from the maximum (resp. minimum) of I_1. Recall in the principal domain reformulation the introduced α integer variable represents how many steps of width T are needed to reach the original variable's value from the chosen principal domain. With this in mind, γ^U and γ^L give a tighter bound on the number of steps of width T needed to capture all possible values of θ_2 with respect to any value of θ_1 as a result of the linking constraint $\theta_2 = h(\theta_1)$. More specifically, we have

$$-\left\lceil \frac{\gamma^L}{T} \right\rceil \leq \alpha_2 - \alpha_1 \leq \left\lceil \frac{\gamma^U}{T} \right\rceil \tag{3.23}$$

As a simple example, consider $y_1 = \sin(\theta_1)$ and $y_2 = \sin(\theta_2)$ with $\theta_1 \in [-2\pi, 2\pi]$ and $\theta_2 \in [-4\pi, 4\pi]$. We will suppose θ_1 and θ_2 are related by the linking constraint $\theta_2 = \theta_1 + b$ with $0 \leq b \leq 2\pi$. This linking constraint may represent a physical constraint on a vehicle i.e., it may not turn more than an angle of 2π in addition to the initial angle θ_1 in the stage corresponding to θ_2. Let $[0, 2\pi]$ be the chosen principal domain for both θ_1 and θ_2. Without any modifications, the principal domain reformulations would give

$$y_1 = \sin(\hat{\theta}_1) \tag{3.24a}$$

$$y_2 = \sin(\hat{\theta}_2) \tag{3.24b}$$

$$\hat{\theta}_1, \hat{\theta}_2 \in [0, 2\pi] \tag{3.24c}$$

$$\theta_1 \in [-2\pi, 2\pi] \tag{3.24d}$$

$$\theta_2 \in [-4\pi, 4\pi] \tag{3.24e}$$

$$\alpha_1 \in \{-1, 0\} \tag{3.24f}$$

$$\alpha_2 \in \{-2, -1, 0, 1\} \tag{3.24g}$$

We now note by the linking constraint $K_1 = -2\pi$ and $K_2 = 4\pi$. From this we find $\gamma^L = 0$ and $\gamma^U = 1$. Therefore, we may add the additional constraints

$$0 \leq \alpha_2 - \alpha_1 \leq 1 \tag{3.25}$$

In words, (3.25) states the value of θ_2 is either the same number of 2π steps from $[0, 2\pi]$ as θ_1 or is one step further in the increasing direction. For this particular example, this reduces (3.24g) to $\alpha_2 \in \{-1, 0, 1\}$. In general, a direct reduction like this may not always happen. However, the addition of (3.25) introduces information to a solver when the value of α_1 has been determined. For example, suppose (3.24) is relaxed using the previously described polyhedral relaxations and is being solved by a MILP solver using branch-and-bound. In one of the branches, we may have $\alpha_1 = -1$. If we include the constraint (3.25), we then know $-1 \leq \alpha_2 \leq 0$ holds in this branch, halving the number of possible values of α_2 from the (3.24). As a result, if the MILP solver must branch once more, only two values of α_2 need to be considered (in this branch). This may potentially lead to reduced computation times for solving the MILP. Attempting to take advantage of the branching decisions when solving the MILP corresponding to the principal domain reformulation is the subject of the discussion that immediately follows.

3.6.4.1 Branching Decisions

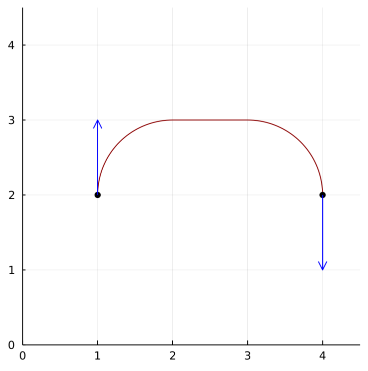
Consider univariate, bounded, continuous, trigonometric functions $f_1(x_1), \ldots, f_m(x_m)$ each with period T that have been relaxed using the polyhedral relaxations of Section 3.4. The following discussion will hold for general T-periodic functions that are relaxed and solved using a branch-and-bound procedure. Furthermore, suppose $x_1, \ldots, x_m$ are linked sequentially, i.e., $x_2 = h_1(x_1), x_3 = h_2(x_2), \ldots, x_m = h_{m-1}(x_{m-1})$ where each h_i is bounded and continuous. As previously described, these linking constraints result in inequalities of the form (3.23) for consecutive variables α_i and α_{i+1} when using the same principal domain. When solving the corresponding MILP with these added inequalities, we may elect to branch on each α_i (corresponding to x_i) in sequential order. That is, when branching to solve the MILP we first consider α_1, then α_2, and so on until α_m. Consider a branch where we have set α_i, $i < m$, to be equal to some value. By the linking constraint $x_{i+1} = h_i(x_i)$, the resulting inequality of form (3.23) for α_i and α_{i+1} may reduce the number of values to consider for α_{i+1}. Consequently, if we require to branch once more we may do so on α_{i+1} and require fewer children subproblems for this branch as a result. This will ultimately reduce the number of subproblems needed to be solved in the extreme case of an exhaustive search (by effectively avoiding generating many infeasible subproblems due to the various values of α_i). In the case where additional linking constraints between non-consecutive variables are present, a similar approach can be taken to identify a potentially stronger branching scheme. These other cases are not discussed here.

3.7 Motivating Example - Markov-Dubins Path Planning Problem

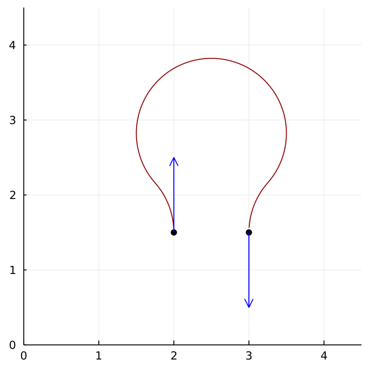
The Markov-Dubins path planning problem [10] (MDPPP) is a natural extension of the classical two-point Markov-Dubins problem [8]. The MDPPP may be formulated as follows. Consider a vehicle travelling in the plane starting at a point $p_1 = (x_1, y_1) \in \mathbb{R}^2$, passing through $n-2$ intermediate points $p_i = (x_i, y_i) \in \mathbb{R}^2$, $i = 2, \ldots, n-1$, and arriving at a final point $p_n = (x_n, y_n) \in \mathbb{R}^2$, in the sequence $(p_1, \ldots, p_n)$. The vehicle may or may not have a specified initial and final heading angle $\theta_0 \in [0, 2\pi]$ and $\theta_n \in [0, 2\pi]$ at p_1 and p_n, respectively. The vehicle is a Dubins vehicle [8] and so it has a minimum turning radius $\rho > 0$ and travels at constant speed. The MDPPP is then to find a path of minimum length passing through the points in the specified sequence with radius of curvature at least ρ everywhere from p_1 to p_n.

In Dubins' seminal work [8], it was shown the optimal path in the case where $n = 2$ is of type CCC or CSC where C denotes a circular arc segment (left or right turn) of radius ρ and S denotes a straight line segment. For the circular arc segments of radius ρ, let a left turn (counter-clockwise) be denoted by L and a right turn (clockwise) be denoted by R. For $n = 2$, the optimal solution is then one of the following: LRL, RLR, LSL, LSR, RSR, or RSL. Examples of RSR, LRL, and LSR are shown in Figure 3.9. Examples of LSL, RLR, and RSL are obtained by reversing the paths shown in Figure 3.9. We note that a segment in the optimal solution may have zero length. By Bellman's principle of optimality [57], the subpath connecting two consecutive points in the optimal solution to the MDPPP must then also be of type CCC or CSC. An example solution to the MDPPP for $n = 3$ is shown in Figure 3.10.

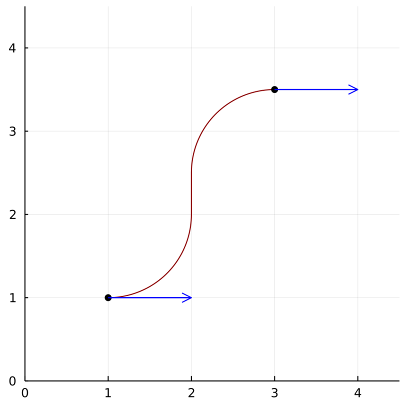
Following the work of [10], we can formulate the MDPPP as a NLP as follows. We refer to the subpath connecting consecutive points p_i and p_{i+1} as stage i. There are $N = n - 1$ stages. We note that the word describing the optimal subpath at stage i is a subset of the sequence (L, R, S, L, R). For each stage i, define $\xi_j^i \geq 0$, where $j = 1, \ldots, 5$ denotes which letter (L, R, or S) in the sequence (L, R, S, L, R) and ξ_j^i represents the length of the corresponding segment type. (For example, if the path taken at stage i is LRL, then $\xi_3^i = \xi_5^i = 0$ and $\xi_1^i, \xi_2^i, \xi_4^i \geq 0$.) For $j \in \{1, 2, 4, 5\}$, define θ_j^i to be heading angle of the vehicle *at the end* of the corresponding segment (L or R) at stage i.



(a) RSR



(b) LRL



(c) LSR

Figure 3.9: Example of optimal Dubins paths when $n = 2$. Examples of the remaining optimal words are obtained by reversing the initial and final heading directions (blue arrows).

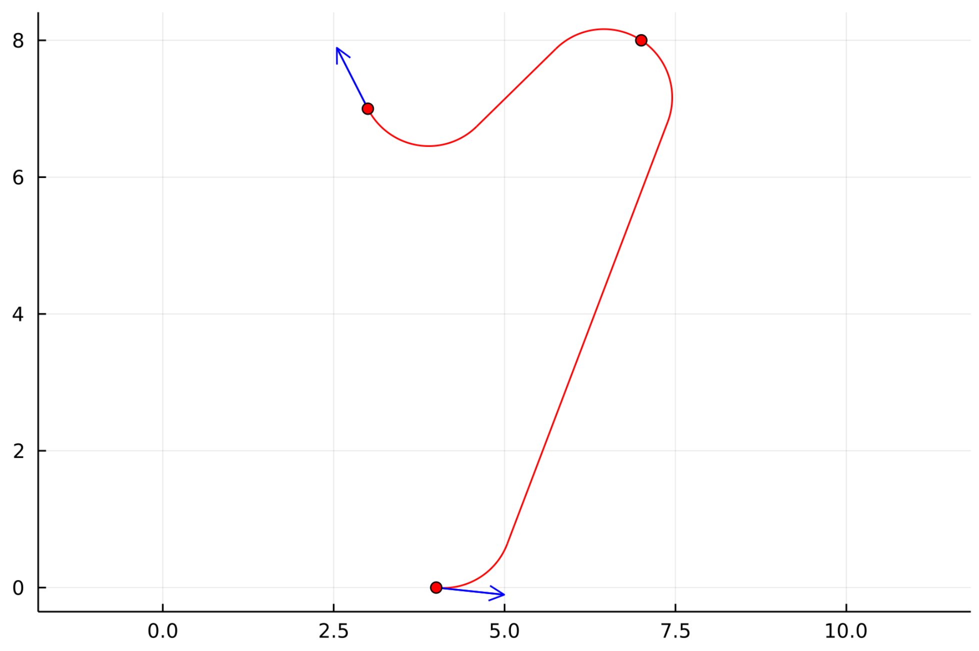


Figure 3.10: Optimal solution for MDPPP with $n = 3$. The path is LSL followed by LSR.

Let θ_0^i be the heading angle at the start of stage i, i.e., the heading angle when the vehicle reaches point p_i or the initial heading angle in the case of p_1. Let $\mathcal{S} = \{1, \dots, N\}$. The NLP formulation [10] (after factoring) of the MDPPP is then:

$$(MDPPP) \qquad \text{minimize} \sum_{i=1}^{N} \sum_{j=1}^{5} \xi_j^i \tag{3.26a}$$

$$\text{s.t.} \quad x_i - x_{i+1} + \rho \left[-w_0^i + 2w_1^i - 2w_2^i + 2w_4^i - w_5^i \right] + \mu^i = 0, \quad \forall i \in \mathcal{S} \tag{3.26b}$$

$$y_i - y_{i+1} + \rho \left[z_0^i - 2z_1^i + 2z_2^i - 2z_4^i + z_5^i \right] + \nu^i = 0, \quad \forall i \in \mathcal{S} \tag{3.26c}$$

$$w_j^i = \sin(\theta_j^i), \quad \forall i \in \mathcal{S}, \quad j \in \{0, 1, 2, 4, 5\} \tag{3.26d}$$

$$z_j^i = \cos(\theta_j^i), \quad \forall i \in \mathcal{S}, \quad j \in \{0, 1, 2, 4, 5\} \tag{3.26e}$$

$$\mu^i = \xi_3^i z_2^i, \qquad \forall i \in \mathcal{S} \tag{3.26f}$$

$$\nu^i = \xi_3^i w_2^i, \qquad \forall i \in \mathcal{S} \tag{3.26g}$$

$$\begin{aligned} &\theta_1^i = \theta_0^i + \rho^{-1}\xi_1^i, \qquad \theta_2^i = \theta_1^i - \rho^{-1}\xi_2^i, \\ &\theta_4^i = \theta_2^i + \rho^{-1}\xi_4^i, \qquad \theta_5^i = \theta_4^i - \rho^{-1}\xi_4^i, \quad \forall i \in \mathcal{S} \end{aligned} \tag{3.26h}$$

$$w_0^{i+1} = w_5^i, \qquad i = 1, \dots, N-1 \tag{3.26i}$$

$$z_0^{i+1} = z_5^i, \qquad i = 1, \dots, N-1 \tag{3.26j}$$

$$\begin{aligned} &\theta_0^i \in [0, 2\pi], \qquad \theta_1^i \in [0, 4\pi] \\ &\theta_2^i, \theta_4^i \in [-2\pi, 4\pi], \quad \theta_5^i \in [-4\pi, 4\pi], \quad \forall i \in \mathcal{S} \end{aligned} \tag{3.26k}$$

$$\theta_0^1 = \theta_0, \qquad w_5^N = \sin(\theta_n), \qquad z_5^N = \cos(\theta_n) \tag{3.26l*}$$

$$\xi_j^i \geq 0, \qquad \forall i \in \mathcal{S}, \quad j = 1, \dots, 5 \tag{3.26m}$$

Objective (3.26a) says to minimize the total path length from p_1 to p_n. Constraints (3.26b) and (3.26c) represent the horizontal and vertical displacement, respectively, after taking a $LRSLR$ path [58, 10] from p_i to p_{i+1} for each stage i, where the trigonometric and bilinear terms have been factored out with constraints (3.26d)-(3.26e) and (3.26f)-(3.26g), respectively. The left and right turns of radius ρ are characterized by constraints (3.26h), where an increase (resp. decrease) in angle from one segment to another corresponds to a left (resp. right) turn. Constraints (3.26i)-(3.26j) ensure the slope of the path at the end of stage i and at the beginning of stage $i+1$ are equal, i.e., continuity is maintained.

We have chosen to use this problem to illustrate the effectiveness of our proposed algorithm for several reasons. Firstly, all nonlinearities present in (3.26) are trigonometric (sine and cosine) or bilinear. The MDPPP provides a natural example where sharing partitions for both trigonometric and bilinear terms is possible. The trigonometric terms are defined over closed intervals given by (3.26k). The bilinear terms are also capable of being defined over closed intervals by using knowledge of the optimal solutions to the MDPPP (see Section 3.7.1.1). Secondly, by the nature of the problem the box constraints in (3.26k) are, in some sense, relatively loose. This motivates the idea of using principal domains to replace the original domains of the trigonometric terms. Finally, the complexity of the MDPPP will likely increase the influence of the chosen refinement schemes and refinement strategy on the overall time to solve. This will help in comparing these refinement schemes and refinement strategies.

3.7.1 Additional Constraints

We now introduce some additional constraints that can be added to the MINLP formulation of the MDPPP in order to reduce computation time. These are included to ensure the computational times stay within a reasonable range.

3.7.1.1 Bounding Arc Lengths

In order to apply the MILP relaxations for the trigonometric terms (3.26d)-(3.26e) and the bilinear terms (3.26f)-(3.26g), we require the arc lengths of the straight line segments, ξ_3^i, be bounded above (since we require a closed interval). Consider two consecutive points p_i and p_{i+1} in the MDPPP. It can be shown that in an optimal solution of the MDPPP we must have

$$\xi_3^i \leq \|p_i - p_{i+1}\| + 4\rho, \quad \forall i \in \mathcal{S} \tag{3.27}$$

where $\|\cdot\|$ denotes the Euclidean norm. We may also bound the circular arc segments by noting that taking more than a full circle in any circular segment will never be optimal. Therefore,

$$\xi_j^i \leq 2\pi\rho, \quad \forall i \in \mathcal{S}, \quad j \in \{1,2,4,5\} \tag{3.28}$$

3.7.1.2 Restricting the Number of Segments

As previously mentioned, the sub-path for two consecutive points in the optimal solution to the MDPPP must be of type CCC or CSC. In the current formulation, we are not strictly requiring at most three segments be used at each stage. This is because the optimal solution will automatically satisfy this property. In order to reduce the computation time needed to solve the MDPPP, we can add constraints enforcing at most three segments are used at each stage. For each stage i and segment j, let β_j^i be a binary variable where $\beta_j^i = 1$ if segment j is used in stage i and $\beta_j^i = 0$ otherwise. We can then add the constraints

$$\sum_{j=1}^{5} \beta_j^i \leq 3, \quad \forall i \in \mathcal{S} \tag{3.29}$$

and replace the upper bounds (3.27) and (3.28) by

$$\xi_j^i \leq M_j^i \beta_j^i, \quad \forall i \in \mathcal{S}, \quad j = 1, \dots, 5 \tag{3.30}$$

where the constant M_j^i represents an upper bound for the arc length ξ_j^i as described in (3.27) and (3.28). If tighter bounds are available, they could be easily substituted into (3.30). By including constraints (3.29) and (3.30), the formulation for the MDPPP becomes an MINLP.

3.7.1.3 CSC Conditions

We next take advantage of two results that limit the type of sub-paths in an optimal solution of the MDPPP for two or more consecutive points.

Consider two points p_i and p_{i+1} and suppose the Euclidean distance between the two points is at least 4ρ. It can be shown the optimal Dubins path from p_i and p_{i+1} cannot be of type CCC. Suppose points p_i and p_{i+1} in the MDPPP, corresponding to stage i, are at least 4ρ from each other. We may then introduce the following constraints.

$$\beta_1^i + \beta_2^i \leq 1 \tag{3.31}$$

$$\beta_3^i = 1 \tag{3.32}$$

$$\beta_4^i + \beta_5^i \leq 1 \tag{3.33}$$

Recall β_1^i, β_4^i and β_2^i, β_5^i correspond to a left turn and a right turn, respectively, in the sequence (L, R, S, L, R). Similarly, β_3^i corresponds to a straight line segment. Constraints (3.31) and (3.33) then simply says at most two turns are taken between points p_i and p_{i+1}. Additionally, constraint (3.32) requires a straight line segment be used, though it may be a degenerate case with zero length (i.e., we still allow for $\xi_3^i = 0$).

Constraints (3.31)-(3.33) impose additional constraints for a single stage due to optimality conditions from the physics of the problem. We can go further by also considering consecutive stages. In [10], the following result was derived using optimal control theory for the MDPPP. Consider two consecutive stages i and $i+1$ and suppose the Euclidean distance between consecutive points is at least 4ρ for both stages. Therefore, each stages will only admit a CSC path in the optimal solution to the MDPPP. Under these conditions, we can make use of two results in [10] (see Theorem 4 and Proposition 2 therein) to get the following corollary.

Corollary 3.7.1. Suppose two consecutive stages i and $i+1$ are such that both only admit CSC paths in the optimal solution to the MDPPP. Then the type (L or R) of the final turn in stage i must be the same type (L or R) as the first turn in stage $i+1$. Furthermore, the length of the final turn in stage i and the length of the first turn in stage $i+1$ must be equal. Additionally, the first turn (L or R) in stage $i+1$ has length less than $\pi\rho$.

From Corollary 3.7.1, we get the following additional constraints for consecutive stages admitting only CSC paths

$$\beta_4^i = \beta_1^{i+1} \tag{3.34}$$

$$\beta_5^i = \beta_2^{i+1} \tag{3.35}$$

$$\xi_4^i = \xi_1^{i+1} \tag{3.36}$$

$$\xi_5^i = \xi_2^{i+1} \tag{3.37}$$

$$\xi_1^{i+1} + \xi_2^{i+1} \leq \pi\rho \tag{3.38}$$

Constraints (3.31)-(3.38) can be added to the formulation for the MDPPP by a pre-processing step before solving. We note that constraint (3.38) could have also been written as

$$\xi_4^i + \xi_5^i \leq \pi\rho \tag{3.39}$$

by the first statement in Corollary 3.7.1.

3.8 Computational Results

In this section we present computational results related to the MDPPP presented in Section 3.7.

3.8.1 Problem Generation

Each instance consists of n points ranging from $n = 2$ to $n = 8$ in a 10×10 grid. For each value of n, ten instances were generated by randomly placing the n points with the additional requirement that consecutive points are separated by a Euclidean distance of at least 4ρ where $\rho = 1$ for all instances. This additional requirement was enforced to reduce the overall computation time needed for all instances by making use of the additional constraints listed in Section 3.7.1. The initial and final heading angles were randomly chosen from $[0, 2\pi]$ for each instance.

3.8.2 Implementation Details

All instances were solved on a 64-bit Windows computer with an AMD Ryzen 7 2700 processor at 3.2 GHz with 16 GB of RAM (2400 MHz) using Julia v1.6.3 [21]. All MILPs were modelled using the JuMP (v1.2.0) modelling package [22] for mathematical programs in Julia. All MILPs were solved using CPLEX 20.1 with default settings. A solution was taken to be optimal when the relative gap was under 1 percent and all instances were given a 1 hour time limit. Warm-starting was used for all instances to improve computational time. All variables except for the binary and non-negative continuous variables involved in the polyhedral relaxations were warm-started at each iteration of the algorithm. Shared partitions were used whenever possible.

In order to tighten the bounds on the heading angles and arc lengths in the MINLP model of the MDPPP, we used feasibility-based bounds tightening (FBBT) [59]. In general, FBBT may be used multiple times (potentially an infinite number of times), successively tightening the bounds for some or all variables while approaching a fixed point. However, for this particular MINLP it was found that FBBT only needed to be used once before reaching a fixed point. We note that this does not mean the bounds are as tight as possible, but rather the bounds could no longer be tightened using feasibility information.

As mentioned in Section 3.2, we require a procedure for generating a feasible solution given a solution to the MILP relaxation at each iteration of the algorithm. To do this, from the MILP solution we take the heading angle at the start of each stage (i.e., θ_0^i for $i = 1, \ldots, N$) and the final heading angle and compute the shortest Dubins path between these points using these heading angles. This provides a feasible solution that can be used to update the best-known upper bound. In general, this procedure will not always produce a tighter upper bound at each iteration.

3.8.3 Results

3.8.3.1 Original Formulation vs. Principal Domain Reformulation

We first compare results for the original formulation (3.26) with results for the MDPPP where the trigonometric terms and angle bounds have been replaced using the principal domain reformulation discussed in Section 3.6.1. For each angle, a principle domain of $[0, 2\pi]$ was chosen for simplicity. The results for the original MDPPP formulation and the reformulation using principal domains are shown in Table 3.1 and Table 3.2, respectively. The first column '**instance**' has entries of the form $n - q$ where n is the number of points and q is the instance number for that value of n. Column '**t**' indicates the time (in seconds) for the algorithm to terminate (either due to a relative gap below 1% or the time limit being exceeded). Entries '**' indicate an instance exceeded the 1 hour time limit. Column '**iter.**' indicates the number of refinement iterations performed by the algorithm. Column '**bin**' indicates the number of binary variables added to the formulation due to refinement. Equivalently, entries in the '**bin**' column correspond to the number of partition points added due to refinement. The top-most row in each table lists which refinement scheme (bisection, direct, non-uniform two-point, non-uniform three-point) was used for the instances. For brevity, we will refer to the non-uniform two-point and non-uniform three-point refinement schemes with NU2 and NU3, respectively. For both Table 3.1 and Table 3.2, the complete refinement strategy was used (i.e., all partitions were refined at each refinement iteration of the algorithm).

In Table 3.1, it can be seen most instances are solved within the 1 hour time limit with the exception of instances 7-2 and 7-7 using NU3. We also see the instances with $n = 7$ take significantly longer than the instances with $n = 6$ using any of the proposed refinement schemes. Simultaneously, the $n = 7$ instances and $n = 6$ instances appear to take roughly the same number of refinement iterations to reach the desired relative gap. It also appears the number of added binary variables is not the primary bottleneck, as the $n = 6$ and $n = 7$ instances have comparable values. This seems to suggest the formulation itself may be causing the sudden drastic increase in computational time. In particular, it is likely that solving the MILP relaxation of the original formulation (3.26) is difficult due to ambiguity arising from not taking into account the periodic nature of the trigonometric terms over their corresponding angle domains. As n increases, the influence of this ambiguity increases, leading to a significant jump in the computational effort required. We also note in Table 3.1 the NU2 refinement scheme appears to perform most consistently as n increases.

In Table 3.2, it can be seen all instances up to $n = 7$ are solved within the 1 hour time limit. Additionally, we see the increase in computational time from $n = 6$ to $n = 7$ is more tame than in Table 3.1. This is especially true when using the NU2 or NU3 refinement schemes. As was the case in Table 3.1, the number of refinement iterations and the number of added binary variables are comparable for the $n = 6$ and $n = 7$ instances. This then suggests the primary contributing factor in limiting the jump in computational effort required is the reformulation using principal domains. In doing so, we have simplified the relationship between angles in consecutive stages. We remark that the computational times in Table 3.2 are dependent on the principal domain chosen for each angle and so choosing a different set of principal domains may lead to better times. For the sake of this article, Table 3.2 is sufficient to show advantage of using principal domains in place of large angle domains. In Table 3.2, we see instances with $n = 8$ are significantly more difficult than the $n = 7$ instances. Each refinement scheme for $n = 8$ has one or more instances exceeding the 1 hour time limit. We also see of the four proposed refinement schemes, NU2 performed the best for $n = 8$.

Table 3.1: Original formulation results.

	Bisection			Direct			NU2			NU3		
instance	**t**	**iter**	**bin**	**t**	**iter.**	**bin**	**t**	**iter.**	**bin**	**t**	**iter.**	**bin**
5–1	39.09	4	89	45.65	3	67	24.74	2	92	54.18	2	137
5–2	85.4	5	115	76.59	4	86	139.88	4	183	93.58	3	206
5–3	35.13	4	91	42.37	4	85	62.43	3	138	60.06	3	203
5–4	140.27	6	134	104.27	5	102	85.26	4	173	168.74	4	254
5–5	56.74	5	110	60.9	4	85	66.27	3	132	77.51	3	195
5–6	97.83	6	110	66.77	4	75	74.34	3	116	113.93	3	172
5–7	38.72	5	108	39.12	4	86	29.6	3	132	66.09	3	198
5–8	50.78	4	89	43.51	4	86	37.44	3	135	86.52	3	200
5–9	44.68	4	91	57.72	3	64	69.1	3	134	49.43	3	198
5–10	76.82	5	107	51.38	4	78	76.73	3	120	98.62	3	188
–	–	–	–	–	–	–	–	–	–	–	–	–
6–1	134.86	5	142	101.05	4	104	98.04	3	168	135.99	3	244
6–2	214.27	5	141	196.11	4	103	181.13	3	165	252.54	3	248
6–3	207.58	4	114	142.98	3	81	178.67	3	173	256.54	3	249
6–4	468.18	6	173	424.05	5	132	425.33	3	174	416.8	3	256
6–5	491.81	7	197	414.65	5	136	267.73	3	168	190.72	3	243
6–6	303.83	6	160	383.62	6	141	346.78	4	212	178.66	3	227
6–7	146.63	5	131	160.59	4	94	187.29	3	152	190.62	3	223
6–8	460.59	6	174	736.12	5	140	313.04	4	232	220.05	3	260
6–9	354.27	6	166	262.23	5	133	435.09	4	219	271.96	4	326
6–10	192.72	5	135	262.55	4	95	227.03	3	162	127.68	2	150
–	–	–	–	–	–	–	–	–	–	–	–	–
7–1	204.76	4	137	551.01	4	133	317.56	3	204	345.17	2	201
7–2	669.32	5	166	1974.86	5	157	572.96	3	200	**	1	102
7–3	516.44	4	140	781.15	4	135	1781.79	3	210	803.68	3	309
7–4	963.81	4	134	158.51	4	130	1047.18	3	199	1849.16	3	292
7–5	2072.35	4	136	2476.62	4	136	1270.12	3	204	2816.3	3	305
7–6	1229.08	6	205	2953.63	5	164	1518.99	3	204	2655.86	3	302
7–7	2500.93	5	173	3307.15	5	162	2604.86	4	276	**	2	205
7–8	506.25	5	172	638.5	4	129	566.41	3	210	1282.62	3	311
7–9	1317.84	6	200	515.22	4	130	645.33	3	188	490.54	3	291
7–10	904.65	5	171	2296.51	5	149	607.45	3	202	290.43	2	197

Table 3.2: Principal domain results.

	Bisection			Direct			NU2			NU3		
instance	**t**	**iter**	**bin**	**t**	**iter.**	**bin**	**t**	**iter.**	**bin**	**t**	**iter.**	**bin**
6–1	56.24	5	142	28.5	3	80	31.4	3	163	29.36	2	161
6–2	169.22	5	141	76.79	3	83	74.58	3	166	46.82	2	162
6–3	101.59	4	116	76.76	3	85	63.3	3	174	53.1	2	174
6–4	126.45	5	145	155.6	4	114	80.92	3	174	181.87	3	260
6–5	214.7	7	194	134.05	5	120	117.5	4	207	90.14	3	232
6–6	99.11	6	171	95.22	5	126	49.29	3	168	85.19	3	245
6–7	82.16	5	143	32.86	3	81	53.2	3	174	39.98	2	171
6–8	131.67	5	145	173.92	5	130	172.37	4	232	210.71	4	339
6–9	175.75	6	174	75.79	4	106	65.78	3	174	204.48	3	253
6–10	145.21	5	145	95.89	4	111	58.22	3	174	79.63	3	253
–	–	–	–	–	–	–	–	–	–	–	–	–
7–1	2205.12	5	202	223.02	4	133	267.15	3	206	72.85	2	204
7–2	345.13	5	175	284.44	4	122	271.33	3	208	385.73	3	302
7–3	308.31	4	140	224.76	4	134	86.68	2	140	101.16	2	210
7–4	438.38	4	140	179.29	3	102	366.9	3	210	194.21	2	207
7–5	1279.65	4	140	1567.77	4	131	322.18	2	140	293.36	2	210
7–6	719.48	6	207	1012.66	4	129	1155.93	3	198	609.64	2	201
7–7	507.65	4	140	813.75	4	132	323.24	3	209	824.05	3	306
7–8	134.71	5	175	285.71	4	126	122.32	3	207	313.16	3	299
7–9	198.46	4	139	198.32	4	133	184.81	3	206	142.63	2	203
7–10	303.46	5	169	811.68	5	138	168.97	3	198	726.83	3	289
–	–	–	–	–	–	–	–	–	–	–	–	–
8–1	2207.29	5	202	**	3	115	1405.03	3	240	1010.79	3	351
8–2	805.63	5	205	933.15	3	116	795.59	3	246	**	3	364
8–3	664.52	5	202	1209.03	4	143	649.35	3	240	1026.15	3	350
8–4	977.74	5	199	442.52	3	113	895.21	3	232	227.54	2	228
8–5	2666.26	6	241	1997.71	4	151	3530.79	4	318	**	2	231
8–6	**	3	122	**	3	118	**	3	244	**	2	243
8–7	963.75	4	162	1204.89	3	120	928.82	3	242	629.7	2	240
8–8	**	6	245	3113.63	4	152	2417.68	4	324	2284.58	3	355
8–9	2965.78	5	199	2305.21	4	155	1157.14	3	246	857.99	2	243
8–10	795.74	5	202	2614.42	4	143	1849.28	3	236	656.73	3	333

3.8.3.2 Complete Refinement Strategy vs. k-Worst Refinement Strategy

We next compare the complete refinement strategy with the k-worst refinement strategy. Due to the results in Table 3.1 and Table 3.2, we use the principal domain reformulation for all subsequent instances. For each refinement scheme, we vary the value of k for the k-worst refinement strategy.

In particular, we take k to be 25%, 50%, and 100% of all available partitions for each instance (rounding up when necessary). Note that the 100% case corresponds to the complete refinement strategy. The results for the bisection, direct, NU2, and NU3 refinement schemes are shown in Tables 3.3-3.6, respectively. For each refinement scheme, we restrict our attention to $n \geq 6$, since the influence of the refinement strategy for smaller n is minimal.

In Table 3.3, we see all three refinement strategies are relatively equal in overall performance when using bisection, with $k = 25\%$ performing slightly worse on average when compared to the other strategies. It is not surprising that $k = 25\%$ performs slightly worse, since it can be seen in Table 3.3 it typically requires more refinement iterations than the other strategies.

In Table 3.4, we see when using the direct refinement scheme the k-worst strategy with $k = 50\%$ outperforms the other strategies. Additionally, we see when $k = 50\%$ outperforms the other strategies, it tends to do so by a fair margin. We once again see $k = 25\%$ typically requires more refinement iterations overall, which is a major contributor to the overall solve time. We also see $k = 50\%$ and the complete refinement strategy have comparable refinement iterations for nearly all instances in Table 3.4. In the few instances for $n = 8$ where the complete refinement strategy outperforms $k = 50\%$, we see the complete refinement strategy required the same or less refinement iterations than $k = 50\%$.

In Table 3.5, we see when using the non-uniform two-point refinement scheme the k-worst strategy with $k = 50\%$ once again outperforms the other strategies. As was the case with the direct refinement scheme, when $k = 50\%$ outperforms the other strategies it tends to do so by a noticeable margin. This is especially clear in instances 8-2, 8-3, 8-5, 8-8, 8-9, and 8-10 in Table 3.5. As before, we see $k = 25\%$ typically requires more refinement iterations than $k = 50\%$ or the complete refinement strategy. For this particular refinement scheme, we see for $n = 8$ the $k = 25\%$ strategy performs worse than $k = 50\%$ in all solved instances. This suggests as n increases, these additional refinement iterations require too much time to justify the benefit of reducing the number of additional binary variables. We also see the difference in the number of added binary variables between $k = 25\%$ and $k = 50\%$ is not especially large in the $n = 8$ instances. Conversely, for smaller n the $k = 25\%$ strategy is able to reach a 1 percent relative gap with significantly less added binary variables. This is especially noticeable for $n = 7$ in Table 3.5.

In Table 3.6, we see when using the non-uniform three-point refinement scheme we have once again $k = 50\%$ outperforming the other strategies overall. We also again see $k = 25\%$ requires more refinement iterations than the other strategies on average. It's interesting to note that the complete refinement strategy performed noticeably worse than $k = 25\%$ and $k = 50\%$, with three of the $n = 8$ instances exceeding the 1 hour time limit. Despite this, the number of binary variables added in the complete refinement strategy is comparable to the other strategies, suggesting the binary variables themselves are not the primary bottleneck (with a potential exception for instance 8-2).

Table 3.3: k-Worst Results - Bisection.

	k = 25 %			k = 50 %			k = 100 %		
instance	**t**	**iter.**	**bin**	**t**	**iter.**	**bin**	**t**	**iter.**	**bin**
5–1	19.2	4	48	21.98	4	72	20.73	4	89
5–2	35.89	6	72	30.5	5	90	39.64	5	115
5–3	27.47	6	72	25.49	5	90	15.84	4	92
5–4	39.54	6	72	33.59	5	90	43.69	5	110
5–5	20.31	6	72	21.02	5	90	33.49	5	115
5–6	44.22	7	84	33.7	6	107	39.67	6	128
5–7	36.17	7	84	16.77	4	72	16.17	4	89
5–8	24.09	5	60	28.49	5	90	21.88	4	92
5–9	32.96	6	72	15.88	4	72	20.22	4	87
5–10	29.8	6	71	24.95	5	90	32.16	5	114
–	–	–	–	–	–	–	–	–	–
6–1	36.95	5	75	55.15	5	110	56.24	5	142
6–2	169.33	6	90	154.57	5	110	169.22	5	141
6–3	104.63	6	90	84.3	4	88	101.59	4	116
6–4	137.67	7	105	123	6	132	126.45	5	145
6–5	137.7	8	119	186.52	7	152	214.7	7	194
6–6	107.04	7	105	104.82	6	132	99.11	6	171
6–7	61.79	6	90	49.15	5	110	82.16	5	143
6–8	163.79	7	105	88.28	5	110	131.67	5	145
6–9	167.17	7	104	153.99	6	132	175.75	6	174
6–10	86.41	6	90	77.08	5	110	145.21	5	145
–	–	–	–	–	–	–	–	–	–
7–1	489.04	7	126	156.76	5	135	2205.12	5	202
7–2	327.07	6	108	285.75	5	135	345.13	5	175
7–3	273.98	5	90	104.41	4	108	308.31	4	140
7–4	366.01	6	108	363.75	4	108	438.38	4	140
7–5	524.14	5	90	703.43	4	108	1279.65	4	140
7–6	725.79	7	126	615.79	6	162	719.48	6	207
7–7	894.76	7	126	560.14	5	135	507.65	4	140
7–8	236.71	6	108	183.4	5	135	134.71	5	175
7–9	175.73	6	108	208.67	5	135	198.46	4	139
7–10	877.64	6	108	164.29	5	135	303.46	5	169
–	–	–	–	–	–	–	–	–	–
8–1	1360.66	6	126	1142.86	5	155	2207.29	5	202
8–2	3362.94	8	168	1111.02	5	155	805.63	5	205
8–3	1128.49	7	147	451.66	5	155	664.52	5	202
8–4	500.09	6	126	1041.69	5	155	977.74	5	199
8–5	2789.1	7	147	**	6	186	2666.26	6	241
8–6	**	5	105	**	4	124	**	3	122

instance	**t**	**iter.**	**bin**	**t**	**iter.**	**bin**	**t**	**iter.**	**bin**
8–7	1311.64	6	126	643.22	4	124	963.75	4	162
8–8	**	6	126	2464.24	6	185	**	6	245
8–9	**	5	105	2231.56	5	155	2965.78	5	199
8–10	1862.22	7	147	1219.08	6	186	795.74	5	202

Table 3.4: k-Worst Results - Direct.

	k = 25 %			k = 50 %			k = 100 %		
instance	**t**	**iter**	**bin**	**t**	**iter.**	**bin**	**t**	**iter.**	**bin**
5–1	12.63	3	36	8.77	2	36	9.32	2	45
5–2	29.01	5	59	19.04	4	72	20.06	3	69
5–3	36.42	5	59	23.54	4	72	30.73	4	84
5–4	47.21	6	72	36.78	5	89	32.76	4	82
5–5	18.67	5	60	17.6	4	72	23.99	4	87
5–6	22.76	4	48	38.04	5	86	32.6	5	89
5–7	14.46	4	47	10.03	3	54	11.31	3	66
5–8	26.07	5	58	19.84	4	70	29.75	4	88
5–9	27.39	5	60	24.05	4	72	31.56	4	88
5–10	16.26	4	46	12.53	3	54	13.99	3	67
–	–	–	–	–	–	–	–	–	–
6–1	23.77	4	59	20.14	3	66	28.5	3	80
6–2	69.33	4	60	43.59	3	66	76.79	3	83
6–3	60.39	4	60	61.32	3	66	76.76	3	85
6–4	144.5	6	90	122.11	4	88	155.6	4	114
6–5	116.64	6	90	134.66	5	105	134.05	5	120
6–6	118.66	6	90	81.78	5	107	95.22	5	126
6–7	45.29	4	60	63.56	4	87	32.86	3	81
6–8	175.22	6	90	137.99	5	110	173.92	5	130
6–9	121.35	5	75	80.47	4	88	75.79	4	106
6–10	64.95	4	60	69.62	4	88	95.89	4	111
–	–	–	–	–	–	–	–	–	–
7–1	286.42	5	89	176.84	4	108	223.02	4	133
7–2	318.95	5	88	450.93	4	107	284.44	4	122
7–3	169.5	5	88	127.34	4	108	224.76	4	134
7–4	394.93	5	90	240.47	4	108	179.29	3	102
7–5	443.71	4	72	816.89	4	108	1567.77	4	131
7–6	322.42	5	90	576.72	4	108	1012.66	4	129
7–7	308.3	5	90	1246.79	5	134	813.75	4	132
7–8	589.12	5	90	216.34	4	108	285.71	4	126
7–9	257.31	5	90	220.79	4	108	198.32	4	133
7–10	591.49	5	88	617.41	4	105	811.68	5	138
–	–	–	–	–	–	–	–	–	–
8–1	2598.56	5	104	1941.35	4	124	**	3	115
8–2	2041.39	5	105	958.85	4	124	933.15	3	116
8–3	648.06	5	104	476.91	4	124	1209.03	4	143
8–4	516.85	4	84	289.67	3	93	442.52	3	113
8–5	2533.47	6	125	3598.72	5	155	1997.71	4	151
8–6	**	4	84	3598.84	4	122	**	3	118

instance	**t**	**iter**	**bin**	**t**	**iter.**	**bin**	**t**	**iter.**	**bin**
8–7	1342.65	5	102	928.19	3	93	1204.89	3	120
8–8	3530.37	6	125	1885.85	4	124	3113.63	4	152
8–9	2040.53	5	104	2278.59	4	122	2305.21	4	155
8–10	953.98	5	104	933.37	4	124	2614.42	4	143

Table 3.5: k-Worst Results - Non-Uniform Two-Point.

	k = 25 %			k = 50 %			k = 100 %		
instance	**t**	**iter**	**bin**	**t**	**iter.**	**bin**	**t**	**iter.**	**bin**
5–1	12.59	2	48	13.89	2	72	11.48	2	92
5–2	31.44	4	96	21.21	3	108	21.13	3	138
5–3	15	3	72	16.67	3	108	17.66	3	138
5–4	25.94	4	96	22.45	3	108	22.92	3	134
5–5	21.67	4	96	14.47	3	108	32.69	3	138
5–6	28.01	4	96	22.33	3	106	22.59	3	121
5–7	12.48	3	72	18.47	3	108	18.44	3	138
5–8	18.98	3	72	24.27	3	108	26.37	3	138
5–9	12.62	3	72	28.23	3	108	26.23	3	132
5–10	21.29	4	95	20.83	3	108	27.62	3	137
–	–	–	–	–	–	–	–	–	–
6–1	27.96	3	90	44.23	3	132	31.4	3	163
6–2	94.75	4	120	53.53	3	132	74.58	3	166
6–3	37.53	3	90	54.23	3	132	63.3	3	174
6–4	70.72	4	120	53.86	3	132	80.92	3	174
6–5	113.94	5	150	87.87	4	172	117.5	4	207
6–6	95.63	5	150	41.41	3	132	49.29	3	168
6–7	39.58	3	90	52.22	3	132	53.2	3	174
6–8	139.03	5	150	137.15	4	176	172.37	4	232
6–9	97.01	4	120	104.16	3	132	65.78	3	174
6–10	86.08	4	120	46.87	3	132	58.22	3	174
–	–	–	–	–	–	–	–	–	–
7–1	112.46	4	144	97.65	3	162	267.15	3	206
7–2	284.79	3	108	215.19	3	162	271.33	3	208
7–3	140.39	3	108	133.99	3	162	86.68	2	140
7–4	386.93	4	144	170.34	3	162	366.9	3	210
7–5	492.88	3	108	304.11	2	108	322.18	2	140
7–6	573.55	3	108	640.53	3	162	1155.93	3	198
7–7	539.99	4	144	405.53	3	162	323.24	3	209
7–8	80.34	3	108	329.16	3	162	122.32	3	207
7–9	251.79	3	108	181.09	3	162	184.81	3	206
7–10	124.6	3	108	180.97	3	162	168.97	3	198
–	–	–	–	–	–	–	–	–	–
8–1	1789.28	4	168	297.18	3	186	1405.03	3	240

instance	**t**	**iter**	**bin**	**t**	**iter.**	**bin**	**t**	**iter.**	**bin**
8–2	2985.97	4	168	1248.97	3	186	795.59	3	246
8–3	571.06	4	168	364.35	3	186	649.35	3	240
8–4	743.2	3	126	392.27	3	186	895.21	3	232
8–5	1599.23	5	210	1209.47	3	186	3530.79	4	318
8–6	**	4	168	**	3	186	**	3	244
8–7	1247.97	4	168	990.37	3	186	928.82	3	242
8–8	2555.22	5	208	1798.73	3	184	2417.68	4	324
8–9	2305.63	3	126	834.69	3	186	1157.14	3	246
8–10	467.95	4	168	297.22	3	186	1849.28	3	236

Table 3.6: k-Worst Results - Non-Uniform Three-Point.

	k = 25 %			k = 50 %			k = 100 %		
instance	**t**	**iter**	**bin**	**t**	**iter.**	**bin**	**t**	**iter.**	**bin**
5–1	11.37	2	72	13.63	2	108	15.82	2	137
5–2	46.64	4	144	37.51	3	162	42.94	3	206
5–3	25.23	3	108	32.31	3	162	27.77	3	203
5–4	29.38	3	108	37.61	3	160	43.27	3	184
5–5	22.36	3	108	23.87	3	162	27.44	3	199
5–6	25.78	3	108	28.55	3	159	54.38	3	183
5–7	19.27	3	108	12.43	2	108	15.31	2	135
5–8	26.54	3	108	33.98	3	162	40.94	3	199
5–9	28.74	3	108	29.12	3	160	30.51	3	191
5–10	18.35	3	108	14.69	2	108	16.65	2	138
–	–	–	–	–	–	–	–	–	–
6–1	29.35	3	135	21.98	2	132	29.36	2	161
6–2	150.83	3	135	49.14	2	132	46.82	2	162
6–3	57.19	3	135	41.69	2	132	53.1	2	174
6–4	64.04	3	132	94.9	3	198	181.87	3	260
6–5	122.03	4	180	74.03	3	192	90.14	3	232
6–6	104.77	4	180	61.07	3	195	85.19	3	245
6–7	66.39	3	135	41.04	2	132	39.98	2	171
6–8	127.62	4	180	159.39	4	264	210.71	4	339
6–9	159.62	4	180	110.59	3	198	204.48	3	253
6–10	67.13	3	135	62.12	3	198	79.63	3	253
–	–	–	–	–	–	–	–	–	–
7–1	137.29	3	162	204.74	3	243	72.85	2	204
7–2	312.51	3	160	333.99	3	243	385.73	3	302
7–3	151.28	3	162	186.06	3	243	101.16	2	210
7–4	220.49	3	162	561.82	3	243	194.21	2	207
7–5	1031.92	4	216	526.54	2	162	293.36	2	210
7–6	1265.01	3	162	409.81	2	162	609.64	2	201
7–7	875.97	4	216	533.14	3	243	824.05	3	306
7–8	151.35	3	162	340.35	3	243	313.16	3	299
7–9	124.13	2	108	106.54	2	162	142.63	2	203
7–10	324.76	3	162	521.24	3	243	726.83	3	289
–	–	–	–	–	–	–	–	–	–
8–1	1121.86	3	189	1101.29	3	278	1010.79	3	351

instance	**t**	**iter**	**bin**	**t**	**iter.**	**bin**	**t**	**iter.**	**bin**
8–2	2004.67	4	252	1236.84	3	278	**	3	364
8–3	1012.12	3	189	396.73	3	279	1026.15	3	350
8–4	292.09	3	189	109.86	2	186	227.54	2	228
8–5	2862.5	4	252	1583.59	3	279	**	2	231
8–6	3433.67	4	252	**	2	186	**	2	243
8–7	1187.75	3	189	385.81	2	186	629.7	2	240
8–8	**	3	189	1980.69	3	279	2284.58	3	355
8–9	2090.45	3	189	2433.41	2	186	857.99	2	243
8–10	798.72	4	252	1036.06	3	279	656.73	3	333

4. CONCLUDING REMARKS AND FUTURE WORK

This dissertation considered the assisted shortest path problem or ASPP. The ASPP consists of a primary agent traveling in an impeded environment, represented by a graph, to a destination in minimum cost while being assisted by a secondary, support agent. Two restricted variants of this problem were first considered and a method for each restricted variation was provided. Afterwards, a generalized form of this problem was formulated and an exact algorithm for this generalized form was presented for the case of an unyielding support. Noting the problem definition requires values for the travel costs between points in the environment, a procedure was presented to solve factorable mixed-integer non-linear programs with trigonometric terms and this method was applied to solving the Markov-Dubins path planning problem, which can be used to determine these desired travel costs for the graph defining the ASPP.

The ASPP as presented in this dissertation can be further extended to capture an even larger class of problems that may be used for many real-world applications. By extending the ASPP to consider multiple primary agents and multiple support agents, more complex real-world behavior can be captured. Furthermore, the interaction between the primary agent(s) and secondary agent(s) need not be limited to the servicing interaction as described in this dissertation. For example, there may be scenarios where multiple agents must be present at the same vertex before being permitted to take an edge, such as in the case of box-pushing where multiple agents work together to move a large, physical obstruction in the environment that would otherwise be impossible to move using a single agent. If multiple support agents are available, the capabilities of the support agents may differ amongst themselves and this may change how each individual support agent is able to interact with a primary agent. These extensions and many others are worth further investigation for their utility.

REFERENCES

[1] P. R. Wurman, R. D’Andrea, and M. Mountz, “Coordinating hundreds of cooperative, autonomous vehicles in warehouses,” *AI magazine*, vol. 29, no. 1, pp. 9–9, 2008.

[2] N. M. Kou, C. Peng, H. Ma, T. S. Kumar, and S. Koenig, “Idle time optimization for target assignment and path finding in sortation centers,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 9925–9932, 2020.

[3] J. Li, A. Tinka, S. Kiesel, J. W. Durham, T. S. Kumar, and S. Koenig, “Lifelong multi-agent path finding in large-scale warehouses,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 11272–11281, 2021.

[4] B. Doroodgar, M. Ficocelli, B. Mobedi, and G. Nejat, “The search for survivors: Cooperative human-robot interaction in search and rescue environments using semi-autonomous robots,” in *2010 IEEE International Conference on Robotics and Automation*, pp. 2858–2863, IEEE, 2010.

[5] K. Braekers, K. Ramaekers, and I. Van Nieuwenhuyse, “The vehicle routing problem: State of the art classification and review,” *Computers & industrial engineering*, vol. 99, pp. 300–313, 2016.

[6] T. K. Ralphs, L. Kopman, W. R. Pulleyblank, and L. E. Trotter, “On the capacitated vehicle routing problem,” *Mathematical programming*, vol. 94, pp. 343–359, 2003.

[7] H. C. Lau, M. Sim, and K. M. Teo, “Vehicle routing problem with time windows and a limited number of vehicles,” *European journal of operational research*, vol. 148, no. 3, pp. 559–569, 2003.

[8] L. E. Dubins, “On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents,” *American Journal of mathematics*, vol. 79, no. 3, pp. 497–516, 1957.

[9] Z. Chen and T. Shima, “Shortest dubins paths through three points,” *Automatica*, vol. 105, pp. 368–375, 2019.

[10] C. Y. Kaya, “Markov–dubins interpolating curves,” *Computational Optimization and Applications*, vol. 73, no. 2, pp. 647–677, 2019.

[11] Y. Lin and L. Schrage, “The global solver in the LINDO api,” *Optimization Methods & Software*, vol. 24, no. 4-5, pp. 657–668, 2009.

[12] P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Wächter, “Branching and bounds tighteningtechniques for non-convex minlp,” *Optimization Methods & Software*, vol. 24, no. 4-5, pp. 597–634, 2009.

[13] G. Nemhauser and L. Wolsey, *Integer and Combinatorial Optimization.* Wiley, 1988.

[14] M. Desrochers and F. Soumis, “A generalized permanent labelling algorithm for the shortest path problem with time windows,” *INFOR: Information Systems and Operational Research*, vol. 26, no. 3, pp. 191–212, 1988.

[15] C. Montez, S. Rathinam, S. Darbha, D. Casbeer, and S. G. Manyam, “An approximation algorithm for an assisted shortest path problem,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 8024–8030, IEEE, 2021.

[16] C. Montez, S. Rathinam, S. Darbha, and D. Casbeer, “Finding shortest paths for a team of convoy and repair vehicles,” in *AIAA Scitech 2021 Forum*, p. 1769, 2021.

[17] L. Parker, *Heterogeneous multi-robot cooperation.* PhD thesis, MIT, 1994.

[18] B. Donald, J. Jennings, and D. Rus, “Analyzing teams of cooperating mobile robots,” *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, vol. 3, pp. 1896–1903, 1994.

[19] L. Steels, “Cooperation between distributed agents through self-organization,” *IEEE International Workshop on Intelligent Robots and Systems, Towards a New Frontier of Applications*, pp. 8–14, 1990.

[20] Y. Cao, A. Fukunaga, and A. Kahng, “Cooperative mobile robotics: Antecedents and directions,” *Autonomous Robots*, pp. 7–27, 1997.

[21] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, “Julia: A fresh approach to numerical computing,” *SIAM review*, vol. 59, no. 1, pp. 65–98, 2017.

[22] I. Dunning, J. Huchette, and M. Lubin, “Jump: A modeling language for mathematical optimization,” *SIAM Review*, vol. 59, no. 2, pp. 295–320, 2017.

[23] Gurobi Optimization, LLC, “Gurobi Optimizer Reference Manual,” 2022.

[24] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[25] Z. Zhang, N. R. Sturtevant, R. Holte, J. Schaeffer, and A. Felner, “A* search with inconsistent heuristics,” in *Twenty-First International Joint Conference on Artificial Intelligence*, 2009.

[26] A. Martelli, “On the complexity of admissible search algorithms,” *Artificial Intelligence*, vol. 8, no. 1, pp. 1–13, 1977.

[27] R. Nissim and R. Brafman, “Distributed heuristic forward search for multi-agent planning,” *Journal of Artificial Intelligence Research*, vol. 51, pp. 293–332, 2014.

[28] A. Torreño, E. Onaindia, A. Komenda, and M. Štolba, “Cooperative multi-agent planning: A survey,” *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, pp. 1–32, 2017.

[29] H. Lee, J. M. Pinto, I. E. Grossmann, and S. Park, “Mixed-integer linear programming model for refinery short-term scheduling of crude oil unloading with inventory management,” *Industrial & Engineering Chemistry Research*, vol. 35, no. 5, pp. 1630–1641, 1996.

[30] Z. Jia and M. Ierapetritou, “Mixed-integer linear programming model for gasoline blending and distribution scheduling,” *Industrial & Engineering Chemistry Research*, vol. 42, no. 4, pp. 825–835, 2003.

[31] S. Yadav and M. A. Shaik, “Short-term scheduling of refinery crude oil operations,” *Industrial & engineering chemistry research*, vol. 51, no. 27, pp. 9287–9299, 2012.

[32] E. Rubio-Castro, J. M. Ponce-Ortega, M. Serna-González, M. M. El-Halwagi, and V. Pham, “Global optimization in property-based interplant water integration,” *AIChE Journal*, vol. 59, no. 3, pp. 813–833, 2013.

[33] J. P. d. S. Catalão, H. M. I. Pousinho, and V. M. F. Mendes, “Hydro energy systems management in portugal: profit-based evaluation of a mixed-integer nonlinear approach,” *Energy*, vol. 36, no. 1, pp. 500–507, 2011.

[34] B. Galan and I. E. Grossmann, “Optimal design of distributed wastewater treatment networks,” *Industrial & engineering chemistry research*, vol. 37, no. 10, pp. 4036–4048, 1998.

[35] T. Achterberg, “SCIP: solving constraint integer programs,” *Mathematical Programming Computation*, vol. 1, no. 1, pp. 1–41, 2009.

[36] N. V. Sahinidis, “BARON: A general purpose global optimization software package,” *Journal of global optimization*, vol. 8, no. 2, pp. 201–205, 1996.

[37] R. Misener and C. A. Floudas, “ANTIGONE: algorithms for continuous/integer global optimization of nonlinear equations,” *Journal of Global Optimization*, vol. 59, no. 2, pp. 503–526, 2014.

[38] G. P. McCormick, “Computability of global solutions to factorable nonconvex programs: Part iconvex underestimating problems,” *Mathematical programming*, vol. 10, no. 1, pp. 147–175, 1976.

[39] R. Horst and H. Tuy, *Global optimization: Deterministic approaches*. Springer Science & Business Media, 2013.

[40] E. M. Smith and C. C. Pantelides, “Global optimisation of nonconvex MINLPs,” *Computers & Chemical Engineering*, vol. 21, pp. S791–S796, 1997.

[41] C. A. Floudas and P. M. Pardalos, *Recent advances in global optimization*. Princeton University Press, 2014.

[42] D. S. Wicaksono and I. A. Karimi, "Piecewise MILP under-and overestimators for global optimization of bilinear programs," *AIChE Journal*, vol. 54, no. 4, pp. 991–1008, 2008.

[43] R. Misener, J. P. Thompson, and C. A. Floudas, "APOGEE: Global optimization of standard, generalized, and extended pooling problems via linear and logarithmic partitioning schemes," *Computers & Chemical Engineering*, vol. 35, no. 5, pp. 876–892, 2011.

[44] J. P. Teles, P. M. Castro, and H. A. Matos, "Univariate parameterization for global optimization of mixed-integer polynomial problems," *European Journal of Operational Research*, vol. 229, no. 3, pp. 613–625, 2013.

[45] P. A. C. Castillo, P. M. Castro, and V. Mahalec, "Global optimization of MIQCPs with dynamic piecewise relaxations," *Journal of Global Optimization*, vol. 71, no. 4, pp. 691–716, 2018.

[46] H. Nagarajan, M. Lu, S. Wang, R. Bent, and K. Sundar, "An adaptive, multivariate partitioning algorithm for global optimization of nonconvex programs," *Journal of Global Optimization*, vol. 74, no. 4, pp. 639–675, 2019.

[47] G. Optimization, "Gurobi 8 performance benchmarks," 2019. `https://www.gurobi.com/pdfs/benchmarks.pdf`.

[48] K. Sundar, S. Sanjeevi, and H. Nagarajan, "Sequence of polyhedral relaxations for nonlinear univariate functions," *Optimization and Engineering*, pp. 1–18, 2021.

[49] P. M. Castro, "Normalized multiparametric disaggregation: an efficient relaxation for mixed-integer bilinear problems," *Journal of Global Optimization*, vol. 64, no. 4, pp. 765–784, 2016.

[50] S. Yıldız and J. P. Vielma, "Incremental and encoding formulations for mixed integer programming," *Operations Research Letters*, vol. 41, no. 6, pp. 654–658, 2013.

[51] CPLEX, IBM ILOG, "User's manual for CPLEX (international business machines corporation)," 2009.

[52] J. P. Vielma, "Mixed integer linear programming formulation techniques," *Siam Review*, vol. 57, no. 1, pp. 3–57, 2015.

[53] J. Huchette and J. P. Vielma, "Nonconvex piecewise linear functions: Advanced formulations and simple modeling tools," *Operations Research*, 2022.

[54] M. L. Bergamini, P. Aguirre, and I. Grossmann, "Logic-based outer approximation for globally optimal synthesis of process networks," *Computers & chemical engineering*, vol. 29, no. 9, pp. 1914–1933, 2005.

[55] I. P. Androulakis, C. D. Maranas, and C. A. Floudas, "αbb: A global optimization method for general constrained nonconvex problems," *Journal of Global Optimization*, vol. 7, no. 4, pp. 337–363, 1995.

[56] H. S. Ryoo and N. V. Sahinidis, "A branch-and-reduce approach to global optimization," *Journal of global optimization*, vol. 8, no. 2, pp. 107–138, 1996.

[57] R. Bellman, "On the theory of dynamic programming," *Proceedings of the National Academy of Sciences*, vol. 38, no. 8, pp. 716–719, 1952.

[58] A. M. Shkel and V. Lumelsky, "Classification of the dubins set," *Robotics and Autonomous Systems*, vol. 34, no. 4, pp. 179–202, 2001.

[59] P. Belotti, S. Cafieri, J. Lee, and L. Liberti, "On feasibility based bounds tightening," 2012.