Generative Multi-Robot Task and Motion Planning Over Long Horizons

by

Enrique Fernández González

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2018

Author . . . .
Department of Aeronautics and Astronautics
December 29th, 2017

Certified by . . . .
Brian C. Williams
Professor, MIT
Thesis Supervisor

Certified by . . . .
Leslie P. Kaelbling
Professor, MIT
Thesis Committee Member

Certified by . . . .
Russ Tedrake
Professor, MIT
Thesis Committee Member

Accepted by . . . .
Paulo C. Lozano
Chairman, Graduate Program Committee

Generative Multi-Robot Task and Motion Planning Over Long Horizons

by

Enrique Fernández González

Submitted to the Department of Aeronautics and Astronautics
on December 29th, 2017, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

The state of the art practice in robotics planning is to script behaviors manually, where each behavior is typically precomputed in advance. However, in order for robots to be able to act robustly and adapt to novel situations, they need to be able to plan sequences of behaviors and activities autonomously. Since the conditions and effects of these behaviors are tightly coupled through time, state and control variables, many problems require that the tasks of activity planning and trajectory optimization are considered together.

There are two key issues underlying effective hybrid activity and trajectory planning: the sufficiently accurate modeling of robot dynamics and the capability of planning over long horizons. Hybrid activity and trajectory planners that employ mixed integer programming within a discrete time formulation are able to accurately model complex dynamics for robot vehicles, but are often restricted to relatively short horizons. On the other hand, current hybrid activity planners that employ continuous time formulations can handle longer horizons but they only allow actions to have continuous effects with constant rate of change, and restrict the allowed state constraints to linear inequalities. This greatly limits the expressivity of the problems that these approaches can solve.

In this work we present *Scotty*, a planning system for hybrid activity and trajectory planning problems. Unlike other continuous time planners, Scotty can solve a broad class of expressive robotic planning problems by supporting convex quadratic constraints on state variables and control variables that are jointly constrained and that affect multiple state variables simultaneously. In order to efficiently generate practical plans for coordinated mobile robots over long horizons, our approach employs recent methods in convex optimization combined with methods for planning with relaxed planning graphs and heuristic forward search.

The contributions of this thesis are threefold. First, we introduce a convex, goal-directed scheduling and trajectory planning problem. To solve this problem, we present the ScottyConvexPath planner, which reformulates the problem as a Second Order Cone Program (SOCP). Our formulation allows us to efficiently compute robot trajectories with first order dynamics over long horizons. While straightforward formulations are not convex, we present a convex model that does not require state, control or time discretization. Second, we introduce the ScottyActivity planner, a state of the art hybrid activity and trajectory planner that interleaves heuristic forward search with delete relaxations and consistency checks using our convex model. Finally, we present ScottyPath, a qualitative state plan planner that computes control and obstacle-free state trajectories for robots in order to satisfy the temporally extended goals and constraints that ScottyActivity imposes. ScottyPath finds obstacle-free paths in which all robots are guaranteed to always remain within obstacle-free safe regions, which are computed in advance. We introduce several new robotic planning domains, that we use to evaluate the scalability of our planning system and compare the performance of our approach against other prior methods. Our results show that ScottyActivity performs similarly to other state of the art heuristic forward search activity planners, while solving much more expressive robotic planning problems. On the other hand, ScottyPath can generate obstacle-free paths where robots are contained in obstacle-free convex regions more than two orders of magnitude faster than alternative mixed-integer approaches.

Thesis Supervisor: Brian C. Williams
Title: Professor, MIT

Thesis Committee Member: Leslie P. Kaelbling
Title: Professor, MIT

Thesis Committee Member: Russ Tedrake
Title: Professor, MIT

Thesis Reader: Erez Karpas
Title: Assistant Professor, Technion

Thesis Reader: Richard Camilli
Title: Principal Investigator, WHOI

Acknowledgments

As cliché as it may sound, completing my PhD at MIT is a dream come true. MIT is a wonderful place, and I am eternally thankful for my time here. For over five years, I have had the fortune to work and share my time with incredible people that have supported me over the years. This thesis has been shaped in part by them.

First of all, I would like to thank my advisor, Brian Williams, for accepting me in the group and, hence, giving me the opportunity of a lifetime, even when I did not have any experience in the field. I would like to thank him for challenging me everyday and for his endless efforts in attempting to make me a better researcher. I would also like to thank my committee members and readers, Leslie Kaelbling, Russ Tedrake, Erez Karpas and Rich Camilli, for their constructive feedback and their guidance in this thesis. I would like to thank Leslie and Russ, whose classes I have had the fortune to take, for being an example on how to lead groundbreaking research groups with a contagious passion. I would like to thank Rich for his patience explaining over and over the real way in which underwater robotic missions take place. I could gladly spend hours listening to his expedition stories. Special thanks to Erez, for all his help throughout these years. His stay in our group made a big difference in my work. Erez guided me when I felt lost, taught me to write my first paper and was always available to talk, regardless of whether he was in Boston or Israel, or I was in Singapore or California. His part in this thesis cannot be overstated.

I would like to thank all the members of the MERS group: Ameya, Andreas, Andrew, Askhan, Brian, Ben, Christian, Claudio, Cyrus, Dan, David, Erez, Eric, James, Jingkai, Jonathan, Larry, Marlyse, Matt, Nick, Nikhil, Pedro, Peng, Sang, Sean, Simon, Shawn, Spencer, Steve, Szymon, Tiago, Yuening. Thank you for creating a great environment where everyone always helped each other. I could not have asked for better teammates and I will miss our daily interactions. Thank you Andreas, Tiago, Christian and Ashkan for your guidance and advice throughout the years. Special thanks to Steve, for being a great neighbor, with a contagious enthusiasm for creating cool stuff and who always kept our robots alive; Peng, for being the nicest, most generous teammate I have ever had; Simon, for always trying to keep the group social, and for being, for many years, the only other guy in the lab that enjoyed soccer; Pedro, for being annoyingly smart and somewhat funny, but always right; and James, for the great two years we spent as labmates and for flying all the way from South Africa to show up at my wedding in Spain.

I would also like to thank 'Obra Social La Caixa', for giving me the fellowship that brought me here, and for inspiring generations and generations of young Spaniards to dream high. I would also like to thank the SUTD-MIT Graduate Fellows Program, for their financial support and, especially, for the wonderful opportunity of spending a semester teaching Math in Singapore. My gratitude goes well to Boeing and Exxon, for their generous financial support throughout the years.

Over the last five and a half years I have had the fortune to meet incredible talented, humble and generous people that have made this an amazing experience. Thank you to my friends from the Spain@MIT club, for winning more soccer championships than I can remember, and for making Boston feel a home far away from home. Special thanks to Paula, Maite, Noel, Ferran, Juanito, Enrique, Alex, Joel, Hasier, Adriana, and Jordi. Thank you to my dear friend Julian, whose arrival in Boston provided me with a great support, and who is still every bit of an amazing roommate today, as he was in a tiny town in Denmark seven years ago.

I would also like to thank the MIT Sport Taekwondo Club, which has been an essential part of my life at MIT. The workouts, tournaments, defeats, victories and friendships constitute one of my fondest memories. Thank you to my co-instructors, co-captains, and teammates for teaching me so much, not only about taekwondo, but specially about recovering from failure, defeat and injury. I will miss you dearly. Specially, thank you Master Chuang, for making me an athlete, for giving me confidence, for turning me into an ECTC all-star and for trusting me to be a team captain and lead classes.

I would like to thank my family and friends at home and around the world for their love and support, no matter how far I have been or for how long. You know who you are. Thank you, specially, for making me feel like nothing has changed every time I am home. It means the world to me.

Special thanks, of course, to my parents, for their endless love, and whose sacrifices have provided me with the best opportunities. I thank them for their constant caring advice, and for their unwavering support, no matter what I have decided to do or how far away I have moved. This thesis is dedicated to them.

Most importantly, I would like to thank my wife, Anna. Thank you for being always there for me. Thank you for comforting and encouraging me when I felt I was unable to continue. Thank you for always believing in me more than I do. Thank you for leaving everything in Barcelona to come with me to start this adventure, and for supporting me every step in the way. I wouldn't have made it without you. I wish to dedicate this thesis to her as well.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Due to advances in mechanical design and control, the capabilities of robots have been improving at a dramatic rate over the last few years. However, most robots operating in the real world are not autonomous. For example, in the recent DARPA Robotics Challenge [55], participants demonstrated impressive humanoid robot behaviors, such as walking and climbing stairs, grabbing and using power tools to drill holes and even driving cars [35]. However, all of these behaviors were controlled remotely by human operators. Other robots are used routinely in scientific missions, such as the ones performed by Woods Hole Oceanographic Institution (WHOI) and others, in which autonomous underwater vehicles (AUVs) are sent to collect data of scientific interest determined by experts. AUVs have been used, among others, to track hydrocarbon plumes [18], explore ancient shipwrecks [9] or even ice-covered Arctic ocean regions [57]. Although these robots operate mostly on their own while the mission is underway, they often execute fixed scripts that are hand-written by experts in a tedious, time-consuming and error-prone way. Teleoperation or script-based methods do not scale in a cost efficient way and are not appropriate for situations with long communication delays (e.g. space) or where not all the information is known in advance. For this and other cases it is desirable to have autonomous robots capable of reasoning about their goals and the environment they operate in.

In order to plan missions for autonomous robot vehicles with state-dependent goals that are subject to temporal deadlines and coordination constraints, it becomes essential to model their dynamics with sufficient accuracy. Planners handling these missions need to consider, at a minimum, the allowed velocities that these vehicles can travel with, as well as be able to subject these robots to complex state constraints, such as being inside regions or maintaining certain distances with other robots. Planning for these missions involves reasoning over discrete and continuous conditions and effects, as well as the previously mentioned robot dynamics, coordination constraints and temporal deadlines. As a consequence, it becomes necessary to consider activity execution times, state and control variables jointly. The problem is, therefore, a hybrid activity and trajectory planning problem. Finally, the typical robotic missions that we consider often span many hours or days. Some of the activities in such missions are short, such as the activation or deactivation of sensor suites. Others execute over long durations, such as the traversal of long distances between regions of interest. Therefore, it is essential to handle short and long term activities efficiently over long horizons.

Over the last few years, the robotics community has had tremendous success with trajectory optimization and other sophisticated methods to control robots with many degrees of freedom and complicated dynamics. It is now common to see, not only in simulation, but even more impressively, in real hardware, demonstrations of robots walking in complex terrains, climbing stairs, running and even jumping [26, 56, 3, 77]. However, as impressive as these demonstrations are, the robotics community has not often considered the activity planning problem, and researchers have resorted to sequencing these complex behaviors computed using trajectory optimization by hand. These approaches have been, in general, restricted to limited horizons in which fixed time discretization works well. Therefore, these methods do not address the problem of planning missions of robot vehicles over long horizons in which activity planning is required.

Model-based generative planners have been applied successfully to space missions [95]. For example, the Remote Agent system was able to reason with temporal and resource constraints in order to autonomously diagnose and repair failures on-board the Deep Space 1 spacecraft [72]. Other planners have been used to plan science activities for rovers on Mars [5]. However, these planners have, in general, considered only discrete conditions and effects and relied on domain-specific ‘planning experts’ to solve their problems. These approaches are, therefore, not suitable to solve the joint activity and motion planning problems that we seek to address in this thesis.

In order to reason effectively with both the discrete and continuous behaviors that robots exhibit, Kongming [64, 63] was introduced as the first approach that merged activity planning and trajectory optimization to generate practical plans for real robots. By reasoning with both discrete and continuous effects depending on control variables, Kongming made important advances in merging both worlds, and demonstrated its practical usefulness in real underwater robot missions. Unfortunately, Kongming’s approach requires a fixed time discretization that hinders its ability to scale well to missions with long horizons in which short and long lived activities coexist.

On the other hand, heuristic forward search (HFS) approaches for activity planning have shown over the last two decades immense progress in the scale of the problems that they can solve. A large part of their success comes from the effectiveness of new heuristics, such as delete-relaxations [47] or, more recently, landmarks [81], together with the adequacy of greedy search. Some of these approaches have been extended to support continuous time-dependent effects. This is the case for the COLIN planner [23], that uses an approach based on heuristic forward search with delete relaxations to handle both temporal planning and continuous time-dependent effects. Like previous planners such as LPSAT [96] and LPGP [69] that interleaved linear programming with discrete search, COLIN relies on linear programs to test the consistency of partial states. Many of these planners do not discretize time and can, therefore, handle long horizons with short and long activities well. However, these approaches are not sufficiently expressive for solving robot hybrid activity and trajectory planning problems. These planners support only continuous time-dependent effects with constant rates of change and state variable constraints that are simple linear inequalities. While some robotic hybrid activity and trajectory planning problems can be modeled with this restriction, this requires defining many activities with multiple discretized values for the different desired rates of change of the continuous effects (such as the velocities of each robot). This does not scale well to the practical robotic problems that we aim to solve. Moreover, many typical robot planning problems, like the ones that we study in this thesis, cannot be modeled with that approach since the state constraints are only limited to simple linear inequalities.

Finally, over the last few years, the robotics community has shown a great interest in the combined Task and Motion Planning problem (TAMP) [90, 16, 41, 70]. Many of these approaches combine state of the art discrete activity planners with specialized motion planners for robotic applications. In order to do that, these approaches often discretize the robot states on demand as they need to, and commit early to these discretized states as soon as a feasible one is found. Most of the work in TAMP has focused on manipulation tasks, where planners need to handle complex and highly non-linear and non-convex constraints over many degrees of freedom. While these approaches have shown promising results, they are often limited to classical activity planning, with no temporal constraints, and in which dynamics are largely ignored. They often use discretized steps and are limited to short horizons, and therefore these approaches are generally not applicable to the robot vehicle mission planning problems that we target in this work.

This thesis delivers the *Scotty Planning System*. Scotty is a hybrid activity and trajectory planner that plans missions that involve the coordination of multiple robot vehicles over long horizons. By supporting convex, non-linear state constraints and control variables that often model controllable rates of change or robot velocities, Scotty can model a wide range of real robotic missions. By using a continuous time formulation, Scotty scales efficiently to missions with long durations, even in the presence of both short and long lived activities. Scotty exploits recent advances in convex optimization and avoids discretizing state or control variables, which in turn prevents early commitment to inadequate discretization choices. Scotty uses continuous convex optimization to choose continuous states, control variables and times, but leaves this choice open until the final plan is found, allowing a great amount of flexibility to the optimizer until the end, which others have shown to provide great advantages [92].

In order to solve the hybrid activity and trajectory planning problem, Scotty divides the problem in two subproblems that are each solved with a different module. First, the *ScottyActivity* module solves the hybrid activity and trajectory planning problem in the absence of obstacles. ScottyActivity uses heuristic forward search and a convex model that does not discretize time or state variables that scales well to long horizons. The solution that *ScottyActivity* finds is a *Qualitative State Plan* (QSP) that describes the constraints on continuous state and motion trajectories that each robot is subject to at different steps of the mission, as well as the activities and behaviors that need to be executed and when. The QSP returned by *ScottyActivity* is flexible, in that it does not enforce specific values of states or activity execution times, but it provides, instead, a collection of continuous time, state and motion constraints that a feasible plan needs to satisfy.

Second, *ScottyPath* takes the QSP that ScottyActivity generates and returns a plan consisting of the schedule of activities, and the control and state trajectories of each vehicle that are guaranteed to be obstacle free. Instead of considering the obstacles directly, ScottyPath solves the dual problem in which vehicles need to remain inside convex safe regions, which are computed in advance. This allows ScottyPath to use an approach based on informed search and convex optimization in order to compute multi-vehicle obstacle-free trajectories with coordination constraints over long horizons.

Both planners solve problems that require highly combinatorial discrete choices: the selection of the activities and their order, in the case of ScottyActivity; and the sequence of convex safe regions that each robot must traverse, in the case of ScottyPath. In both cases, we make the discrete choices by using heuristic forward search, and a subplanner that solves a relaxed problem in order to test the consistency and compute the heuristic value of candidate plans. By leveraging informed heuristics, our approach performs two orders of magnitude faster than mixed-integer approaches using branch and bound search. Both ScottyActivity and ScottyPath use the same subplanner, *ScottyConvexPath.* The problem that *ScottyConvexPath* solves is a relaxed problem in which the activities and their order are fixed and the environment is obstacle-free. Under those conditions, the combinatorial discrete choices are eliminated and the problem can be solved very efficiently using convex optimization. ScottyConvexPath is a core contribution of this thesis and the component that allows us to plan over long horizons. It is possible thanks to three insights. First, due to recent advances in optimization, a restricted form of quadratically constrained programs, called Second Order Cone Programs (SOCPs), can be solved efficiently for real world problems. Second, nearly all of the requirements of the robot vehicle missions that Scotty targets can be encoded with cone constraints, with the exception of a non-convex term resulting from the product of control variables and time. Third, an encoding trick allows us to eliminate this non-convex term, resulting in a SOCP encoding that is very fast to solve and that our planners repeatedly use to test the consistency and compute the heuristic value of partial plans. Our SOCP encoding allows us to impose upper bound constraints on the norm of vectors of control variables (e.g. $v_x^2 + v_y^2 \leq v_{max}^2$), enforce convex quadratic state constraints (such as being inside ellipsoidal regions or ensuring a maximum distance between objects) and use the same control variables in as many simultaneous effects as needed. We do this without resorting to time, state or control variable discretization, which allows us to efficiently scale to long horizons.

Throughout the rest of this chapter, we describe the goals that this thesis addresses, we present a summary of our contributions and we describe the organization of this thesis.

1.1 Thesis Goals

In this thesis, we aim to solve activity and motion planning problems for multiple robotic vehicles over long horizons. Such problems commonly arise during the planning of scientific missions, such as the ones involving multiple coordinated underwater robots, which Woods Hole Oceanographic Institute routinely executes.

This thesis aims to solve planning problems with the following requirements.

1. **Sufficiently accurate modeling of mission constraints and robot dy-**

namics. While planning for missions over long horizons, it is often possible to neglect low level detailed dynamics. However, in order to ensure that temporal deadlines and coordination constraints can be satisfied, it is essential to consider, at a minimum, the velocity limits that each robot is subject to. Furthermore, in order to express the operating requirements of the robots, it is necessary to consider sufficiently rich state constraints, which should allow, for example, to require a robot to remain inside a region or to enforce a maximum distance constraint.

2. **Multiple robot vehicles subject to coordination constraints**. Robotic oceanographic science missions involve, at a minimum, an underwater or surface robot and a vessel. Moreover, due to the high cost of operating a vessel, some missions benefit from having multiple underwater robots achieve different mission goals simultaneously. There are often coordination constraints between the locations of the robots and the vessel. For example, it is often desirable to ensure that the robots are always within a maximum distance that allows for ultrasound communications between the vehicles.

3. **Plan over long horizons, with missions involving both short and long lived activities**. Typical oceanographic missions range from 10 and 20 hours to weeks. Certain activities, such as transits, involve long periods of time. Other activities, such as the engagement or disengagement of sensor suites, can be executed in short periods of time. It is important that our planner is able to reason over all mission activities jointly, for the duration of the mission.

In Chapter 3, we provide an example scenario that illustrates these requirements, and that we use to describe our problem statement and approach in later chapters of this thesis.

1.2 Thesis Contributions

In order to address the robotic planning problems with the requirements that we describe in Section 1.1, this thesis makes the following contributions.

1. **An architecture for hybrid activity and trajectory planning for mobile coordinated robots.**

   We introduce a planning architecture in Chapter 3 that decomposes the planning and execution of missions involving multiple coordinated robots into multiple subproblems that are solved by specialized planners.

2. **A goal-directed scheduler and trajectory planner over long horizons**

   We present *ScottyConvexPath*, a planner for relaxed problems in which the activities and their order are fixed and the environment is obstacle-free. ScottyConvexPath reformulates the problem as a second order cone program (SOCP) that is efficiently solved in order to compute control and state trajectories for multiple coordinated robots along with the execution times of the activities. Since our encoding does not require discretization of either time, state or control, it scales to arbitrarily long horizons. ScottyConvexPath computes optimal trajectories involving first order dynamics and convex quadratic state constraints. ScottyActivity and ScottyPath use ScottyConvexPath to efficiently test the consistency of partial plans and compute their heuristic value.

3. **A hybrid activity and trajectory planner.**

   We present *ScottyActivity*, a heuristic forward search activity and trajectory planner that computes plans for coordinated mobile robots over long horizons. Our novel planner exploits advances in planning with delete relaxations in order to achieve scalable performance, while enabling planning formulations that are more expressive than prior art thanks to the presence of control variables and convex quadratic state constraints.

4. **A goal-directed path planner over convex safe regions.**

Our final contribution is *ScottyPath*, a path planner for goal-directed problems given as qualitative state plans (QSPs). ScottyPath finds control and state trajectories for mobile coordinated robots. ScottyPath guarantees that the robot trajectories are collision-free by ensuring that each robot always remains within a convex safe region. ScottyPath uses a novel approach that interleaves informed search with calls to ScottyConvexPath, that is used to simultaneously compute the consistency, cost and heuristic of candidate partial plans. Our approach simultaneously assigns sequences of safe regions to each robot and computes control and state trajectories that satisfy temporally extended goals. While ScottyPath is designed to operate in conjunction with ScottyActivity, it has also be used on its own to solve manually defined QSPs that describe missions oceanographic with multiple underwater robots.

1.3 Organization of This Thesis

The Scotty Planning system consists of three modules. This is reflected in the organization of this thesis. In Chapter 2, we describe the prior work in activity and trajectory planning related to this thesis. In Chapter 3, we describe the architecture of the Scotty Planning System, and we provide an example scenario based on a real underwater robotic science mission that we use to inspire our thesis and to illustrate the concepts throughout the rest of the chapters. The problem statement addressed by this thesis is described in Chapter 4.

Part I describes the first of Scotty's modules: *ScottyConvexPath*. This relaxed planner is the core component used in the remaining two other Scotty modules: ScottyActivity and ScottyPath. Before describing the convex model that ScottyConvexPath uses, Chapter 5 details the restriction of piecewise constant control trajectories that we require in order to efficiently satisfy conditions over arbitrarily long horizons. The convex optimization model is then described in detail in Chapter 6.

Part II describes *ScottyActivity*: the hybrid activity and trajectory planning module in the Scotty system. ScottyActivity selects robot activities and behaviors in order to generate a qualitative state plan that satisfies the problem constraints. ScottyActivity solves a relaxed problem in which obstacles are not considered. Chapter 7 describes the extension to PDDL that we define in order to specify PDDL-S problems, the input problems to the ScottyActivity planner. Our heuristic forward search planning approach is detailed in Chapter 8. Finally, Chapter 9 presents the experimental results of the ScottyActivity planner.

Part III covers the last module of the Scotty Planning System: *ScottyPath*. ScottyPath takes the qualitative state plan generated by ScottyActivity and produces an activity schedule and control and collision-free state trajectories for each vehicle. For pedagogical purposes, Chapter 10 describes a simpler, independent geometric path planner that explains the core concepts used by ScottyPath. ScottyPath's approach is described in Chapter 11. Finally, Chapter 12 shows the experimental results for the ScottyPath planner.

This thesis concludes with Chapter 13, which presents a summary of contributions and avenues for future work.

Chapter 2

Related Work

In this chapter we discuss relevant literature in hybrid planning that inspires this thesis and that we leverage in our approach.

2.1 Temporal and Hybrid Discrete-Continuous Planning

Since the creation of the Planning Domain Definition Language (PDDL) [71] and the advent of the International Planning Competitions, the planning community has made immense progress in the scale of the *classical planning* problems that can be solved. A significant part of this progress comes from the success of heuristic forward search and very effective heuristics, such as delete-relaxations [47] or, more recently, landmarks [81]. Very recently, researchers have developed incomplete width-based search techniques that can run in polynomial time and present an empirical performance that is better than prior state of the art in *classical planning* problems [67, 66].

Classical planning problems only involve discrete conditions and effects. Very early, researchers extended classical planning problems with the ability to reason with durative actions where each action could have a different duration. For example, the *TGP* planner [89] represents mutual exclusions between actions of different duration in the planning graph and *Sapa* [32] uses a heuristic forward chaining approach in order to handle durative actions and deadline goals. However, these approaches reduce durative actions to *compressed action* representations and can not solve problems with 'required concurrency' [25]. The PDDL2.1 standard [39] extends classical planning with new semantics that allow temporal durative activities, metric variables and, even, continuous effects. One of the first planners capable of planning with the full temporal semantics of PDDL2.1, including required concurrency and Timed Initial Literals [46], was *CRIKEY3* [24]. This planner uses an approach that interleaves heuristic forward search with the scheduling of action starts and ends, which is done by solving a Simple Temporal Problem [27]. Later, the Temporal Fast Downward planner [34] demonstrated increased performance in temporal planning problems by using a context-enhanced additive heuristic. Apart from the temporal extension to classical planning, researchers also extended the classical formulation to support metric variables, which have commonly been used to model resources. For example, *Metric-FF* [45] extends the delete-relaxation heuristic by storing, in each layer of the planning graph, the minimum and maximum values that each variable could reach. We discuss in Section 8.3 that the ScottyActivity heuristic uses a similar approach that is inspired by this concept.

Except for time, all the planners discussed so far are only able to reason with discrete conditions and effects. However, may problems, such as the robotic missions that we study in this thesis, require reasoning with continuous change. One of the first planners capable of planning with continuous change was *ZENO*, a least commitment planner that combines first-order logic, constraint satisfaction and first order continuous effects. ZENO was impressively advanced for its day, but its approach only scales to problems with a handful of actions.

In order to support planning for robotic applications, *Kongming* [64] was one of the first planners able to reason with hybrid activities, which consist of discrete and continuous conditions and effects. Kongming's continuous effects are very expressive, as they can model k-th order discrete-time dynamics. Kongming uses an approach that mixes an analog to the Planning Graph from *Graphplan* [11] and mixed integer optimization. Due to the features it supports, Kongming is perhaps the planner that is closest to ScottyActivity in the type of problems that both can solve. Kongming is the first activity planner that supports control variables (such as controllable velocities in moving robots) affecting continuous effects. One of the main innovations introduced by Kongming is its representation of continuous effects with flow tubes [48, 49], that are abstractions of the infinite number of trajectories that a continuous action can produce. Another key innovation introduced by Kongming is the Hybrid Flow Graph, the continuous analog to Graphplan's Planning Graph. Hybrid actions connect initial state regions to goal regions after some fixed duration using the flow tubes generated from the system dynamics. Kongming expands the Hybrid Planning Graph with alternating action and fact layers until the goal conditions are non-mutex in the last fact layer. The problem is then encoded as a mixed logic linear program (MLLP) that contains both the continuous constraints representing the dynamics and the logic constraints on binary variables that describe the discrete conditions and effects and the mutexes in the planning graph. This approach is a continuous analog to *Blackbox* [54], which combines the advantages from both Graphplan and *SATPLAN* [53] by encoding the planning graph as a SAT problem. Kongming alternates between trying to solve the MLLP and adding additional layers to the graph until the MLLP solver returns a solution. Later, the Kongming planner was extended to support temporally-extended goals by reformulating them into durative actions with effects that add specific predicates that need to hold at the end of the plan. This extension also added the capability of supporting actions with flexible durations [63, 62].

Kongming is an innovative planner that can solve expressive hybrid planning problems. However, it suffers from performance degradation issues in medium to large problems due to the fixed time-step discretization that its graph layers are subject to. In problems in which the planning horizon is moderately large and where short and long lived activities coexist, this involves creating many layers. As the number of layers increases, identifying mutex relations becomes exponentially more complicated. This also slows down significantly the MLLP solver, as each additional layer adds significantly more additional variables and constraints. Kongming inspires Scotty in its representation of continuous effects that depend on continuous control variables.

Instead of resorting to time discretization, the *COLIN* planner [21, 23] extended CRIKEY3's continuous-time, heuristic forward search approach to support problems with linear time-varying processes. COLIN solves temporal planning problems with continuous effects as defined in PDDL 2.1. Continuous effects are limited to constant rates of change, that are specified by a fixed value. In order to solve the mixed discrete-continuous planning problem, COLIN tests the consistency of partial plans by solving a linear program. This approach was first introduced by other planners such as LPSAT [96] and LPGP [69]. COLIN's heuristic is based on the Temporal Relaxed Planning Graph and delete relaxations. Although COLIN is an efficient and capable planner, it is not expressive enough to solve the kind of robotic problems that are the focus of this thesis. This is the case since the continuous effects that COLIN supports are limited to fixed rates of change, according to the following equation:

$$x(t_{end}) = x(t_{start}) + \text{rate-of-change} \cdot (t_{end} - t_{start})$$

COLIN's formulation cannot represent continuous controllable rates of change (control variables), that are required to model, for example, the controllable velocities of moving vehicles. Although an arbitrary number of actions can be defined having each a different discretized rate of change, we show in Chapter 9 that this approach does not scale well.

COLIN was later extended by *POPF* [22], that improves its performance by using a partial order representation of the underlying state. A further extension, *OPTIC* [6], was introduced to support preferences, a capability added in PDDL3 [42]. Although these planners lack the ability to represent robot behaviors naturally (due to, among others, the absence of control variables), they have been used in multiple robotic applications due to their robustness and scalability. For example, POPF was used to find automated inspection plans of underwater installations [19] and OPTIC was used to plan surveillance missions for low-cost quadcopters [7]. However, in these cases the planners did not explicitly consider the continuous motions of the robot, but instead chose a mission path by selecting discrete waypoints that were either previously generated using random sampling motion planning techniques, or manually specified. Other researchers have approached the problem of controlling hybrid systems with linear time-invariant dynamics by discretizing the control inputs and presolving the solutions to the linear dynamic equations for a fixed time step [68]. Solutions for each discretized control input are then encoded as actions with numeric effects that a state of the art planner, such as Temporal Fast Downward [34], uses to find a state and control trajectory. However this approach is not likely to scale well due to the fixed discretization required in time and control inputs.

In some situations, it is necessary to consider explicitly the robot motions during planning to ensure, for example, that temporal and spatial constraints are satisfied at all times during the mission. This is the reason why we developed the Scotty1 planner [36, 37], the previous version of the ScottyActivity planner presented in this thesis. Scotty1 extends the expressivity of previous heuristic forward search planning approaches by adding support for continuous control variables, that are essential to model robotic domains. Scotty1 combines the advantages of Kongming and COLIN. In particular, continuous effects support controllable rates of change, and are modeled with flow tubes. Scotty1's approach is inspired by COLIN, and consists of heuristic forward search combined with linear programs for testing the consistency of partial plans. The *cqScotty planner* [38], which we call *ScottyActivity* in this thesis, improves Scotty1 by supporting convex quadratic constraints and by allowing control variables to appear in multiple continuous effects simultaneously.

Other planners have recently explored planning with control parameters. For example, POPCORN [85] formalized the notion of continuous control parameters as an additional element of actions that are chosen by the planner. Contrary to Scotty, POPCORN's control parameters can only be used in discrete numeric effects and not as rates of change in continuous effects. Other recent approaches have also considered continuous control parameters, but are limited to discrete time and change [76].

Most of the planning approaches discussed previously are limited to linear continuous effects. However, over the last few years the planning community has made attempts to use the previous planning formalisms in non-linear settings. One popular approach consists in interleaving temporal planning with an external domain-specific solver capable of reasoning with complex non-linear change. This is often known as planning with 'semantic attachments' [33]. This approach has been used successfully to solve power balancing problems in an electricity network [78]. Another approach extends COLIN to handle a limited form of non-linear continuous monotonic effects using an iterative convergence method that repeatedly solves linear programs [4]. However, the assumptions required by this approach are not compatible with typical robot behaviors. While most semantic attachment techniques treat the external solver as a black box, a recent approach uses approximations of external complex numeric calculations in order to compute informative heuristic values that guide the search more efficiently than prior methods [8].

In order to significantly increase the capabilities of PDDL, PDDL+ was introduced in 2006 [40]. PDDL+ enhances the expressivity of PDDL by introducing processes, events and must-happen semantics. While non-linear change can also be represented with PDDL2.1, the rich semantics of PDDL+ processes and events, that make it easier to specify must-happen physical behaviors, has encouraged researchers to support non-linear effects in most PDDL+ planners. One of the first PDDL+ planners was *TM-LPSAT* [88], a planner that compiles PDDL+ problems into propositional atoms and linear constraints over numeric variables that are then solved with LPSAT. Another PDDL+ planner, *UPMurphi* [31], uses an uninformed discretize and validate approach. The *DiNo* planner [79] recently extended this approach with a novel heuristic that vastly improves UPMurphi's performance. Lately, the planning community has also explored model-checking [13, 12] and SAT Modulo Theories (SMT) [15, 20] based techniques to solve PDDL+ planning problems with promising results. Another recent approach generalized interval-based relaxations [1, 2] to solve PDDL+ problems with non-linear processes [86]. Most PDDL+ planners are, in general, more expressive than our planner in several ways, like their support of processes, events, must-happen semantics and continuous change that is non-linear in time. However, their semantics do not represent robot dynamics accurately, since they do not support control variables and are unable to model, for example, the motivating scenario that we describe in Section 3.1. Several of these planners also suffer from scalability issues since they require time and state discretization.

2.2 Optimization-based Approaches for Robotics Planning

Over the last few years, advances in trajectory optimization due to better performing solvers and more efficient encoding techniques have led to impressive results in the control of underactuated robots. While we do not intend to provide an extensive review of the vast field of trajectory optimization, we limit ourselves to highlight a few examples of optimization-based planning approaches for robotics applications and their relation with this thesis.

Since the advent of sampling-based algorithms, the most widely used approach for solving manipulation motion planning problems has been the rapidly-exploring random tree (*RRT*) algorithm [59] or one of its many variants, such as the asymptotically optimal *RRT** [52]. However, over the last few years, promising alternative approaches to motion planning for manipulation problems based on optimization methods have emerged. For instance, the *CHOMP* [80] planner uses covariant gradient descent to refine continuous paths. On the other hand, *TrajOpt* [87], which uses sequential convex optimization, has demonstrated impressive results. Part of the success of these planners is due to a clever encoding of collision constraints, that uses computer graphics techniques to compute the minimum distance to obstacles, and the use of penalty collision coefficients that are iteratively increased as needed.

Trajectory optimization has also been used extensively to control robots with complex dynamics in real environments. For example, the MIT team that participated in the DARPA Robotics Challenge [55] made extensive use of trajectory optimization for solving most of the competition tasks [35]. For example, they were able to demonstrate whole-body motion planning with centroidal dynamics and full kinematics using trajectory optimization [26] with similar collision constraints as the ones used by CHOMP and TrajOpt.

Other optimization techniques, such as sum of squares programming (SOS), have been applied to planning collision-free maneuvers for quadcopters in cluttered environments [28, 58]. This was achieved by exploiting the differential flatness of quadcopters and using a mixed-integer formulation that forced the robots to remain in obstacle-free convex regions that were computed in advance with the *IRIS* algorithm [30]. Similar approaches were used to plan the footstep placements of MIT's Atlas robot and were demonstrated during the DRC competition [29]. The ScottyPath planner, presented in Part III of this thesis, also finds obstacle-free trajectories in which robots are forced to remain in obstacle-free safe regions computed, in advance, with the IRIS algorithm. However, our approach relies on informed search and convex optimization, as opposed to integer programming, which allows us to scale to significantly larger problems.

Optimization-based techniques have also been used extensively to control robots using Model Predictive Control (MPC). These techniques have been extended to reason with risk and uncertainty. For example, the PSulu planner [74, 10, 73, 75] uses linear programming and iterative risk allocation (IRA) to generate chance-constrained plans that provide guarantees about the safety of the robot.

This thesis leverages advances in trajectory optimization, and is inspired by all these approaches that have successfully applied optimization techniques to the planning and control of mobile robots.

2.3 Combined Task and Motion Planning Approaches (TAMP)

Over the last few years, the robotics community has recently expressed a significant interest in the combined task and motion planning problem, and many interesting approaches have emerged. One of the most interesting ones integrates off-the-shelf task and motion planners by using a novel representational symbolic abstraction

[90]. The architecture of this planner alternates between solving discrete symbolic planning problems with abstracted poses, verifying and refining those plans using a motion planner and generating additional discrete poses when needed. This approach is innovative in that whenever the motion planner is unable to find collision free paths between poses, objects are removed one by one in order to discover which one is responsible for the plan failure. This information is then passed to the symbolic planner, which can decide, for example, that an object in the way should first be moved to a different location. A related approach [16] interweaves a symbolic planner (MetricFF) with a roadmap motion planner. In this case the symbolic planning formulation is extended to include geometric constraints and a roadmap for each movable element is maintained and extended as necessary.

On the other hand, the FFRob planner [41] approaches this problem by extending the heuristic ideas of the symbolic FF planner to motion planning by using a semantic attachments strategy. A search state, for FFRob, is composed of traditional literals and domain-dependent literals, such as reachability conditions, whose boolean value is determined by a test function that is evaluated lazily on demand. As an example, FFRob maintains a conditional reachability graph that represents the connectivity of configurations that are sampled during search.

Another interesting approach by *Lozano-Perez et al* frames this problem as a geometric constraint-satisfaction problem [70]. In this case, the search aims to find a sequence of activities forming a plan skeleton in which the specific robot and object poses are not bound until the CSP is solved by an off-the-shelf constraint satisfaction solver. However, the domains of the unbound variables need to be arbitrarily discretized in advance. Most of the planners discussed above approach the problem by bringing the continuous world of motion planning into the discrete symbolic planning formulation. This is often done by discretizing the continuous space either in advance or lazily during runtime as needed. The main advantage of this idea is that it allows the use of slightly modified off-the-shelf symbolic and motion planners. An important disadvantage, though, is the need to perform a discretization whose size needs to be chosen in advance.

Alternative approaches have attempted to follow the opposite direction by bringing the discrete symbolic world into the continuous space. These approaches use symbolic planning to generate sequences of actions in which the values of the continuous variables are not bound. The values of the continuous variables are then found by solving a large optimization problem. The main advantage of these methods is that no arbitrary discretization is required and that the solver can choose the best values for the continuous variables at the end in order to optimize for some criteria. This approach would have seemed intractable a few years ago, but impressive advances in trajectory optimization and non-linear solvers have made this a real possibility. For instance, *Toussaint* describes in [92] a logic-geometric approach that he demonstrates in a manipulation problem in which a robot picks and places cylinders and plates from a table in order to assemble the highest possible stable tower. Abstract plans defined by action sequences are generated using a relatively simple symbolic planning approach. The parameters of the solution abstract plan are found by solving a large optimization problem at the end. The solution to this optimization problem defines the best final and intermediate positions of all the objects in order to assemble the highest tower. This approach was recently extended with a more complex search method that combines ideas from branch and bound and Monte Carlo tree search (MCTS). Another related approach uses sequential quadratic programming to jointly optimize parameters in an abstracted task plan in which the actions are simple STRIPS operators [44]. Both planners presented in this thesis, ScottyActivity and ScottyPath, use a similar approach in which a sequence of activities defines a plan skeleton, whose parameters are found by solving a large optimization problem. In the same spirit, we do not reuse the solutions to intermediate optimization problems, which prevents our planners from committing to inadequate intermediate values and provides the solver with the maximum possible flexibility until the end.

Most of the TAMP approaches discussed so far have dealt almost exclusively with manipulation problems, which present on itself very complicated domain specific challenges. These planners often neglect temporal constraints or robot dynamics as these are not necessary to solve common manipulation problems.

On the other hand, the Scotty planner presented in this thesis considers problems where the robot behaviors are subject to dynamics and where temporal and state constraints are tightly coupled. Scotty solves a hybrid planning problem that requires robot behaviors, tasks and control plans to be computed jointly. We discuss this *hybrid activity and motion planning problem* in Chapter 3, where we provide an example scenario that we use throughout the rest of this thesis.

Chapter 3

The Scotty System: An Architecture for Hybrid Activity and Trajectory Planning

Common robotic applications require reasoning with continuous and discrete robot behaviors. These behaviors specify, for example, how robots move according to their dynamics and are often subject to state constraints. Mission goals often involve a combination of temporal and continuous and discrete state constraints in order to specify, for example, that a robot needs to visit a certain location before the deadline, or that a sample has to be acquired by the end of the mission. In order to satisfy the mission, a hybrid plan needs to specify not only what robot behaviors have to be executed and when, but also how to execute them by providing a control plan. Since the robot behaviors and mission specifications are tightly coupled by temporal and state constraints, the selection of the robot behaviors, their schedule and their control plans and trajectories need to be determined jointly. For example, a ship may need to meet an autonomous underwater vehicle (AUV) to pick it up. The location where the pickup takes place may not matter as long as it happens within the temporal deadlines and before the AUV runs out of battery. By jointly considering the mission constraints and the the dynamics and actuation constraints of the vehicles, we can determine the appropriate location where the pickup should take place and the

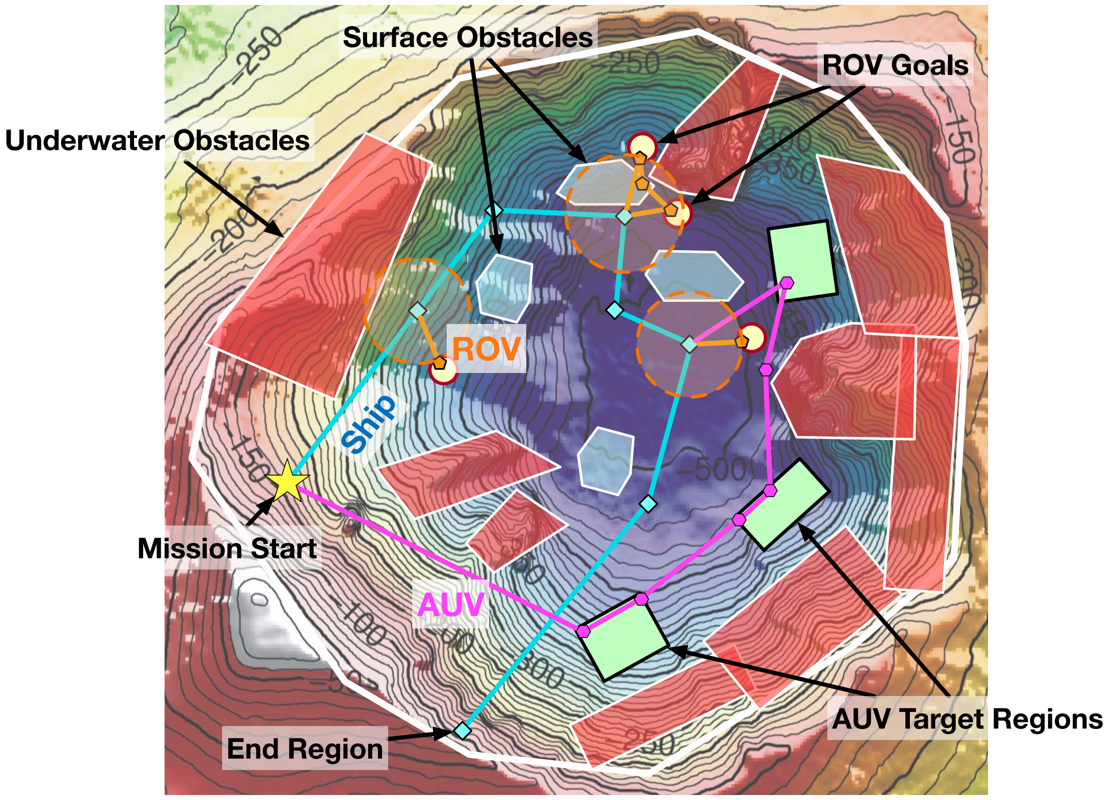


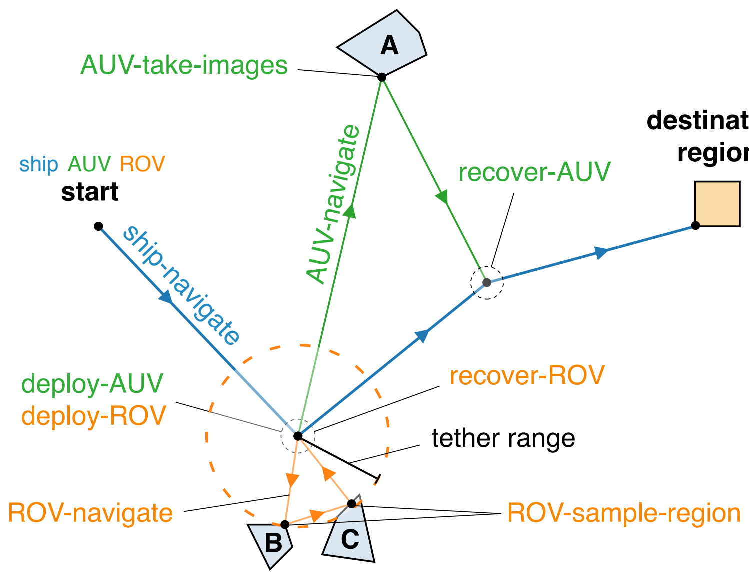
Figure 3-1: Robotic scientific exploration mission to the Kolumbo caldera.

trajectories of the vehicles in order to minimize the distance traveled by the ship. The previous is an example of a *hybrid activity and trajectory planning problem*, in which the behaviors/activities that need to be selected, their schedule and their associated control plans are selected jointly.

This thesis presents Scotty, an architecture for solving this type of *hybrid activity and trajectory planning problems*. In this chapter we start by describing, in Section 3.1, a motivating scenario based on a scientific ocean exploration mission that will take place in 2019. We then describe the Scotty Planning System along with its main components in Section 3.2.

3.1 Example Motivating Scenario

Our example scenario is modeled after a real underwater science mission to the Kolumbo volcano off the coast of Santorini (Greece) that will take place in 2019.



(a)

Vehicle motions

$$\dot{x}_i(t) = v_{x i}$$

$$\dot{y}_i(t) = v_{y_i}$$

AUV Battery decrease

$$\dot{b}_{AUV}(t) = -k \cdot \|\mathbf{v}_{AUV}\|$$

Max velocity constraints

$$v_{x_i}^2 + v_{y_i}^2 \leq v_{max_i}^2$$

ROV tether constraint

$$\|\mathbf{x}_{ROV} - \mathbf{x}_{Ship}\|^2 \leq R_{tether}^2$$

Recover constraints

$$\|\mathbf{x}_{AUV} - \mathbf{x}_{Ship}\|^2 \leq d_{\text{recover}}^2$$

Region constraints

$$\mathbf{x}_j \in R_i$$

(b)

Figure 3-2: Our motivation scenario exhibits interesting constraints such as the maximum distance constraint between the ship and the ROV (b). Our planner is able to select the best position for deploying and recovering the AUV and the ROV that satisfies all the constraints without requiring discretization of either time or state (a).

The Kolumbo volcano is an active submarine volcano that is geologically and biologically very interesting due to the hydrothermal vents present in its caldera. Moreover, subsea CO_2 pools not known to exist before were recently discovered nearby [17]. In 2019, Woods Hole Oceanographic Institute (WHOI) will lead an expedition to the Kolumbo caldera in which autonomous planning algorithms developed at the MIT MERS lab will be used to coordinate and plan the activities and trajectories of multiple vehicles. This mission is funded, in part, by the NASA PSTAR program as an analog mission to explore issues such as limited communications and harsh environments that future robotic missions to Europa and other moons in the Solar System will experience. In particular, in the third stage of this mission Scotty and other planners will be used to coordinate the trajectories and scientific activities of an autonomous underwater vehicle (AUV), a ship and a Remotely Operated Vehicle (ROV) tethered to the ship. We model our example scenario after this stage of the Kolumbo mission (Figure 3-1).

In our example scenario, the ship is initially transporting both the AUV and the

ROV to the science site. The AUV needs to take images at multiple regions, shown in green, while the ROV needs to take samples in the regions represented by the yellow circles. All three vehicles need to reach the destination region at the end, can navigate on their own and have their own v_x, v_y velocities. The velocities can be freely chosen, but their norms are upper-bound constrained ($v_{x_i}^2 + v_{y_i}^2 \leq v_{max_i}^2 \quad \forall i \in \{\text{ship}, \text{ROV}, \text{AUV}\}$). Whenever the ROV is deployed, the ship needs to remain still at the deployment location until the ROV is recovered again. Moreover, the ROV is tethered to the ship, and therefore it can only move within a circle centered at the ship with radius the tether length ($(x_R - x_S)^2 + (y_R - y_S)^2 \leq R_{tether}^2$). Both the AUV and the ROV can be picked up when at most 2 meters away from the ship. The AUV can navigate on its own once deployed, but it has a finite battery that limits how long it can travel on its own ($\dot{b}_{\text{AUV}} = -k \cdot \|\mathbf{v}_{\text{AUV}}\|$). The scenario presents surface obstacles that the ship needs to avoid. Similarly, there are underwater obstacles that represent regions that the ROV and the AUV are not allowed to cross.

For clarity purposes, Figure 3-2 shows a valid plan for a simpler version of the motivation problem, in which the ROV needs to take samples in two regions and the AUV needs to take images in one region. This plan presents several interesting characteristics. First, by stationing the ship and deploying the ROV at the appropriate location, both sampling regions can be visited without violating the tether range constraint, which saves time and fuel. Second, the AUV does not have a sufficiently large battery to reach the destination region on its own. Therefore, the ship needs to meet the AUV at a non fixed nor discretized intermediate location, pick it up and transport it to the destination region. Finally, the battery decrease function is also interesting, as it depends on the norm of the velocity of the AUV. In this work we show how Scotty can solve this problem by integrating heuristic forward search and convex optimization.

The example scenario described in this section showcases the needs of typical robotic missions that the Scotty Planning System has been designed to solve. First, we need a way to express the requirements of the robot behaviors that allow them to move according to their controllable velocities, or the fact that the AUV battery

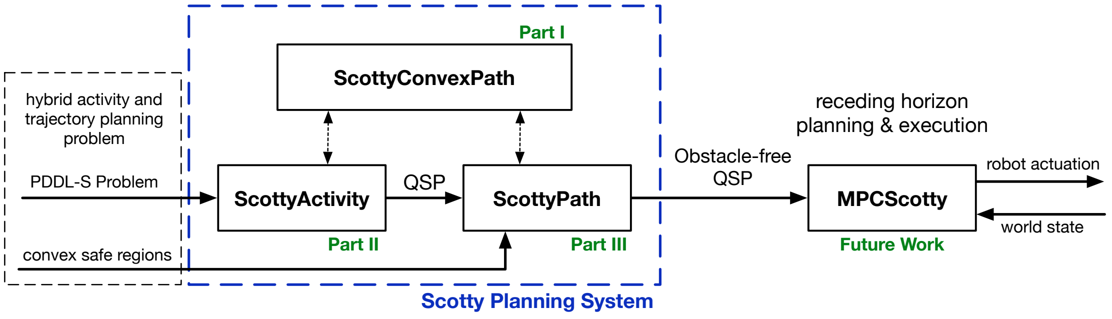


Figure 3-3: Scotty Architecture

gets drained according to the velocity that the AUV moves at. Moreover we need to model the state constraints that have to be satisfied at certain points in the mission: e.g. the ROV always satisfying the tether range and the AUV staying within region A while the images are taken, to name a few. Third, we need to be able to specify an objective to minimize, such the total duration of the mission or the distance traveled by the ship. Finally, each vehicle may need to avoid obstacles in the environment.

3.2 The Scotty Planning System

This thesis presents Scotty, a hybrid activity and trajectory planning architecture designed to plan for missions like the one described in Section 3.1. Such missions often involve multiple coordinated robots over long horizons. In order to efficiently plan long missions in which both short-lived and long-lived activities coexist, Scotty uses a formulation that is continuous in time, control and state. The continuous time requirement limits Scotty to reason with simplified first order dynamics in which the velocities of the robots are considered controllable within their lower and upper bounds. However, these dynamics are well suited for planning missions that span long durations, such as the one presented earlier.

The *hybrid activity and trajectory planning* problem that Scotty solves is hard due to its highly combinatorial nature. In particular, Scotty needs to make discrete combinatorial choices to select the activities and their order, and to avoid obstacles. We solve these two problems separately in order to leverage specialized heuristics, that allow us to solve the problem more efficiently. First, the activity planning problem is solved in the absence of obstacles. Then, the plan is refined in order to avoid the obstacles in the environment. Our architecture is shown in Figure 3-3.

The first module in our architecture is *ScottyActivity* [36, 37, 38], a hybrid generative activity and trajectory planner that solves the problem in the absence of obstacles. The solution returned by ScottyActivity forms a qualitative state plan that describes the activities or behaviors that each robot uses to achieve the goals and the order in which these are engaged and disengaged. ScottyActivity is described in Part II of this thesis.

The next module is *ScottyPath*, a planner for the qualitative state plans (QSPs) that ScottyActivity founds. Such QSPs do not consider obstacles but describe, qualitatively, all the constraints that need to be considered and in which order, including the behaviors that are active at different points in the mission. ScottyPath takes the QSPs generated by ScottyActivity and a set of convex safe regions for each vehicle and returns an obstacle-free plan in which all vehicles are guaranteed to be, at all times, inside one of such regions. ScottyPath is described in Part III of this thesis.

The last step in our architecture involves the execution of such plans. While the simple first-order dynamics model using during the first two stages is sufficient for planning long-term missions, a more accurate model is needed during execution. The *MPCScotty* planner is designed for such purpose. MPCScotty uses a receding horizon approach that jointly combines more accurate dynamics and constraints for a limited detailed horizon, and the same simpler constraints for the rest of the plan. *MPCScotty*, is being developed at the MIT MERS lab, and only a brief discussion of its approach is described in this thesis.

All the planners in the Scotty Architecture use, as their core component, the *ScottyConvexPath* planner. ScottyConvexPath solves relaxed problems in which the activities and their order are fixed and the environment is obstacle-free. Under those conditions, the problem does not present discrete combinatorial choices, and is solved efficiently using convex optimization. The convex optimization model that Scotty-ConvexPath uses describes the robots first order dynamics and does not require discretization of time, state or control. Our model, described in Part I of this thesis, is a second order cone program (SOCP), which allows us to represent convex quadratic constraints, that often arise in robotic planning problems. Since our model does not discretize time or state, it can be used to efficiently check candidate plans requiring coordination constraints over long horizons.

In the rest of this section we provide additional details of each of the modules of the Scotty Planning System and we discuss the application of the architecture to an example problem.

3.2.1 ScottyActivity

ScottyActivity is a hybrid activity and trajectory planner. ScottyActivity solves *PDDL-S* planning problems, which we describe in Chapter 4. PDDL-S problems augment the well-known PDDL2.1 specification [39] with *continuous effects* and *continuous control variables* that we use to model the robot behaviors present in robotic missions like the one we describe in Section 3.1.

ScottyActivity is a generative planner that uses heuristic forward search and a delete-relaxation heuristic to exploit the breakthrough advances in planning achieved over the two last decades. ScottyActivity uses ScottyConvexPath to efficiently check the consistency of partial skeleton plans, without resorting to time, state or control discretization.

ScottyActivity does not consider obstacles at this stage. The solution found by ScottyActivity contains a grounded plan that consists of an schedule for the chosen activities and a control and state trajectory for each vehicle. However, the solution can also be represented as a qualitative state plan (QSP) that contains a sequence that describes when behaviors are engaged and disengaged along with all the temporal, state and actuation constraints that a valid plan must satisfy. This QSP is then the input to the ScottyPath planner.

We describe ScottyActivity in Part II of this thesis. Chapter 7 describes our extensions to the PDDL2.1 syntax that allow us to express PDDL-S problems. Chapter 8 describes our planning approach. Finally, Chapter 9 describes the experiments performed to evaluate the scalability and performance of ScottyActivity.

3.2.2 ScottyPath

The solution QSPs that ScottyActivity generates do not consider obstacles. *Scotty-Path* finds obstacle-free solution plans that satisfy all the constraints in the QSP. In order to do so, ScottyPath forces each vehicle to always remain inside one of a set of overlapping convex safe regions. These safe regions, which are generated in advance, are provided to ScottyPath as another input, besides the QSP.

The path planning problem solved by ScottyPath is challenging because the QSP does to explicitly define the locations that each robot needs to visit. Instead, it provides a set of constraints that couple the trajectories of multiple robots during the full plan. In order to assign safe regions to each robot, ScottyPath uses an informed search approach that uses ScottyConvexPath to jointly compute the cost and heuristic of candidate search nodes. We use a novel optimization-based approach to determine when intermediate goal conditions can be satisfied by candidate search nodes.

ScottyPath is described in Part III of this thesis. For pedagogical purposes, Chapter 10 describes first a simpler geometric path planner that introduces the convex safe regions and uses a similar approach based on informed search and convex optimization. The ScottyPath planning approach, that handles first-order dynamics, as well as multiple vehicles with coordination constraints, is described in Chapter 11. Finally, Chapter 12 describes the experiments that we use to evaluate the performance and scalability of ScottyPath.

3.2.3 MPCScotty

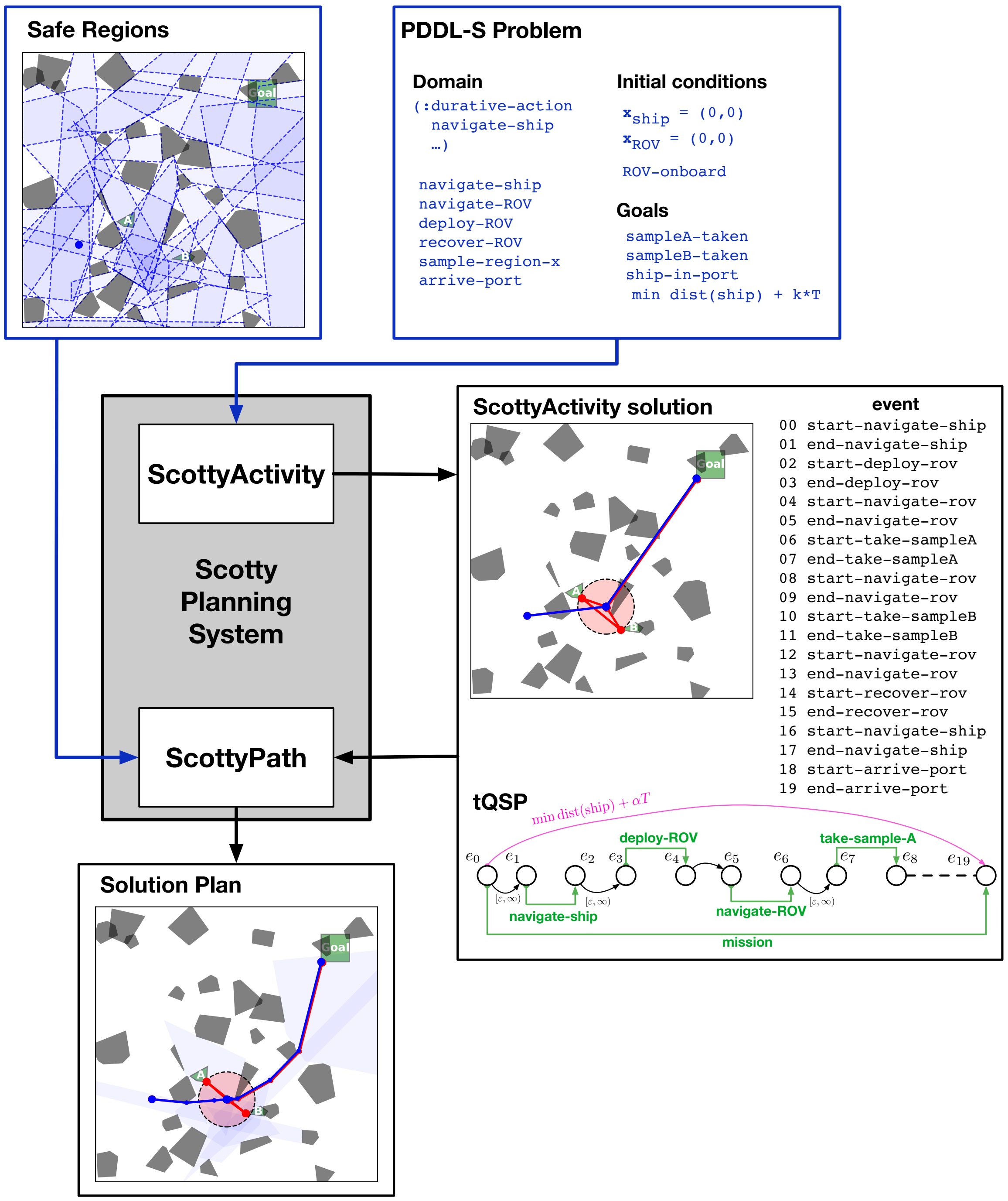
MPCScotty is designed to execute the plans found by the first two stages of the Scotty architecture. A more accurate dynamics model than the simpler, first-order model used by ScottyActivity and ScottyPath is needed during execution. Moreover, this stage also needs to consider vehicle to vehicle collisions as well as new obstacles detected by the robots as well as changing conditions. All these cannot be represented with a convex, continuous-time model. However, the missions that this stage executes still span long durations and cannot be efficiently solved with a time discretized formulation with a small timestep.

MPCScotty solves this problem by using a receding horizon approach that combines, in a single integer program, a limited-horizon discrete-time detailed formulation with a simpler, continuous-time formulation that spans the full plan. The detailed part of the model can represent accurate dynamics and complicated non-convex constraints, since it discretizes time and is limited to a short horizon. The continuous-time part of the model uses the same first-order dynamics and constraints from the convex model that ScottyConvexPath uses. Because this part of the model does not discretize time, it can model the full plan beyond the detailed horizon, and it provides guidance over the full plan. Since this planner uses a receding horizon approach, unforeseen circumstances, such as changing obstacles or inaccurate robot models, are considered in further replanning iterations. The innovation of this planner lies on how the discrete-time and the continuous-time models are joined in a single mixed-integer second order cone program (MISOCP).

MPCScotty is still being developed at the MIT MERS lab and is, therefore, out of the scope for this thesis. However, further details about this planner are provided in Chapter 13.

3.2.4 Example Usage of the Planning Architecture

Figure 3-4 presents an example that illustrates how ScottyActivity and ScottyPath are used to solve a hybrid activity and trajectory planning problem. The problem presented in Figure 3-4 is similar to the example scenario described in Section 3.1, but without the AUV. In this problem, the ROV needs to take samples at regions A and B, and the ship needs to end at the goal region with the ROV on board. As shown in the figure, ScottyActivity takes the problem as a PDDL-S problem. The PDDL-S problem describes the initial conditions of the robots as well as the model of the activities that can be executed, the goal conditions that need to be satisfied at the end of the plan and the objective. The plan returned by ScottyActivity, shown



Domain

```
(:durative-action
   navigate-ship
   …)

navigate-ship
navigate-ROV
deploy-ROV
recover-ROV
sample-region-x
arrive-port
```

Initial conditions

```
xship = (0,0)
xROV = (0,0)
ROV-onboard
```

Goals

```
sampleA-taken
sampleB-taken
ship-in-port
min dist(ship) + k*T
```

event

```
00 start-navigate-ship
01 end-navigate-ship
02 start-deploy-rov
03 end-deploy-rov
04 start-navigate-rov
05 end-navigate-rov
06 start-take-sampleA
07 end-take-sampleA
08 start-navigate-rov
09 end-navigate-rov
10 start-take-sampleB
11 end-take-sampleB
12 start-navigate-rov
13 end-navigate-rov
14 start-recover-rov
15 end-recover-rov
16 start-navigate-ship
17 end-navigate-ship
18 start-arrive-port
19 end-arrive-port
```

Figure 3-4: Example that illustrates how the Scotty Planning System is used to solve a hybrid activity and trajectory planning problem with obstacles.

on the right, is a QSP that describes the sequence of engaged and disengaged robot behaviors along with the constraints that need to be satisfied at each step in the plan. Since the plan found by ScottyActivity ignores obstacles, the trajectory of the ship collides with surface obstacles. ScottyPath then takes the QSP and a set of safe regions for the ship in order to find the obstacle-free plan shown at the bottom of the figure.

Chapter 4

Problem Statement

In this chapter we define the problem statement that Scotty solves. Scotty is designed to solve hybrid activity and trajectory planning problems for robotic missions like the one presented in Section 3.1. Recall that the Scotty Planning System, as described in Section 3.2, divides the hybrid activity and trajectory planning problem into two subproblems that are solved sequentially.

The first problem is the activity planning problem, which we solve with the Scotty-Activity planner. This problem consists of a model of the behaviors or activities that robots can execute, the goals that each robot must achieve, the initial conditions for each robot and an objective function. We call this problem a *PDDL-S* problem.

The second problem that the Scotty Planning System solves is the path planning problem for the qualitative state plan generated by ScottyActivity. This QSP does not consider obstacles. However, it describes the robot behaviors used to solve the problem and the order in which these are engaged and disengaged, as well as the temporal and state constraints that each robot must satisfy at each step in the plan. ScottyPath takes the QSP and a set of convex safe regions for each vehicle, and returns an obstacle-free plan in which each vehicle is guaranteed to always remain inside one of the safe regions.

The input to the Scotty Planning System consists of two parts: the PDDL-S problem, and the sets of safe regions for each vehicle. In this chapter we define PDDL-S problems, along with their solution. We defer describing the safe regions until Part III of this thesis, where they are explained in detail.

4.1 The PDDL-S Problem

Having defined the elements that model robot behaviors, activities and constraints, we now proceed to define formally the problem that the ScottyActivity planner solves. ScottyActivity solves a hybrid activity and trajectory optimization planning problem that we call a *PDDL-S* planning problem.

A PDDL-S problem describes the hybrid activity and trajectory planning problem that ScottyActivity solves. The main element of the PDDL-S problem is the *hybrid activity*, which describes a behavior that a robot can execute. This consists of the effects that derive from the execution of such behavior, such as moving in one direction or enabling or disabling a sensor, and the operating constraints that need to hold at the start, end or while the behavior is being executed. The PDDL-S problem also provides the initial conditions for each robot, the goals that need to be satisfied at the end of the plan and the objective function that should be minimized. We provide a formal definition for PDDL-S problems next.

Definition 4.1 (*PDDL-S Problem*)**.** A *PDDL-S* problem is a planning problem given by the tuple $\langle I, G, CV, A, O\rangle$, where:

- $I = \langle \mathbf{x}_0, \mathbf{p}_0\rangle$ is the initial state, which is a complete assignment to the state variables, $\mathbf{x}_0 = \mathbf{x}(0)$, and propositional variables, $\mathbf{p}_0 = \mathbf{p}(0)$ at the beginning.

- $G = \langle S_G, P_G\rangle$ is the goal. The goal consists of a set of state variable constraints, S_G, that need to be satisfied at the end of the plan and a set of propositional variables P_G whose value needs to be *true* at the end of the plan.

- $CV = \langle \mathbf{c}, CC\rangle$ is the tuple of the vector of control variables, $\mathbf{c}$, and the convex quadratic constraints operating on the control variables, CC, as defined in Section 4.1.4.

- A is the set of hybrid durative activities.

- O is the objective function.

▲

We provide formal definitions for all the elements that compose a PDDL-S problem throughout the rest of this section.

Note that PDDL-S problems have similar characteristics to PDDL2.1 problems [39]. The main differences between PDDL2.1 problems and PDDL-S problems, which we describe in Section 4.1.1, lie in the definition of the continuous effects and control variables supported by our hybrid durative activities. These allow us to represent behaviors with continuously controllable parameters, which we use to model first order dynamics.

4.1.1 Hybrid Durative Activities

In the course of a typical robotic mission, robot behaviors may need to be engaged or disengaged at different times and state constraints may become active depending on the behaviour currently being executed or the goal that the robot is currently trying to achieve. While the main focus of our planner is dealing with continuous behaviors efficiently over long horizons, we also need to be able to reason with discrete conditions and effects in order to find plans for typical robotic missions. We use durative activities like the ones that have long been used by the activity planning community to model the switched behaviors described before. In particular, we use *hybrid durative activities*, which are similar to PDDL2.1 [39] activities except for some differences that we highlight below.

Definition 4.2 (*Hybrid Durative Activity*)**.** A *hybrid durative activity* a is given by the tuple $\langle dur, pre_{\vdash}, eff_{\vdash}, pre_{\leftrightarrow}, eff_{\leftrightarrow}, pre_{\dashv}, eff_{\dashv}\rangle$, where:

- dur is the tuple $\langle d_l, d_u\rangle$ that describes the minimum and maximum duration of a. As in PDDL 2.1, the duration of the activity is assumed to be controllable by the planner.

- $pre_{\vdash}(pre_{\dashv})$ are the conditions that must hold at the *start* (*end*) of a. These conditions can be of two types. First, a propositional condition can require a proposition $p_j \in \mathbf{p}$ to hold. Second, a condition can be a state variable constraint as defined in Section 4.1.3.

- $\mathit{eff}_{\vdash}(\mathit{eff}_{\dashv})$ are the *starting* (*ending*) effects of a that indicate the resulting change in the state as a result of applying the activity. A set of effects $\mathit{eff}_x, x \in \{\vdash, \dashv\}$ consists of:
  - eff_x^-, the set of propositions to be deleted (set to *false*) from the state.
  - eff_x^+, the set of propositions to be added (set to *true*) to the state.
- $\mathit{pre}_{\leftrightarrow}$ are the invariant conditions of a that must hold throughout the duration of the activity. These can also be propositions that need to hold or state variable constraints.
- $\mathit{eff}_{\leftrightarrow}$ are the *continuous effects* of a that are active while the activity is being executed.

▲

The main differences between *hybrid durative activities* and PDDL2.1 durative activities are two. First, we support continuous effects that depend on control variables. Second, we support convex quadratic constraints. While convex quadratic constraints can be specified in PDDL2.1, we are not aware of any other PDDL2.1 planner that supports them.

4.1.2 State

The *state* of the system is given by a set of discrete *propositional* variables, and a set of continuous *state* variables. The discrete variables are boolean variables are used, for example, to model that a certain sensor or motor is turned on, or whether a sample has been taking in a certain region. The state variables, which are continuous, represent the position of the robot, as well as other properties, such as the battery level of a vehicle.

Definition 4.3 (*State*)**.** The *state* of the system, $\mathbf{s} = \langle \mathbf{x}, \mathbf{p} \rangle$, is a tuple of *state variables*, $\mathbf{x}$, and *propositional variables*, $\mathbf{p}$. The state variables are given by a vector of real valued variables, $\mathbf{x} = \langle x_1, x_2, \ldots, x_n \rangle \in \mathbb{R}^n$. The propositional variables are given by a vector of propositions that can be true or false, $\mathbf{p} = \langle p_1, p_2, \ldots, p_l \rangle$. ▲

We further define a special kind of state variables that we call *resources*. By imposing some special restrictions on resource variables, we can use them to model complicated effects such as the velocity-dependant decrease on the battery level. The special effects that we can subject resources to are defined in Section 4.1.4.2. The necessary restrictions that we impose on resources are explained later, when we describe our convex model in Chapter 6.

Definition 4.4 (*Resource*)**.** A *resource* is a type of state variable that is subject to the following restrictions:

- Resources can only appear with negative coefficients in the objective function. That is, they can only be maximized.
- Resources can only be subject to *greater or equal than* constraints.

▲

In the example scenario presented in Section 3.1, the state variables are the x and y coordinates of the ship, the AUV and the ROV as well as the battery level of the AUV. Additionally, the battery level is a state variable that is a resource. The propositional variables in the example scenario indicate, for example, whether the images and samples have been taken, or whether the AUV and ROV are deployed or on-board the ship.

4.1.3 State Constraints

Robots are often subject to state constraints. Some of these constraints may be intrinsic to their dynamics or modes of operation. For example, the AUV in the example scenario can only continue moving while its battery level is greater than 0. Similarly, the ROV can only separate from the ship a distance smaller than the length of the tether than connects them. Other state constraints, however, are mission dependent. For example, in order to successfully capture the images required for the mission, the AUV needs to be inside region A while the pictures are taken.

We restrict the state constraints in our model to convex quadratic constraints. This restriction allows us to efficiently check constraints that need to be maintained over arbitrarily long durations, as explained in Chapter 5, and to use an efficient convex model and a state of the art convex quadratic solver, as explained in Chapter 6. However, note that convex quadratic constraints are sufficient to express a wide range of real-world constraints that appear in typical robotic missions, like the ones shown in the example scenario.

State variable constraints are defined as follows:

Definition 4.5 (*Convex Quadratic State Constraint*)**.** A *convex quadratic state constraint* is a constraint in the form of $g(\mathbf{x}) \leq 0$, where $g : \mathbb{R}^n \to \mathbb{R}$ is a convex quadratic function operating on the vector of state variables, $\mathbf{x}$. ▲

Recall that any quadratic function $g : \mathbb{R}^n \to \mathbb{R}$ can be written as $g(\mathbf{x}) = \mathbf{x}^T A\mathbf{x} + \mathbf{k}^T\mathbf{x} + b$, where $A \in \mathbb{R}^{n\times n}$, $\mathbf{k} \in \mathbb{R}^n$, and $b \in \mathbb{R}$ are constants. Function g is convex if and only if A is a positive semidefinite matrix [14]. Note that since any linear expression is also a convex quadratic expression, it is also possible under our model to subject state variables to linear inequality constraints. As a consequence, linear equality constraints are also possible.

As mentioned before, the example scenario presents multiple state constraints. The ROV tether constraint, for example, is represented with the convex quadratic constraint: $(x_{ROV} - x_{ship})^2 + (y_{ROV} - y_{ship})^2 - R_{tether}^2 \leq 0$. The constraint that forces the AUV to remain inside the images region is also a convex quadratic constraint. In particular, the images region is a polyhedron and the constraint is, therefore, a linear inequality constraint in the form of $A \cdot \mathbf{x}_{AUV} \leq \mathbf{b}$. The battery level of the AUV is a resource that is subject to always be nonnegative, $b_{AUV} \geq 0$.

4.1.4 Control Variables and Continuous Effects

As most activity planners do, we operate on the assumption that the planner is the sole agent of change. As a result, state variables can only change their value through continuous effects operating on them. These continuous effects may be seen as behaviors that enable or disable robot dynamics. The change on the state variables due to the presence of continuous effects depends on the *control variables* that these effects make use of. Control variables are continuous parameters that are assumed to be controlled within their lower and upper bounds. The main purpose of control variables is to model the controllable velocities of robots. However, control variables can also model, for example, a controllable rate of change of a battery, as long as the dynamics are represented with the continuous effects that we describe in this section. Control variables are defined as follows.

Definition 4.6 (*Control Variables,* $\mathbf{c}$)**.** The control variables vector, $\mathbf{c}$, is a vector of control variables $\mathbf{c} = \langle c_1, c_2, \ldots, c_m \rangle$, where each control variable c_j is a real valued parameter that is continuously controllable within its fixed lower and upper bounds, c_{j_l} and c_{j_u}. ▲

In the example scenario, the control variables vector is a vector of the v_x, v_y velocities of the ship, the AUV and the ROV, $\mathbf{c} = \langle v_{x_{ship}}, v_{y_{ship}}, v_{x_{AUV}}, v_{y_{AUV}}, v_{x_{ROV}}, v_{y_{ROV}} \rangle$.

Control variables can be subject to control variable constraints. We restrict the control variable constraints to convex quadratic constraints for reasons that will become apparent when we describe, in Chapter 6, the convex optimization model that Scotty uses.

Definition 4.7 (*Convex Quadratic Control Variable Constraint*)**.** A *convex quadratic control variable constraint* is a constraint in the form of $g(\mathbf{c}) \leq 0$, where $g : \mathbb{R}^m \rightarrow \mathbb{R}$ is a convex quadratic function operating on the vector of control variables, $\mathbf{c}$. ▲

In the example scenario, there are three convex quadratic constraints operating on the control variables. These three constraints limit the l^2-norm of the velocities of each of the three vehicles, $\|\mathbf{v}_v\|_2 \leq v_{max_v} \quad \forall v \in \{\text{ship}, \text{ROV}, \text{AUV}\}$. Since l^2-norms are convex quadratic functions, the previous constraint can be written as a convex quadratic constraint: $v_{x_v}^2 + v_{y_v}^2 - v_{max_i}^2 \leq 0$.

As described in the beginning of this section, continuous effects produce the change in state variables. Continuous effects are defined as follows.

Definition 4.8 (*Continuous Effect*)**.** A *continuous effect* is a tuple $\langle x, f \rangle$ where x is the state variable that is subject to the effect and $f : \mathbb{R}^m \to \mathbb{R}$ is a function of the control variables vector. An active continuous effect eff on variable x induces a rate of change $\dot{x}_{eff}(t)$ on x such that $\dot{x}_{eff}(t) = f(\mathbf{c}(\mathbf{t}))$. ▲

The change produced in state variable x due to continuous effect *eff* being active from t_a to t_b is given by:

$$\Delta x_{eff}(t) = \int_{t_a}^{t} f(\mathbf{c}(\tau))d\tau, \quad t_a \leq t \leq t_b \tag{4.1}$$

As described later in this thesis in Chapter 5, the solution that ScottyActivity finds restricts the control variables vector to follow a piecewise constant trajectory. This allows us to maintain a continuous time formulation and to efficiently plan for long horizons. This restriction implies that the rates of change for each state variable are piecewise constant and, as a consequence,the state trajectory is piecewise linear.

Continuous effects are additive. That is, multiple continuous effects can be operating on a state variable x during an interval of time. In that case, the rate of change of state variable x is the sum of the rates of change induced by each continuous effect:

$$\dot{x}(t) = \sum_{eff \in E_x(t_a,t_b)} \dot{x}_{eff}, \qquad t_a \leq t \leq t_b \tag{4.2}$$

$$x(t) = x(t_a) + \sum_{eff \in E_x(t_a,t_b)} \Delta x_{eff}(t), \qquad t_a \leq t \leq t_b \tag{4.3}$$

where $E_x(t_a, t_b)$ is the set of active continuous effects operating on x between t_a and t_b.

Note that, as we described above, state variables only change their value due to the presence of continuous effects operating on them. Therefore, the schedule of the continuous effects that are applied to each state variable and the times when these are active, together with the trajectory of the control variables, $\mathbf{c}(t)$, fully determine the trajectory of all state variables, $\mathbf{x}(t)$.

While the description for continuous effects that we describe above can be used with any choice of f, we allow two types of continuous effects in this work. These are described below.

4.1.4.1 Controllable Linear Time-varying Continuous Effects

The first type of continuous effect that we describe induces a rate of change that is a linear combination of the control variables.

Definition 4.9 (*Controllable linear time-varying continuous effect, CLTE*)**.** A *CLTE* is defined by a tuple $\langle x, \mathbf{k} \rangle$, where x is the state variable subject to the effect and $\mathbf{k} \in \mathbb{R}^m$ is a constant vector. A *CLTE* changes a state variable linearly in time with a rate of change that is a linear combination of its control variables. The change in state variable x up to time t due to an ongoing *CLTE* that started at time 0 is given by

$$\Delta x_{CLTE}(t) = \int_0^t \mathbf{k}^T \cdot \mathbf{c}(\tau) d\tau \tag{4.4}$$

▲

In the example scenario, the *CLTEs* allow the vehicles to move according to their control variable velocities. There are six *CLTE* effects, one for each position variable of each of the three vehicles. For example, the two CLTE effects operating each on x_{AUV} and y_{AUV} induce the rates of change $\dot{x}_{AUV} = v_{x_{AUV}}$ and $\dot{y}_{AUV} = v_{y_{AUV}}$.

4.1.4.2 Resource-constrained Norm Effects

The other type of continuous effects that we allow describe a change in a state variable that depends on the l^2-norm of a vector of control variables. This type of effect is useful to model, for example, how the battery of a vehicle decreases as a function of the magnitude of the velocity that the vehicle moves with. For reasons that will become apparent when we introduce our convex optimization model in Chapter 6, we restrict this effect to only be applied to state variables that are *resources*.

Definition 4.10 (*Resource-constrained norm effect, RNE*)**.** A *RNE* is given by a tuple $\langle x, k, \mathbf{c}_e, f \rangle$, where $k \in \mathbb{R}_{>0}$ is a positive real constant, $\mathbf{c}_e$ is a vector of m or fewer control variables, and $f : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is a real nonnegative function. A *RNE* decreases the value of a constrained state variable (resource) x with a rate of change proportional to the function f of the l^2-norm of $\mathbf{c}$, $f(\|\mathbf{c}\|_2)$. ▲

The change on a continuous state variable x up to time t due to an ongoing *RNE* that started at time 0 is given by:

$$\Delta x_{RNE}(t) = -\int_0^t k \cdot f(\|\mathbf{c}_e(\tau)\|) d\tau, \quad k \geq 0 \tag{4.5}$$

In this work we consider two types of *RNEs*. A *linear norm effect* (LNE), $\langle x, k, \mathbf{c}_e, x \to x \rangle$, produces a decrease rate proportional to the norm of $\mathbf{c}_e$. A *linear squared norm effect* (LSNE), $\langle x, k, \mathbf{c}_e, x \to x^2 \rangle$, produces a decrease rate proportional to the square of the norm of $\mathbf{c}_e$.

In the example scenario, the *navigate* activity has a *LNE* effect that makes the battery of the AUV decrease with a rate proportional to the norm of its velocity ($\dot{b}_{AUV}(t) = -k \cdot \|\mathbf{v_{AUV}}\|$). This is equivalent to stating that the battery decrease is proportional to the distance traveled by the AUV. The same effect depending on the square of the norm ($\|\mathbf{v_{AUV}}\|^2$) can be achieved by using a *LSNE* effect instead of the *LNE* one. Since continuous effects are additive, it is also possible to define a change in the battery that is a linear combination of these terms and other *CLTE* effects.

4.1.5 Objective

The objective function of a *PDDL-S* problem is the function that the planner aims to minimize. We restrict the objective O to a linear combination of one or more of the following terms:

- The total duration of the plan (plan makespan).

- The value of a state variable at the end of the plan. In the case of resources, their coefficients are limited to negative values (i.e. resources can only be maximized).
- The sumproducts of the norms (or squared norms) of a vector of control variables and the durations it was active for, i.e. $\int_0^T \|\mathbf{c}_e(\tau)\|^{\{1,2\}}\, d\tau$.

For example, in the motivating scenario, one of the terms minimized is $\int \|\mathbf{v_{SHIP}}\|\, dt$, which minimizes the distance traveled by the ship. A similar term involving, instead, the square of the norm is also possible. This is useful when the control effort needs to be minimized.

4.2 Solution to a PDDL-S Problem

The solution to a PDDL-S problem is given by a plan.

Definition 4.11 (*PDDL-S Plan*)**.** A *PDDL-S Plan* is a tuple $\langle S, \mathbf{f_c} : \mathbb{R} \rightarrow \mathbb{R}^m \rangle$, where

- S is the activity schedule, and is given by a list of triples $\langle a, t_s, d \rangle$, where a is an activity, t_s its start time and d its duration. The latest end time of all activities is denoted by T and corresponds to the plan makespan.
- $\mathbf{f_c} : [0, T) \rightarrow \mathbb{R}^m$ is the control trajectory. The control trajectory assigns a value to all control variables at every time t between the start and the end of the plan.

▲

Given a *PDDL-S plan*, the trajectories of all state variables are fully determined at all times throughout the duration of the plan. In order to compute the value of a state variable x_i at time t, $x_i(t)$, it suffices to apply from 0 to t the continuous effects of the activities for the durations and values of the control variables specified in the plan.

A *valid PDDL-S plan* satisfies all constraints defined by the PDDL-S problem.

Definition 4.12 (*Valid PDDL-S Plan*)**.** A *valid PDDL-S plan* is a *PDDL-S plan* such that:

1. For each activity a which starts at time t, all its discrete and continuous *at start* conditions are satisfied right before time t.
2. For each activity a which ends at time t, all its discrete and continuous *at end* conditions are satisfied right before time t.
3. For each activity a which is ongoing at time t, all its discrete and continuous *over all* conditions are satisfied at time t.
4. The trajectories of the control variables satisfy each global control variable constraint throughout the duration of the plan.
5. The final state at the end of the plan satisfies the goal constraints.

▲

An *optimal PDDL-S plan* is a *valid PDDL-S plan* such that objective O takes the minimum possible value. ScottyActivity finds *valid PDDL-S plans* but it is not guaranteed to find *optimal PDDL-S plans* due to the greedy nature of its search algorithm, as we explain in Chapter 8.

4.2.1 PDDL-S Plans with Piecewise Constant Control

For reasons explained in Chapter 5, the output of ScottyActivity is a PDDL-S plan with piecewise constant control.

Definition 4.13 (*PDDL-S Plan with Piecewise Constant Control*)**.** A *PDDL-S Plan with Piecewise Constant Control* is a PDDL-S plan in which the control trajectory is a piecewise constant function where the change points occur at the start and end times of the activities in the plan schedule. A PDDL-S plan with piecewise constant control has the following properties:

1. $N - 1$ *stages*, where N is the number of events in the plan, where an event is the start or end of an activity. Each stage denotes the period of time between consecutive events in the plan.

2. A control trajectory, $\mathbf{c}(t)$, that is piecewise constant. The control variables vector is constant during each stage in the plan.

▲

Since state variables change linearly in time due to continuous effects, the state trajectory in a PDDL-S plan with piecewise constant control is piecewise linear. The state variables change linearly in time during each stage in the plan.

This restriction of piecewise constant control variables and piecewise linear state trajectories does not affect the completeness of ScottyActivity.

Theorem 4.1 (Completeness of PDDL-S Plans with Piecewise Constant Control)**.** *If a PDDL-S problem has a solution, there always exists a solution that is a PDDL-S plan with piecewise constant control.*

Similarly, this restriction does not affect the optimality of the plans that can be obtained.

Theorem 4.2 (Optimality of PDDL-S Plans with Piecewise Constant Control)**.** *The optimal solution to a PDDL-S problem, if one exists, is a PDDL-S plan with piecewise constant control.*

In effect, our linear time dynamics and the absence of obstacles, curvature constraints or other non-convex constraints ensure that any problem solvable with an arbitrarily changing state trajectory is also solvable with a piecewise linear one. Furthermore, since both the state conditions and the objective are required to be convex, the piecewise linear restriction of state variables does not have an effect in optimality either. The optimal solution for a PDDL-S problem is a PDDL-S plan with piecewise constant control. We present the proof for both theorems in Appendix A.

4.2.2 Qualitative State Plans

The PDDL-S plans that ScottyActivity finds can also be represented as qualitative state plans (QSPs). QSPs describe the behaviors that robots execute and when these are engaged or disengaged. QSPs also define the state constraints that robots need to satisfy at different steps in the plan.

We defer the formal description of QSPs, as well as the equivalence between PDDL-S plans and QSPs to Chapter 11 in Part III of this thesis, where they are presented in the context of the ScottyPath planner.

Part I

ScottyConvexPath

Table of Contents

Chapter 5

Efficient Satisfaction of Convex Conditions Over Arbitrarily Long Horizons Through Piecewise Constant Control Trajectories

This chapter starts Part I of this thesis, which describes *ScottyConvexPath*, the sub-planner that both ScottyActivity and ScottyPath use to efficiently check candidate skeleton plans over arbitrarily long durations. In this chapter we present Scotty's approach to satisfying convex state constraints through arbitrarily long durations in an efficient manner. As described in Chapter 4, state constraints are imposed by activities as their *at start*, *at end* and *over all* conditions. One issue the planner must address is to satisfy state constraints throughout continuous time, and not just at sampled events. The second issue is that the method for satisfying state constraints must scale computationally to long horizon problems.

Most robotic planning algorithms enforce these constraints by discretizing states at fixed timesteps and imposing the constraints at each discretized state. Many of these algorithms ignore the constraints between the discretized points of the trajectory. While this strategy does not guarantee that the conditions are satisfied at all times, this usually works well when the discretization time is sufficiently small. Unfortunately, these approaches do not work well for long time horizons, or when different activities happen at a different time scale.

One of the advantages of the Scotty Planning System is that it performs equally well for short and long time horizons and different time scales since time is not discretized. While avoiding time discretization greatly helps performance, it imposes the challenge of how to enforce maintenance conditions throughout horizons that can be arbitrarily long. In order to enforce invariant conditions in an efficient manner, we restrict the trajectories of the control variables chosen by the planner to be piecewise constant. We do not restrict how many segments the planner can select. Therefore, the trajectories of the control variables are given by:

$$\mathbf{c}(t) = [c_1(t), c_2(t), \ldots, c_m(t)] = \mathbf{c}_j, \quad t_j \leq t < t_{j+1}, \quad j \in 0 \ldots N-1 \tag{5.1}$$

, where N is the total number of piecewise constant segments, t_j, called a *switch point*, is the starting time of segment j, $\mathbf{c}(t) \in \mathbb{R}^m$ is the vector of values of all control variables at time t and $\mathbf{c}_j \in \mathbb{R}^m$ is the vector of constant values of all control variables during segment j, between t_j and t_{j+1}.

As described in the problem statement (Chapter 4), continuous change in the state variables only occurs as a result of the action of continuous effects. Recall that the change in a state variable x subject to a continuous effect from the time the effect starts, t_a, to time t can be expressed as:

$$\Delta x_{eff}(t) = \int_{t_a}^{t} f(\mathbf{c}(\tau))d\tau, \quad t_a \leq t \leq t_b \tag{5.2}$$

, where f is a function of the control variables that depends on the type of continuous effect. Since the control variables are piecewise constant, $g(\mathbf{c}(\tau))$ is constant and the previous equation can be simplified to:

$$\Delta x_{eff}(t) = f(\mathbf{c}_l)(t_{l+1} - t_a) + \sum_{k=l+1}^{l+n-1} f(\mathbf{c}_k)(t_{k+1} - t_k) + f(\mathbf{c}_{l+n})(t - t_{l+n}) \tag{5.3}$$

$$t_l \leq t_a \leq t_{l+1} \leq \ldots \leq t_{l+n} \leq t \tag{5.4}$$

, where t_l is the last switch point before t_a and there are n switch points between t_a and t. Equation (5.3) shows that the change in state variable x is linear in time. Since all continuous effects can be expressed in this way and are additive, the piecewise constant restriction on control variables results in state variable trajectories that are piecewise linear in time. The beginning and ends of the linear segments correspond to the switch points of the trajectory of control variables.

Therefore, the trajectory of the state variables in the linear segment l between consecutive switch points t_l and t_{l+1} can be written as:

$$\mathbf{x}(t) = \mathbf{x}(t_l) + \mathbf{C}_\mathbf{x}(t_l \to t_{l+1}) \cdot (t - t_l), \quad t_l \leq t \leq t_{l+1} \tag{5.5}$$

$$\mathbf{x}(t) = [x_1(t) \ldots x_n(t)]^T \tag{5.6}$$

$$\mathbf{C}_\mathbf{x}(t_l \to t_{l+1}) = [C_{x_1}(t_l \to t_{l+1}) \ldots C_{x_n}(t_l \to t_{l+1})]^T \tag{5.7}$$

, where $C_{x_i}(t_l \to t_{l+1}) \in \mathbb{R}$ is a constant value that represents the constant rate of change in state variable x_i due to all the continuous effects operating on the state variable and that is a function of the constant vector of control variables $\mathbf{c}_l$ at segment l.

The piecewise linear restriction on state variables is very useful for our purposes. It allows us to impose conditions over long horizons in an efficient way, without needing to resort to time discretization. We can do this as long as the conditions are *convex*, which they are given our problem statement. In effect, convexity properties ensure that a linear segment is fully contained in a convex set as long as the ends of the segment are contained in the set. As a consequence, in order to ensure that an *invariant* convex state condition is satisfied at all times, we only need to ensure that the convex condition is satisfied at the *switch points* of the state trajectory. The switch points can be separated in time arbitrarily and therefore this is an efficient way to enforce invariant conditions. This is shown with the following lemma and its corresponding proof.

Lemma 5.1. *If the switch points of the piecewise linear state trajectory are contained in a convex set, the full trajectory is contained in the convex set.*

Proof. A set $S \subseteq \mathbb{R}^n$ is convex if and only if $\forall \mathbf{a}, \mathbf{b} \in S$ and $\forall \lambda \in [0, 1]$ we have that $(1 - \lambda)\mathbf{a} + \lambda\mathbf{b} \in S$ as well. Since equation (5.5) describes a straight line of a segment of the piecewise linear trajectory, it can be reformulated in terms of the switch points (or extreme points of the segments) as $\mathbf{x}(t) = \mathbf{x}(t_a) + \frac{t-t_a}{t_b-t_a}(\mathbf{x}(t_b) - \mathbf{x}(t_a))$. Taking $\mathbf{a} = \mathbf{x}(t_a), \mathbf{x} = \mathbf{x}(t_b), \lambda = \frac{t-t_a}{t_b-t_a}$, we get that $\mathbf{x}(t) = (1 - \lambda)\mathbf{a} + \lambda\mathbf{b}$. Since S is a convex set and $a, b \in S$, from the definition of convexity, $\mathbf{x}(t) \in S$ as well. This is true for any value of $t \in [t_a, t_b]$ and for any value of the control variables, as long as they are constant, which they are, given the piecewise constant restriction imposed earlier. □

As described in Section 4.2.1, the piecewise constant restriction on the control trajectory does not affect the completeness or the optimality of the plans that can be found. This is the case given the first order dynamics restriction, and the presence of only convex state conditions and a convex objective function. Recall that the proofs for the completeness and optimality theorems (Theorems 4.1 and 4.2) is provided in Appendix A.

How the piecewise constant control trajectory is built varies slightly in Scotty-Activity and ScottyPath. ScottyActivity's approach places the piecewise constant control segments between consecutive events in skeleton plans. This is described in detail in Chapter 8. For now, it suffices to mention that maintenance state conditions are only added or removed whenever an activity starts or ends (i.e., an event). This means that the maintenance conditions do not change in between events. Therefore, our approach places the switch points of the piecewise linear trajectories at the starts and ends of activities, as this is sufficient to capture the requirements of the problem.

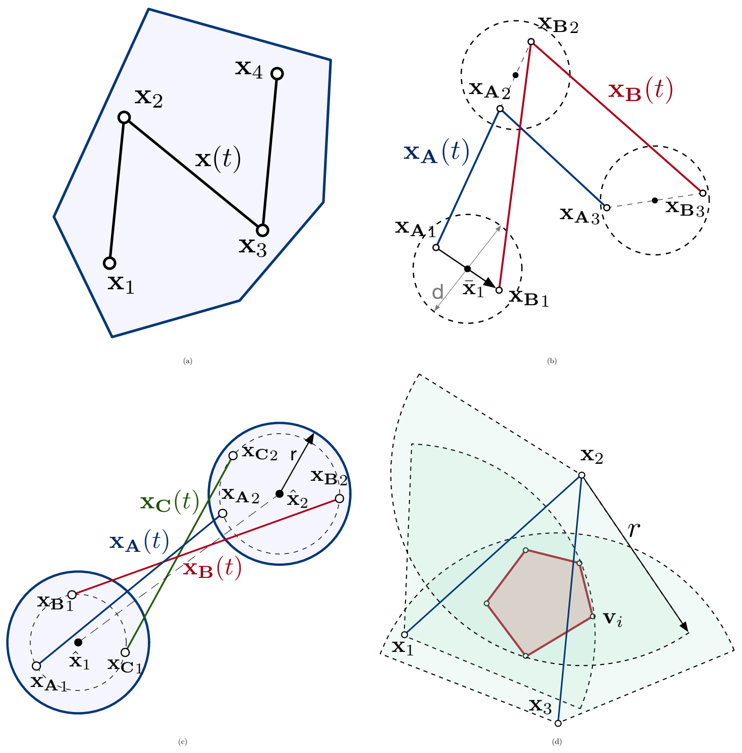

$\mathbf{x}_4$
$\mathbf{x}_2$
$\mathbf{x}(t)$
$\mathbf{x}_3$
$\mathbf{x}_1$
(a)
$\mathbf{x}_{\mathbf{B}2}$
$\mathbf{x}_{\mathbf{A}2}$
$\mathbf{x}_{\mathbf{B}}(t)$
$\mathbf{x}_{\mathbf{A}}(t)$
$\mathbf{x}_{\mathbf{A}1}$
$\mathbf{x}_{\mathbf{A}3}$
$\mathbf{x}_{\mathbf{B}3}$
d
$\bar{\mathbf{x}}_1$
$\mathbf{x}_{\mathbf{B}1}$
(b)
$\mathbf{x}_{\mathbf{C}2}$
r
$\mathbf{x}_{\mathbf{B}2}$
$\mathbf{x}_{\mathbf{C}}(t)$
$\mathbf{x}_{\mathbf{A}2}$
$\hat{\mathbf{x}}_2$
$\mathbf{x}_{\mathbf{A}}(t)$
$\mathbf{x}_{\mathbf{B}}(t)$
$\mathbf{x}_{\mathbf{B}1}$
$\hat{\mathbf{x}}_1$
$\mathbf{x}_{\mathbf{C}1}$
$\mathbf{x}_{\mathbf{A}1}$
(c)
$\mathbf{x}_2$
r
$\mathbf{v}_i$
$\mathbf{x}_1$
$\mathbf{x}_3$
(d)

Figure 5-1: Interesting convex conditions that arise in robotic planning problems..

The piecewise constant control trajectory is slightly different in ScottyPath. In this case, we allow multiple constant control segments between consecutive events. This is necessary to avoid obstacles. During each of these segments, vehicles are constrained to remain inside convex safe regions. We defer all the details about the approach used by ScottyPath to Part II of this thesis.

5.1 Typical Maintenance Convex Conditions in Robotic Applications

We now proceed to give examples of useful convex conditions that commonly arise in robotic planning problems and that, as we have seen, we can enforce efficiently over long horizons.

- **Remain inside a convex region** (Figure 5-1a). As explained in the previous region, we can enforce that a robot remains inside a convex region while moving by imposing that the switch points of the trajectory be inside the convex region. Since trajectories are piecewise linear, ensuring that the switch points are contained in the convex region guarantees that the full trajectory is also contained.

- **Proximity constraint** (Figure 5-1b). We can impose that agents A and B stay within a maximum distance d while performing an activity. Let $\mathbf{x_A}(t)$ and $\mathbf{x_B}(t)$ be the piecewise linear trajectories of A and B. Imposing that both agents are within distance d is equivalent to imposing that $\mathbf{x_A}(t) - \mathbf{x_B}(t) \in C_d$ where C_d is a circle of radius d centered in the origin. Since C_d is a convex set and $\mathbf{x_A}(t)$ and $\mathbf{x_B}(t)$ are piecewise linear, $\bar{\mathbf{x}}(t) = \mathbf{x_A}(t) - \mathbf{x_B}(t)$ is piecewise linear too, and we only need to impose that $\bar{\mathbf{x}}(t_j) \in C_d$ at each switch point j to ensure that A and B are always within distance d.

- **N vehicles inside a circle of radius** r (Figure 5-1c). The center of the n vehicles is given by $\hat{\mathbf{x}}(t) = \frac{\sum^n \mathbf{x_i}(t)}{n}$. The condition that all vehicles remain inside a circle of radius r is equivalent to the n conditions $||\mathbf{x_i}(t) - \hat{\mathbf{x}}(t)|| \leq r$. Since $\mathbf{x_i}(t) - \hat{\mathbf{x}}(t)$ is piecewise linear due to being a linear combination of piecewise linear trajectories and since the norm constraint defines a convex set, the n conditions only need to be imposed at the switch points to ensure that all n vehicles are always inside the circle of radius r at all times.

- **Coverage constraint of a polygonal region** (Figure 5-1d). We can similarly impose that a circle of radius r centered at a moving vehicle always covers a fixed polygonal region. This is equivalent to satisfying the convex constraints $||\mathbf{x}(t) - \mathbf{v_i}|| \leq r$ for each of the $\mathbf{v}_1 \ldots \mathbf{v}_m$ vertices of the polygon. Again, since the m constraints are convex and the motion of the vehicle is piecewise linear, we only need to impose the constraints at each of the switch points of the trajectory of the vehicle.

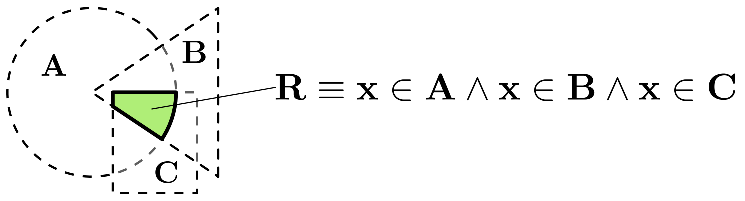

A
B
C
$\mathbf{R} \equiv \mathbf{x} \in \mathbf{A} \wedge \mathbf{x} \in \mathbf{B} \wedge \mathbf{x} \in \mathbf{C}$

Figure 5-2: Complex condition defined by the intersection of simple convex sets.

Further since the intersection of convex sets is convex, we can represent complicated convex regions by intersecting simple convex primitives. Figure 5-2 shows, for example, how a complex region can be achieved by intersecting a circle, a triangle and a rectangle. Ensuring that a vehicle remains inside such region can be achieved by imposing that the switch points of the trajectories remain inside the circle, the rectangle and the triangle at the same time.

Chapter 6

ScottyConvexPath: Trajectory Planning for Skeleton Plans Over Long Horizons With Convex Optimization

An important characteristic of the Scotty Planning System is that it is able to plan over long horizons. This is made possible by the ScottyConvexPath planner, since it uses a continuous time formulation. ScottyConvexPath solves its relaxed planning problem by reformulating it as a convex optimization problem. In this chapter, we describe in detail the convex model that ScottyConvexPath uses to solve the planning problem. Both ScottyActivity and ScottyPath repeatedly use ScottyConvexPath as a subplanner at different steps of their search

In particular, ScottyActivity uses ScottyConvexPath for the following:

- To check the consistency of partial plans.
- To compute the minimum and maximum bounds that each state variable can reach at the end of a given partial plan.
- To compute the cost of a partial plan if the obj-EHC search algorithm is used.

- To compute the schedule of the activities and the control and state trajectories of the solution plan.

On the other hand, ScottyPath uses ScottyConvexPath for the following purposes:

- To check the consistency of a candidate partial safe region assignment for the vehicles.
- To jointly compute the cost and heuristic value of a candidate partial safe region assignment for the vehicles.
- To determine whether a candidate partial safe region assignment can satisfy a given goal in the QSP.
- To compute the schedule of the events and the control and state trajectories of the final obstacle-free plan.

The convex model that ScottyConvexPath uses to solve relaxed problems for ScottyActivity and ScottyPath has the same decision variables and constraints. However, it differs in the objective used in each case. We explain, in detail, the purpose of all these uses of the model when we describe the ScottyActivity and ScottyPath approach in Chapters 8 and 11, respectively.

The straightforward mathematical model that describes the characteristics of the continuous effects and other constraints defined in Chapter 4 is non-linear and non-convex. A key innovation of our work consists in describing an alternative encoding that is convex and can be represented as a Second Order Cone Program (SOCP), a class of convex quadratically constrained programs. Being able to use a SOCP model is important for several reasons. First, there are very efficient solvers for SOCP problems that are orders of magnitude faster than traditional gradient-based methods typically used to solve general non-linear problems. Both ScottyActivity and ScottyPath need to check the consistency of candidate search states using ScottyConvexPath. As a result, thousands of optimization problems are solved while planning for typical robotic missions. Therefore, the runtime of the solver accounts for the majority of the planning time. Second, unlike general non-linear problems, convex optimization problems can be solved with complete algorithms. As a consequence, Scotty's feasibility check is complete and we are guaranteed to only reject states that are infeasible. Finally, unlike general non-linear problems, convex problems do not have local minima. As a consequence, the solution returned by convex solvers is optimal. This is essential for both ScottyActivity and ScottyPath. In the case of ScottyActivity, the optimality of the solution returned by the solver guarantees that the minimum and maximum bounds for each state variable are computed accurately, which is required to ensure that no valid states are incorrectly pruned. Moreover, it also guarantees that the solution plan returned is optimal for the sequence of starts and ends of activities that the heuristic guided search finds. In the case of Scotty-Path, we need the solver to return optimal solutions to guarantee that the cost and heuristic computed for each search node is accurate. This allows us to find optimal obstacle-free paths for the input QSP problems.

SOCPs are harder to solve than LPs. However, they can still be solved in polynomial-time with interior point algorithms. Moreover, our model only uses cone constraints when the problem has convex quadratic state or control variable constraints, or *resource-constrained norm effects*. When these characteristics are absent, our model becomes a linear program. That is, we only suffer the performance degradation of transitioning from LP to SOCP when the problem requires it.

ScottyConvexPath checks the consistency of partial plans that we call plan skeletons, and that we define in the next section. Plan skeletons are augmented with additional activities as the search advances. An important characteristic of our approach is that the event times and state and control trajectories are computed from scratch for every plan skeleton, as opposed to reusing the values found for common part of the plan that is shared with the parent plan skeleton. Therefore, we do not make early commitments to the event times or values of the state or control variables. That is, the optimization can make some choices for event times and state variables at some state in the search, and completely different ones at a later descendant search node. This is important since it allows us to prevent early bad choices that could lead to infeasible or suboptimal plans later on and it lets us provide as much flexibility as possible to the solver in order to make the best choice according to the objective [92].

For clarity purposes, this chapter describes the model that ScottyConvexPath uses from the perspective of the ScottyActivity, where the problem solved is the activity planning problem. The model that ScottyConvexPath uses to solve ScottyPath relaxed problems is slightly different. These differences are explained when we describe the ScottyPath planner in Part III of this thesis.

In the next section of this chapter, we describe plan skeletons and present an example that we use throughout the the rest of this section to illustrate the different parts of the model. The rest of the sections of this chapter describe the decision variables, the constraints and the objective of our model.

6.1 Plan Skeletons

Our convex model is used to check the consistency and to compute the schedule and the control and state trajectories of a *plan skeleton*. A plan skeleton is a qualitative state plan that defines the activities that are selected and the constraints that need to be satisfied at every step of the plan. As we describe in Chapter 8, when we detail ScottyActivity's approach, each activity is divided into a start and an end activity that we call *event*. A plan skeleton is given by a totally ordered sequence of start and end events. Each search state generated during ScottyActivity's search contains a sequence of start and end events that forms a plan skeleton.

We call *stage* the period of time between consecutive events in a plan skeleton. Recall that the control trajectory that ScottyActivity finds is piecewise constant. The constant segments of such trajectory correspond to the stages in the plan skeleton. In other words, the vector of control variables is allowed to change to a different constant value at each event, but needs to remain constant during each stage. We briefly mention here that the main difference between the ScottyActivity convex model and the ScottyPath model is that in the latter, consecutive events in the plan skeleton are connected by a sequence of stages, instead of by a single one. This is needed to avoid

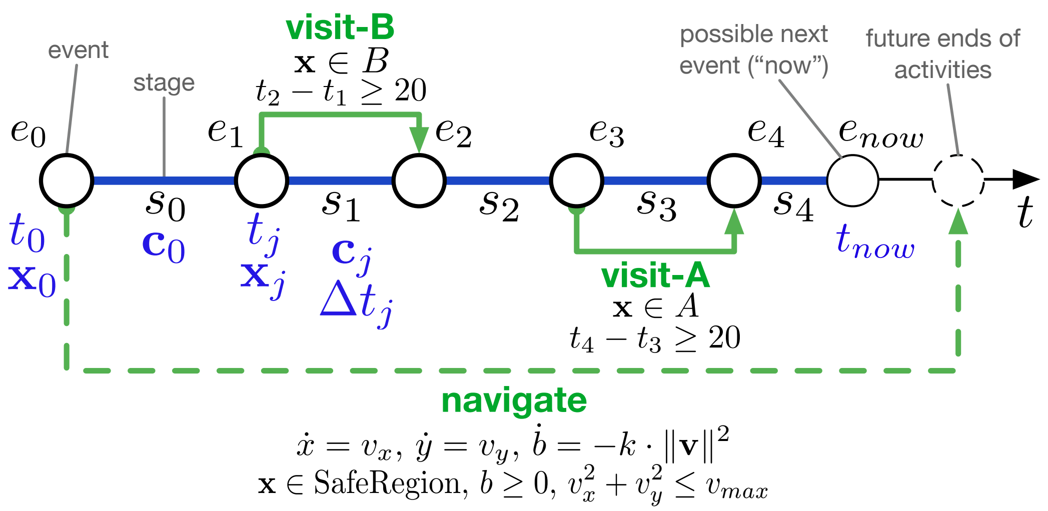


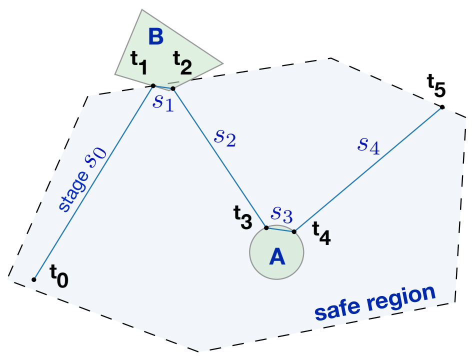
Figure 6-1: Example plan skeleton when the start of *navigate*, the start and end of *visit-A* and the start and end of *visit-B* events have been added but the end of *navigate* is not part of the plan yet.

obstacles. In this case, stages still describe periods of time when the control variables vector takes a constant value. However, ScottyPath additionally constrains vehicles to remain inside some safe region for the duration of each stage. This is described in detail in Chapter 11.

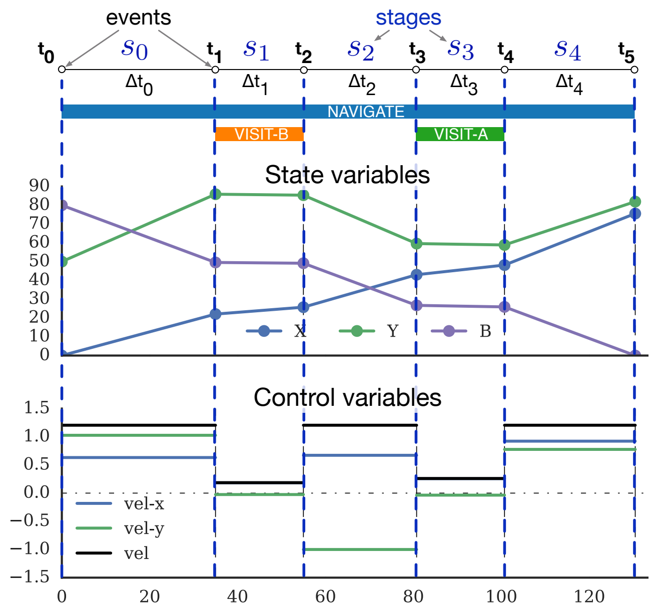
The *at start*, *at end* and *over all* state conditions for the *hybrid activities* fully determine the state conditions that need to be satisfied at every event and during each stage. Similarly, the continuous effects of each activity describe how the state variables can change during each stage in the skeleton plan. Figure 6-1 shows an example plan skeleton for an example problem that we describe next. This example is used throughout the rest of this chapter to describe our convex model.

In our example problem, a vehicle has to visit regions A and B and stay inside them for at least 20 seconds. While doing so, it must always stay inside the safe region and travel with a maximum velocity of 2 m/s. The vehicle has a limited battery that decreases at a rate proportional to the square of the norm of its velocity. For pedagogical purposes, the objective function in this problem is a linear combination of terms that aim to minimize the total plan makespan and to maximize the x and y position of the vehicle at the end of the plan.

This example problem is modeled with three activities: *navigate*, *visit-A* and



(a)



(b)

Figure 6-2: Resulting state and control trajectories for the example problem. The optimization chooses the *switch points* in order to satisfy the constraints and obtain the best objective of maximum x and y with minimum time.

visit-B. Assume, for clarity, that the ScottyActivity search has already found a plan skeleton with 5 events, in which the start of *navigate* is the first event and the start and end of *visit-B* and *visit-A* happen sequentially while *navigate* takes place. Assume, however, that the end of *navigate* has not been placed in the plan yet. The corresponding plan skeleton is shown in Figure 6-1.

The solution to this example problem, once the search is completed and the final *end-navigate* event is placed, is shown in Figure 6-2. Note that our planner automatically chooses the intermediate points and the times of the vehicle state and control trajectory in order to respect the constraints and minimize the objective. In particular note that while the objective includes minimizing the total time, the planner chooses very small velocities during the *visit-B* and *visit-A* activities. The reason for this apparent contradiction is that these activities force the vehicle to remain inside their regions for at least 20 seconds. Since there is no point in leaving these regions early, the planner decides to move very slowly since large speeds decrease the vehicle battery significantly, and this battery is needed to move in the x and y coordinates at the end of the plan. The solution shown is guaranteed to be optimal with respect to the given choice of activities. Note, however, that if ScottyActivity had chosen to visit A before B the solution returned would be very different and the objective reached would be worse.

We describe throughout the rest of this chapter how our convex model is built.

6.2 Preliminary Definitions and Decision Variables

Recall that the control variables take constant values during stages and, as a consequence, the state variables change linearly in time during these stages. Recall, as well, that given the event execution times and the piecewise constant control trajectory, the state trajectory is fully determined.

A plan skeleton with N events, $e_0, \dots e_{N-1}$, has $N-1$ stages, labeled $s_0, \dots, s_{N-2}$. We call stage j, s_j, the one starting at event j and ending at event $j+1$. Execution times and state variable values are associated to events, while the constant values of the control variable trajectory are associated to stages.

The execution times of the events, $t_j \in \mathbb{R}$, are decision variables in our model. The values of the state variables at each event, $\mathbf{x}_j = \mathbf{x}(t_j) \in \mathbb{R}^n$ are also decision variables in our model. While we need to determine the constant values of the control variables during each stage, $\mathbf{c}_j \in \mathbb{R}^m$, these are not decision variables in our model for reasons that we explain later in this chapter. Instead, we use other proxy decision variables that we define later and whose value we use to compute the values of the control variables once the optimization problem is solved. For the sake of clarity in our model, we also define the duration of each stage, Δt_j, as a decision variable whose value is constrained to be $\Delta t_j = t_{j+1} - t_j$. Other auxiliary decision variables are defined for different purposes, and are introduced as needed in this section.

The execution time of the first event and the values of the state variables at that point are constrained to take the initial conditions defined in the PDDL-S problem. In our example, $t_0 = 0, x_0 = 0, y_0 = 50, \text{battery}_0 = 80$.

6.3 Temporal Constraints

Events in the plan skeleton are totally ordered. We enforce this order by constraining the events to be separated by at least ε time ($\Delta t_j \geq \varepsilon$). This is consistent with the PDDL2.1 semantics [39]. We do this because solvers can only enforce non-strict inequality constraints. For each event representing the end of an activity, the constraint that the minimum and maximum duration of that activity is respected is added to the program:

$$d_l(a) \leq t_e(a) - t_s(a) \leq d_u(a), \quad \forall a \in P \tag{6.1}$$

, where $t_s(a)$ and $t_e(a)$ are the event times of the start and end of activity a, $d_l(a)$ and $d_u(a)$ are the lower and upper bounds of the duration and each a is an activity in the plan skeleton.

In the example, the vehicle has to remain in regions A and B for at least 20 seconds. Therefore, the constraints $t_2 - t_1 \geq 20$ and $t_4 - t_3 \geq 20$ are added to the program.

6.4 State Constraints

Each event is the start or end of an activity. For each event, e_j, which is the start (end) of an activity, we add the constraints that enforce that the value of the state variables, $\mathbf{x}_j$, satisfy the *at start* (*at end*) state conditions of the activity.

Moreover, the values of the state variables at event e_j, $\mathbf{x}_j$, need to satisfy all the maintain (*over all*) state conditions of all the activities that have started before event e_j but have not ended yet. Recall from Chapter 5 that, since state conditions are restricted to be convex and the state trajectories are piecewise linear, enforcing the maintain conditions of an activity at all events that take place during the activity execution ensures that the conditions are always satisfied at any continuous point in time during the activity.

In the example, the *visit-B* activity requires that the vehicle remains in region B while the activity takes place. Since B is a polygonal region, this is achieved with the constraint $H(x_j\, y_j) \leq \mathbf{h}$, $j \in \{1,2\}$, where $H \in \mathbb{R}^{2\times 2}$ and $\mathbf{h} \in \mathbb{R}^{2\times 1}$ are the constants of the linear inequalities that represent the polygon. The conditions for the *visit-A* activity are similar, except that the region is a circle and the constraints take the quadratic form of $(x_j - c_x)^2 + (y_j - c_y)^2 \leq R^2$, $j \in \{3,4\}$. Moreover, since events e_1, e_2, e_3 and e_4 take place while the *navigate* activity is ongoing, the state variables at those events need to satisfy its *over all* conditions: remain inside the safe region and keep the battery level above 0.

6.5 State Change

State variables change their value between consecutive events e_j and e_{j+1} due to the presence of continuous effects active during the stage between those events, s_j. Therefore, the value of state variable k at event $j+1$ is given by:

$$x_k(t_{j+1}) = x_k(t_j) + \sum_{eff_i \in E_{x_k}(j)} \Delta x_{k eff_i}(j) \tag{6.2}$$

, where $E_{x_k}(j)$ is the set of all active continuous effects operating on state variable x_k at stage j. Each $\Delta x_{k eff_i}(j)$ is a decision variable describing the change due to a continuous effect during stage j. Each type of continuous effect imposes different constraints on this value, as we describe in the next section.

6.6 Control Variables and Continuous Effects

Recall, from Definition 4.9, that the change produced by a *controllable linear time-varying continuous effect (CLTE)* on state variable x_k during stage j is described by:

$$\Delta x_{kCLTE}(j) = \left(\sum_i k_i \cdot c_i(j) \right) \cdot (t_{j+1} - t_j) \tag{6.3}$$

, where $c_i(j)$ is the constant value that the i-th control variable takes during stage j. Given that the event times are decision variables, the previous equation is non-linear and non-convex if the control variables are also decision variables. To overcome this problem, the first version of ScottyActivity in [36] relaxes this non-linear constraint to a linear interval equation on the state variables. With this relaxation, control variables are not needed as decision variables and, instead, its lower and upper bounds are used in the interval equations. Using this relaxation, however, limits the problems that can be solved. For instance, with this relaxation the same control variable cannot be used in more than one effect simultaneously, since the interval equations would, in practice, allow the solver to choose two different values for the same control variable (e.g. a high velocity to make a vehicle drive fast and simultaneously a low value to make it consume less battery). Another important limitation of this relaxation is that control variables are completely independent of each other and they can only be limited by fixed bounds. Other constraints on the control variables are not possible within this framework. For instance, a moving vehicle would only have independent bounded velocity control variables on x and y. The magnitude of its velocity cannot be limited, as this requires a norm constraint acting on both the v_x and v_y velocities. Therefore, the vehicle would essentially be able to move much faster in diagonal directions than in the x and y directions.

These limitations are addressed here with a different encoding. Instead of using interval equations, we define the decision variable $c_{i\Delta t_j}$ for every control variable being used in each stage j. This decision variable represents the value $c_{i\Delta t_j} = c_i \cdot \Delta t_j$. However this constraint is not added directly, to avoid adding a non-linearity and non-convexity constraint. Instead, the new decision variable $c_{i\Delta t_j}$ is subject to the linear inequality constraints

$$c_{il} \cdot \Delta t_j \leq c_{i\Delta t_j} \leq c_{iu} \cdot \Delta t_j \tag{6.4}$$

where c_{il} and c_{iu} are the constant lower and upper bounds of control variable c_i. We can use this new decision variables to turn (6.3) into a linear equation, as we describe later.

Note that, in effect, instead of asking the solver to pick a value for each control variable, we ask the solver to pick the times of the events and the product of the values of the control variables and the elapsed time between consecutive events. The key idea that makes this encoding work is that, for most purposes, we only need the values of the products of control variables and time intervals instead of the actual values of the control variables. A key advantage of this formulation is that, contrary to the first version of ScottyActivity, control variables can now be used in as many continuous effects as needed, since the $c_{i\Delta t}$ decision variables can appear in other constraints, and the values will be consistent with each other. Moreover, we can now impose constraints on the control variables. For example, we can impose that two control variables, c_1, c_2 always satisfy $c_1 + c_2 \leq 5$. The constraint cannot be encoded directly since c_1 and c_2 are not explicit decision variables. However, we can multiply the equation by Δt_j and impose the condition $c_{1\Delta t_j} + c_{2\Delta t_j} \leq 5 \cdot \Delta t_j$ in every stage in which they are active.

Similarly, we can also impose maximum l_2-norm constraints on sets of control variables. In many robotic applications, like the motivating example in Section 3.1, it is useful to represent the velocity of a robot with its principal components (v_x, v_y) but still limit the total velocity by some fixed amount. This can be expressed with the constraint $||\mathbf{c}|| \leq v_{max}$, where $\mathbf{c} = [v_x, v_y]^T$ is a vector of control variables. To encode this constraint, we multiply the whole equation again by Δt_j to obtain the equation

$$\|\mathbf{c}\| \cdot \Delta t_j = \left\|\mathbf{c}_{\mathbf{\Delta t_j}}\right\| \leq v_{max} \cdot \Delta t_j \tag{6.5}$$

The previous inequality is not a linear constraint, but a particular case of a second order cone constraint. Second order cone constraints are given by:

$$\|\mathbf{A}\mathbf{x} + \mathbf{b}\| \leq \mathbf{c}^T\mathbf{x} + \mathbf{d} \tag{6.6}$$

where $\mathbf{x}$ is the vector of decision variables and the rest are constant parameters. Since $\mathbf{c}_{\mathbf{\Delta t}} = [v_{x\Delta t}, v_{y\Delta t}]^T$ and Δt are decision variables, we can transform (6.5) into (6.6) by taking $\mathbf{b} = \mathbf{d} = \mathbf{0}$ and choosing constants $\mathbf{A}$ and $\mathbf{c}$ as needed.

In the example, the velocities of the vehicle, v_x and v_y are each restricted to be within $(-2, 2)$. Therefore, we define the variables $v_{x\Delta t_j}$ and $v_{y\Delta t_j}$ for each stage and constraint them as:

$$-2 \cdot \Delta t_j \leq v_{x\Delta t_j} \leq 2 \cdot \Delta t_j \tag{6.7}$$

$$-2 \cdot \Delta t_j \leq v_{y\Delta t_j} \leq 2 \cdot \Delta t_j \tag{6.8}$$

$$\tag{6.9}$$

Moreover, the magnitude of the velocity of the vehicle is also constrained to not exceed 2. This is achieved with the following cone constraint for each stage:

$$v_{x\Delta t_j}{}^2 + v_{y\Delta t_j}{}^2 \leq 2^2 \cdot \Delta t_j{}^2 \tag{6.10}$$

6.6.1 CLTE Effects

Using the newly defined $c_{i\Delta t_j}$ variables, the originally non-linear CLTE equation (6.3) can then be rewritten as the linear equation

$$\Delta x_{kCLTE}(j) = \sum_i k_i \cdot c_{i\Delta t_j} \tag{6.11}$$

As an example, consider that the change in state variable x during stage s_1 (during the *visit-B* activity) in the example scenario is given by:

$$x_2 = x_1 + v_{x\Delta t_1} \tag{6.12}$$

6.6.2 RNE Effects

The second type of continuous effects, *resource constrained norm effects (RNE)*, are described with cone constraints. In this work we focus on *linear norm effects* (LNE) and *linear squared norm effects* (LSNE). Recall, from Section 4.1.4.2, that the change due to a LNE effect on a constrained resource (state variable r_k) is given by

$$\Delta r_{kLNE}(j) = -k_{LNE} \cdot \|\mathbf{c}_e\| \cdot \Delta t_j, \quad k_{LNE} \geq 0 \tag{6.13}$$

where k_{LNE} is a non-negative real constant and $\mathbf{c}_e$ is the vector of the control variables involved in the effect. Equation (6.13) is not convex and cannot be encoded directly. However, we can transform (6.13) into a cone constraint by defining a new decision variable b that acts as a bound. The equation is then rewritten as

$$\|\mathbf{c}_{\mathbf{\Delta t_j}}\| \leq b, \quad b \geq 0 \tag{6.14}$$

$$\Delta r_{kLNE}(j) = -k_{LNE} \cdot b \tag{6.15}$$

Equations (6.13) and (6.15) do not represent the same, since b is simply an upper-bound on $||\mathbf{c}_{\mathbf{\Delta t_j}}||$. This is the reason why we have to restrict these effects to constrained resources. In general, the bound b will not be tight and the computed value for resource r_k may not be accurate. However, these equations can accurately model that a constrained resource decreases with the norm of a control variable vector and the bound will become tight to ensure that a resource never dips below some threshold. This is useful to model, for example, how a vehicle's battery decreases as a function of the speed it is traveling at, regardless of the x, y direction. Unfortunately, we cannot use resource constrained norm effects to impose that a certain resource is below some level, since the artificial bound b could take any arbitrary large value to satisfy the constraint trivially without changing the actual value of the norm of the control variable vector. Modeling such condition would require handling a constraint in the form of $||\mathbf{c}_{\mathbf{\Delta t_j}}|| \geq b$. However, this is a non-convex constraint that we do not support since it cannot be represented with a SOCP program. In practice, we have not found many problems in which this is required, but we leave this extension for future work.

The second resource constrained norm effect that we support is the *linear squared norm effect* (LSNE), which is subject to the same limitations as the LNE effect, but in which the decrease rate of the resource variable is proportional to the squared norm:

$$\Delta r_{kLSNE}(j) = -k_{LSNE} \cdot \|\mathbf{c}\|^2 \cdot \Delta t_j, \quad k_{LSNE} \geq 0 \tag{6.16}$$

Again, equation (6.16) is non-convex, but we can use the same principle as before to represent it as a SOCP constraint. Since $\mathbf{c_{\Delta t_j}} = \mathbf{c} \cdot \Delta t_j$, we can write

$$\|\mathbf{c}\|^2 \cdot \Delta t_j = \frac{\mathbf{c_{\Delta t_j}}^T \mathbf{c_{\Delta t_j}}}{\Delta t_j} \leq b, \quad b \geq 0 \tag{6.17}$$

where b is, again, an auxiliary positive bound variable. Finally, equation (6.17) can be rewritten as

$$\mathbf{c_{\Delta t_j}}^T \mathbf{c_{\Delta t_j}} \leq b \cdot \Delta t_j \tag{6.18}$$

$$\Delta r_{kLSNE}(j) = -k_{LSNE} \cdot b \tag{6.19}$$

which is a *rotated cone constraint*, a valid type of convex SOCP constraint, since b and Δt_j are positive.

In the example, the vehicle battery decreases with a rate proportional to the square of the vehicle velocity. This is modeled with the *LSNE* effect present in the *navigate* activity. As explained in the previous section, we use auxiliary decision variables to model this change. For each stage, we define the upper bound decision variables $u_{LSNE}(j)$ for each stage that are subject to the rotated cone constraint described in eq. (6.17). In particular, the constraint takes the form:

$$\mathbf{v_{\Delta t_j}}^T \mathbf{v_{\Delta t_j}} \leq u_{LSNE}(j) \cdot \Delta t_j, \quad u_{LSNE}(j) \geq 0 \tag{6.20}$$

where $\mathbf{v_{\Delta t_j}} = (v_{x\Delta t_j}\ v_{y\Delta t_j})$. Then, the battery is updated in each stage according to $b_{j+1} = b_j - k_{LSNE} \cdot u_{LSNE}(j)$. Although the battery value computed with this model may not be accurate (since the upper-bound decision variables could take arbitrarily large values), these constraints ensure that the battery is never depleted.

6.7 Partial Skeleton Plans

The constraints presented so far are sufficient to represent full plans. However, ScottyActivity also needs to test the consistency of partial skeleton plans. In these partial plans there may be ongoing activities that have started but not ended yet. We use the same t_{now} trick that COLIN [23] uses to detect inconsistencies due to future temporal constraints being violated. An event at t_{now} is placed after all the other events in the partial plan. The state variables at this event are updated according to the active continuous effects and subject to the invariant conditions of activities that have started but not ended yet. For each of these ongoing activities, a decision variable describing the future time where it will end is added and constrained to occur after t_{now} and to respect the activity duration constraints. This helps identify partial plans that are not feasible because future temporal deadlines cannot be met. The 'now' event is also the point at which each state variable is minimized and maximized to find the bounds of each state variable at the end of the given plan skeleton.

In the example problem activity *navigate* has not ended yet (Figure 6-1). Therefore, we place the auxiliary 'now' event, e_{now}, after the last event in the skeleton (the end of *visit-A*) but before the future end of the *navigate* activity.

6.8 Objective

Recall that ScottyActivity and ScottyPath use the convex model for multiple purposes during planning. Different purposes require different objective functions. The objective function that ScottyPath uses is significantly different from that of ScottyActivity, since the model is used to jointly compute the cost and the heuristic of a search node. For this reason, we defer the description of the objective function that ScottyPath uses for Section 11.4.3. In this section we describe the different objective functions that ScottyActivity defines for each of the purposes for which the convex model is used.

When the model is used to test the feasibility of a plan skeleton, no objective is needed. The model is also used to compute the bound of each state variable at the end of a plan skeleton, as we describe in Chapter 8. In that case, the problem is solved $2n$ times to minimize and maximize each of the n state variables at the end of the plan. The $2n$ objectives are, therefore: $\{x_k(t_{now}), -x_k(t_{now})\}$ [1].

ScottyActivity encodes the PDDL-S problem objective in the model in two situations: when determining the accumulated cost of a plan skeleton if the obj-EHC algorithm is being used, and when finding the optimal execution times and control trajectory for the full plan skeleton that satisfies the goal discrete conditions. Recall, from Section 4.1.5, that the PDDL-S objective is a linear combination of the total plan time, the values of the state variables at the end and the sumproducts of the norms (or squared norms) of vectors of control variables and the durations they are active for. The first two are straightforward to model, since the total plan time and the values of the state variables are, respectively, the execution time of the last event in the plan and the values of the state variables at that time. The last option allows us to minimize actuation control.

The sumproduct of the norm of a control variable vector $\mathbf{c}_v$ and the times it is active for is given by

$$\sum_j \|\mathbf{c}_v\| \cdot \Delta t_j \tag{6.21}$$

Note that if $\mathbf{c}_v$ represents the velocity of a vehicle, the previous is equivalent to the distance traveled by that vehicle. In order to minimize eq. (6.21) using our convex model, we use the same approach we followed to represent *LNE* effects (eq. (6.14)). For each stage, we define a bound decision variable b_j that satisfies

$$||\mathbf{c}_{\mathbf{\Delta t_j}}|| \leq b_j, \quad b_j \geq 0 \tag{6.22}$$

Then, minimizing $\sum_j \|\mathbf{c}_v\| \cdot \Delta t_j$ is equivalent to minimizing the sum of the bound variables for each stage ($\sum_j b_j$). We use an analogous approach to minimize the

[1]For valid states, the bounds for the state variables always need to be computed after the feasibility check. Therefore, for efficiency purposes, our model is never used without an objective in practice. The first model solved is simultaneously determining the feasibility of the plan and minimizing the first state variable.

sumproduct of the squared norm. In this case the bound variables are constrained with rotated second order cone constraints (as in the case of the *LSNE* effects):

$$\mathbf{c_{\Delta t_j}}^T \mathbf{c_{\Delta t_j}} \leq b_j \cdot \Delta t_j, \quad b_j \geq 0 \tag{6.23}$$

In the example problem, the objective is a combination of minimizing the total time and maximizing the sum of the x and y coordinates at the end. For the full plan as shown in Figure 6-2a, this is achieved with the minimization objective $0.7t_5 - x_5 - y_5$.

Part II

ScottyActivity

Table of Contents

Chapter 7

Expressing ScottyActivity PDDL-S Problems

ScottyActivity uses control variables to define continuous change and allows more complex state constraints than most activity planners. Since these features cannot be expressed with standard PDDL, we define an additional syntax that we describe in this chapter. We base our syntax in PDDL2.1 [39] since it provides most of the expressiveness that we need, except for control variables. While PDDL+ [40] significantly advances the expressiveness of PDDL2.1, its syntax does not define control variables (or parameters) and its more advanced features, such as processes and events, are not supported by our planner.

We now proceed to describe the additions to the PDDL syntax that we need to express PDDL-S problems. The rest of the PDDL2.1 syntax is the same.

7.1 Control variables and global constraints on control variables

As explained in Section 4.1, each control variable $c_i \in CV$ is defined by its lower (c_{il}) and upper (c_{iu}) bounds. We use the following syntax, that needs to precede the definition of activities, to define a control variable.

```
(:control-variable <cvar-name>
 :bounds (and (>= ?value <lower-bound>)
              (<= ?value <upper-bound>)))
```

The global constraints on control variables can be of two types: linear constraints and norm constraints. We use the keyword `control-constraint` to define named groups of linear control variable constraints such as:

```
(:control-constraint <constraint-name>
 :condition (and ({<=, >=} <lin-expr-left> <lin-expr-right>)
                                           ...))
```

, where `<lin-expr-{left,right}>` are linear expressions on the control variables $c_i \in CV$.

Norm constraints on control variables are expressed by first defining vectors of control variables. Control variable vectors are lists of control variables that can optionally specify a maximum norm according to the following syntax:

```
(:control-variable-vector <vector-name>
 :control-variables ((<cv1> <cv2> ... <cvn>))
 :max-norm <max-norm-value>)
```

In a later section of this chapter, we use vectors of control variables, as defined above, to express RNE effects, that depend on norms of vectors of control variables.

7.2 Continuous change with CLTE and RNE effects

CLTE effects are effects in which the rate of change is expressed as a linear combination of control variables. We use a syntax very similar to that of PDDL2.1 for continuous effects with the difference that, instead of using constant rates of change, we use control variables. In particular, a continuous effect is defined in an activity with the following syntax:

```
({increase,decrease} <state-var> <cvar-time-expr>)
```

, where `<cvar-time-expr>` is a linear combination of control variables multiplied by the `#t` term. As an example (**increase** (x) (∗ (vx) #**t**)) denotes that state variable x is subject to a rate of change that corresponds to the control variable v_x.

RNE effects, that depend on the norm of control variable vectors, are defined similarly by using the keywords `norm` or `norm-sq` to denote the norm or squared norm of control variable vectors. For example (**decrease** (b) (∗ 0.45 (**norm** (velocity) #**t**))) represents that the state variable b is subject to a rate of change of $-0.45\|\mathbf{velocity}\|$, where `velocity` is a vector of control variables as defined above.

7.3 Objectives

Similarly to PDDL2.1, the optional objective of a planning problem is defined with the keyword `metric`, according to the following syntax:

```
(:metric minimize <linear-expr>)
```

As described in Section 4.1.5, the minimization objective is a linear combination of the following types of terms:

- State variables at the end of the plan ($x_i \in V$)
- Plan makespan (using the keyword `total-time`)
- Sum-products of the norms or squared norms of control variable vectors and the durations they are active for. Since control variables are restricted to be piecewise constant, the minimization objective given by the expression `norm(velocity)`, for example, minimizes the sum of the norm of the velocity control variable vector in each segment multiplied by the duration of that segment ($\sum_j \|\mathbf{v}(j)\| \cdot \Delta t_j$). In the case of a velocity, this is equivalent to minimizing the distance traveled by a vehicle. However, a similar term involving the squared norm of a control variable vector is also possible.

7.4 Representing Advanced Convex State Constraints Through State Space Regions

PDDL2.1 syntax allows activity conditions on state variables to be defined using polynomial expressions. Since Scotty only supports convex quadratic state conditions, polynomial expressions are able to represent all the conditions that we can handle. However, manually specifying state conditions using polynomial equations is cumbersome and repetitive, since the same conditions often need to be used in different parts of the domain definition. Moreover, the convex quadratic conditions that Scotty can handle can become fairly complicated and hard to write using direct equations. Polynomial state constraints are still allowed in Scotty. However, for the reasons outlined above, we have developed a region-based system to express state conditions that arise naturally in robotic scenarios in a more succinct and clear way.

We call a *region* the part of the state-space that is allowed by a condition. Regions are defined as a collection of *primitive regions* defined on one or more *parameters*. Analogous to the parameters in PDDL activities, the region parameters are placeholders that are replaced by expressions of state variables when the regions are used in actual activity conditions. Regions are defined before activities with the following syntax:

```
(:region <region-name>
 :parameters (?par1 ?par2 ...)
 :condition (and <primitive-region1(parameter-expr1)>
                 <primitive-region2(parameter-expr2)>
                 ...                                    )
 [:linear-approximation (and ({<=,=,>=} <lin-exp-left> <lin-exp-right>)
                             ...)])
```

Regions defined in this way represent the intersection of the provided primitive regions. Since the primitive regions are convex, their intersection is also convex. We only support regions defined as intersections (`and`) and not disjunctions (`or`) because the latter are not guaranteed to be convex. Scotty supports both linear conditions as well as quadratic conditions, as long as they are convex. For reasons that are explained in Chapter 8, the heuristic needs linear over-approximations for quadratic conditions. Our quadratic region primitives, such as circles, automatically add these linear over-approximations (such as surrounding boxes). In general, automatic linear over-approximations cannot be deduced easily from arbitrary quadratic equations. Therefore, users specifying conditions manually using quadratic equations need to provide a list of linear inequalities that over approximate the quadratic region using the optional parameter `linear-approximation`.

Regions defined in this way can then be used in the *at start*, *at end* or *over all* conditions of activities. Region conditions are defined with the following syntax:

```
(inside (<region-name> <par-expr1(state-variables)>
                       <par-expr2(state-variables)
                        ...                            ))
```

Note that we only support `inside` and not `outside` conditions. In effect, being inside a convex region constitutes a convex condition, while being outside a convex region does not. The quadratic solvers that Scotty uses do not support non-convex conditions. However, even if they did, we would not be able to guarantee that invariant conditions are satisfied throughout a piecewise linear trajectory by only checking the switch points (like we showed we can do with convex conditions in Chapter 5).
Regions used in conditions need to be supplied with a list containing the values to be used in place of their parameters. Each parameter can be given as a state variable, or as an expression involving one or more state variables. This provides additional flexibility to compose complicated regions from simpler ones. As an example, let us consider a sampling region defined by a rectangle.

```
(:region sampling-region
  :parameters (?x ?y)
  :condition (and (in-rect (?x ?y) :corner (5 -10) :width 20 :height 30)
   ))
```

The region `sampling-region(?x, ?y)` has parameters `?x` and `?y`. Note that these parameters are not state variables, but simply placeholders that serve to indicate what the internal primitive region `in-rect` should be defined with respect to. The primitive region `in-rect(?x, ?y)` defines the rectangular 2d region given by the inequalities $\text{corner}_x \leq ?x \leq \text{corner}_x + \text{width}$ and $\text{corner}_y \leq ?y \leq \text{corner}_y + \text{height}$. The `sampling-region` can now be used to indicate, for example, that an AUV needs to remain inside the region while a sample is being taken:

```
(over all (inside (sampling-region (x-auv) (y-auv))))
```

, where `x-auv` and `y-auv` are state variables that denote the position of the AUV. Note that `sampling-region` can be reused in a different activity condition just by feeding a different set of parameters as required. For example, we could indicate that the ship needs to remain in the same region while another activity is being executing just by using `(inside (sampling-region (x-ship) (y-ship)))`. The expressions passed as parameters to the regions can be arbitrarily complex, which makes it very simple to specify complex conditions. For example, we could impose that instead of requiring the AUV or the ship to be inside the region, we would like the center point of the two vehicles to remain in the region by using the following expression:

```
(over all (inside (sampling-region (/ (+ (x-auv) (x-ship)) 2)
                                   (/ (+ (y-auv) (y-ship)) 2))))
```

The internal expression engine in Scotty propagates the parameter expressions in order to obtain the resulting linear conditions $\text{corner}_x \leq \frac{1}{2}(x_{AUV} + x_{ship}) \leq \text{corner}_x + \text{width}$ and $\text{corner}_y \leq \frac{1}{2}(y_{AUV} + y_{ship}) \leq \text{corner}_y + \text{height}$. Expressions passed as parameters to regions can be arbitrarily complex as long as the resulting final expression remains convex quadratic, due to the reasons explained before.

7.4.1 Primitive Regions

In the current version of our planner we have implemented multiple *primitive regions* such as `in-rect`. Since most of the robotic problems where ScottyActivity has been used can be expressed with two dimensional coordinates, the primitives shown here represent 2D conditions. However, our system is not restricted to two dimensions, and additional region primitives can be defined for any number of dimensions.

- Manual convex quadratic inequalities or linear equalities: `({<=,=,>=} <expr-left`

> <expr−right>)

- Rectangles: (**in−rect** (?x ?y) **:corner** (<c_x> <c_y>) **:width** <w> **:height** <h>)

  The rectangle primitive enforces the conditions $c_x \leq ?x \leq c_x + w$ and $c_y \leq ?y \leq c_y + h$. Since these conditions are linear, this primitive does not need to add linear approximations.

- Convex polygons: (**in−poly** (?x ?y) **:vertices** ((<v_{1x}> <v_{1y}>) ... (<v_{nx}> <v_{ny}>)))

  The convex polygon is given by a list of its vertices. Again, no linear approximation equations are needed in this case.

- Circles: (**in−circle** (?x ?y) **:center** (<c_x> <c_y>) **:r** <r>)

  The circle primitive enforces the convex quadratic condition $(?x - c_x)^2 + (?y - c_y)^2 \leq r^2$. This primitive also adds the linear over-approximation conditions of an square of center (c_x, c_y) and side $2r$ surrounding the circle.

- Maximum distance between entities: (**max−distance** ((?x1 ?y1) (?x2 ?y2)) **:d** <d>)

  This primitive ensures that point entities of positions $(?x1, ?y1)$ and $(?x2, ?y2)$ always remain within distance d by enforcing the quadratic condition $(?x1 - ?x2)^2 + (?y1 - ?y2)^2 \leq d^2$. This primitive also provides the linear over-approximation given by the equations $|?x1 - ?x2| \leq d$ and $|?y1 - ?y2| \leq d$.

  Note that, thanks to our powerful parameter expression system, the condition represented by `max-distance` can also be represented with a `circle` region in which the parameters are the x,y components of the difference vector between the entities:

```
(:region max−distance−region
         :parameters (?x1 ?y1 ?x2 ?y2)
         :condition (in−circle (− ?x1 ?x2) (− ?y1 ?y1)
         :center (0 0) :r <d>))
```

- Other previously defined regions: (**in−region** <region−name> (<par−expr1> ...))

  A region can also be composed by the intersection of previously defined regions.

This makes it easy to define complex regions from other simpler, reusable regions.

Chapter 8

ScottyActivity: Joint Activity and Trajectory Planning with Heuristic Forward Search

In this chapter we describe ScottyActivity, the component that performs activity planning in the Scotty Planning System. ScottyActivity heavily relies on ScottyConvexPath, which is described in Part I of this thesis. We begin by providing a high level overview of the planner. We then describe the generation of successor search states, the heuristic and the algorithms that guide the search.

8.1 ScottyActivity In a Nutshell

Recall that the plan that ScottyActivity generates consists both of an activity schedule and a control plan, given as a piecewise constant trajectory of the control variables. ScottyActivity generates the activity schedule and the control plan concurrently, through heuristic forward search to select the order of the activities as a plan skeleton, and ScottyConvexPath to test the feasibility of the plan skeleton. ScottyConvexPath uses convex optimization to test the feasibility of the plan skeleton by finding a valid control trajectory, state trajectory and the execution times of the activities. An informal diagram describing how the ScottyActivity planner works is

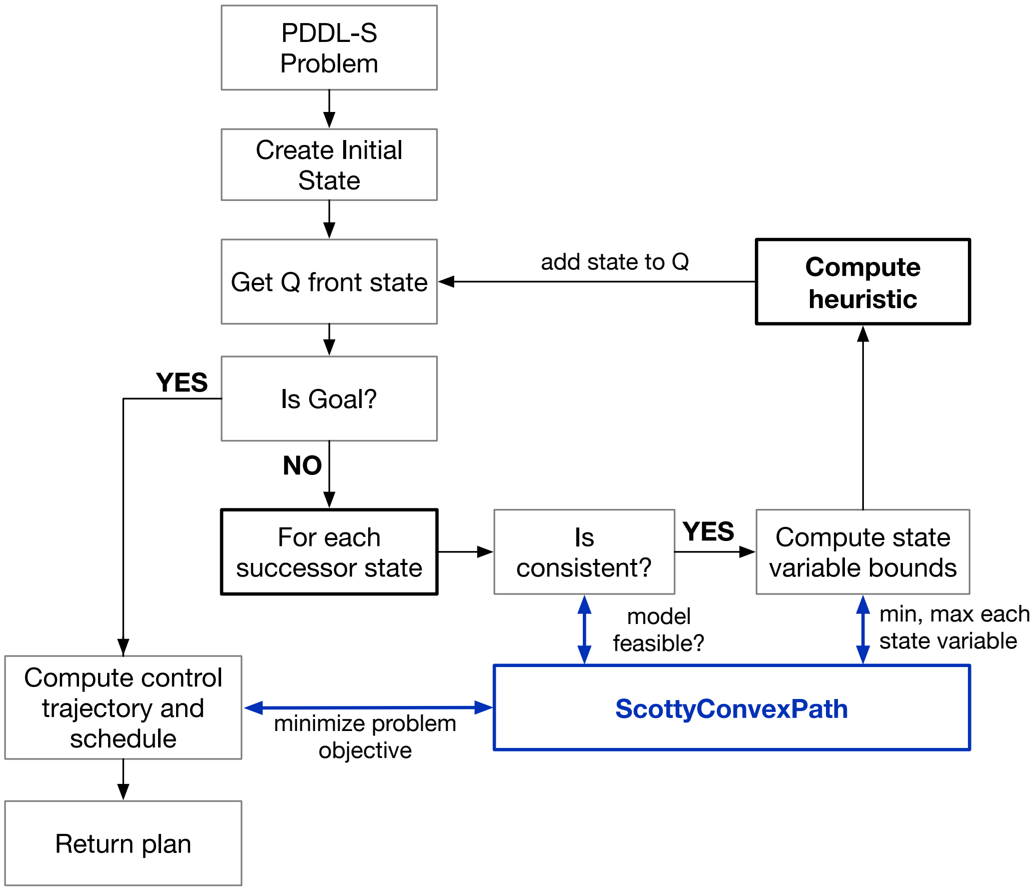


Figure 8-1: Informal diagram describing the overall flow of the ScottyActivity planner. The blue box indicates that ScottyConvexPath is used as a module that is queried at different stages of the planning process and is not part of the flow.

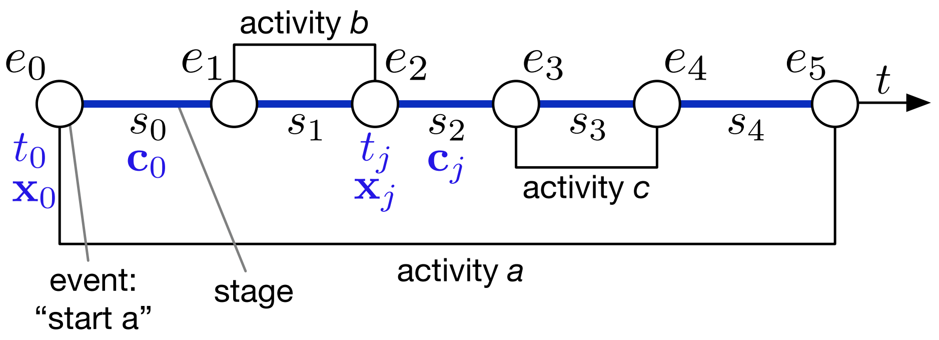


Figure 8-2: A plan skeleton is given by an ordered sequence of start and end events, e_j. Plan skeletons do not have an assigned control or state trajectory, or event execution times. A mathematical program is used to test the consistency of plan skeletons by finding feasible values for the control trajectory $\mathbf{c}(t)$, state trajectory $\mathbf{x}(t)$ and event execution times, t_j.

presented in Figure 8-1. We provide detailed descriptions of each step of the algorithm in this section.

Our method draws inspiration from COLIN [23], LPGP [69], LPSAT [96] and previous planners [91] in that the discrete search is interleaved with consistency checks using an optimization approach. We use heuristic forward search to find an ordered sequence of starts and ends of activities that are analogous to the start and end snap actions used by many temporal planners [69, 24]. We call each start or end of an activity an *event*. We assume that no two events can co-occur, and that they are totally ordered. This allows the planner to consider the introduction of only one event during each planning step. While some heuristic forward search planners allow some flexibility in the order of these events [22], we leave this extension for future work.

We call a *stage* the period of time between consecutive events. In the piecewise constant control trajectory that ScottyActivity finds, control variables have constant values during stages, and the value changes can happen at each event. As described in Chapter 5, the state variables change linearly in time during stages.

Recall, from Section 6.1, that every search state defines a partial schedule of totally ordered events that forms a plan skeleton. The plan skeleton only defines the sequence of starts and ends of activities that are selected and in what order. We call a *stage* the period of time between consecutive events. In the piecewise constant control trajectory that ScottyActivity finds, control variables have constant values during stages, and the value changes can happen at each event. Figure 8-2 shows an example of a plan skeleton.

Search states are constructed so that their plan skeletons satisfy all the discrete conditions imposed by the activities in the partial schedule. However, plan skeletons do not define the control trajectory, the state trajectory or the execution times of the events. In order to check that the plan skeleton can satisfy the continuous constraints, we solve an optimization problem that tries to find a feasible control trajectory, state trajectory and the execution times of the activities. For intermediate search states, this optimization problem is used as a feasibility check. The values returned by the solver are, therefore, discarded. When the search state that satisfies the goal conditions is found, the optimization problem is used one more time to find the control trajectory and activity execution times that minimize the problem objective. These values along with the event schedule are returned as the solution plan.

Successor search states extend the parent plan skeleton with a new start or end event at the end. The convex model for successor plan skeletons is solved from scratch every time, and no intermediate control trajectories are reused. There are multiple reasons why that is the case. First, we do not know what future events could be added to the plan later, and therefore we want to avoid early commitment to values that could make the plan infeasible later. Second, by not committing early to values found by the solver for incomplete plan skeletons, we can optimize the problem objective throughout the complete plan skeleton when the goal is found.

Like many other heuristic forward search planners, we use common greedy search algorithms that work well for relatively large planning problems. Our heuristic is based on the Temporal Relaxed Planning Graph [24] and it provides an estimate of the remaining number of start and end activities to reach the goal.

As described in Chapter 6, our mathematical program is convex. Therefore our consistency checks are complete. However, we use an incomplete greedy search algorithm. As is standard with most greedy HFS planners, we optionally resort to a complete A* search when the incomplete greedy search algorithms fail. In practice, A* is much slower than the greedy search algorithms and we do not report results using A* in this thesis.

ScottyActivity is not an optimal planner in that we do not guarantee that the returned plan is the best possible plan according to the problem objective. However, since our optimization problem is convex, we guarantee that the instantiation of the plan skeleton selected based on the heuristic is optimal. That is, for the order of starts and ends of activities found by the search, there exist no other control trajectories or activity execution times than the ones returned by ScottyActivity that can produce a better objective value.

In the next sections, we describe the different components of our planner in detail.

8.2 Generation of Successor States

Algorithm 8.1: GET-ACTIVITIES

```
Input: A PDDL-S planning problem ($PP$) and a search state ($S$).
Output: A list of helpful applicable activities $A_H$ and list of activities that
        are applicable but not helpful at $S$, $A_{A\setminus H}$.
Algorithm
1 $A_H \leftarrow \{\}$, $A_{A\setminus H} \leftarrow \{\}$
2 for $a$ in $PP.A$ do
3     if DISCRETE-APPLICABLE($S$, $a$) and CONTINUOUS-CONSISTENT($S$, $a$)
4         if IS-HELPFUL($S$, $a$)
5             PUSH($A_H$, $a$)
          else
6             PUSH($A_{A\setminus H}$, $a$)
7 return $A_H$, $A_{A\setminus H}$
```

Each search state contains a plan skeleton formed by an ordered list of events, a set of predicates that hold at this state, a set of activities that have started but not ended at this state, and the independently achievable lower and upper bounds for each state variable (as we explain later in this section).

In order to explore successor states, we need to determine which events (starts or

Algorithm 8.2: TEST-CONSISTENCY

```
Input: A PDDL-S planning problem ($PP$) a search state ($S$) and the partial
       plan leading to that state ($p$).
Output: A state with the independently achievable lower and upper bounds
        for each state variable or nil if the state is not consistent.
Algorithm
1 prog ← BUILD-PROGRAM($PP, p$)
2 if not IS-FEASIBLE(prog)
3     return nil
4 for $x$ in $PP.V$ do
5     $x_{min}$ ← MINIMIZE(prog, $x$)
6     $x_{max}$ ← MINIMIZE(prog, $-x$)
7     SET-BOUNDS ($S$, $x$, $x_{min}$, $x_{max}$)
8 return $S$
```

ends of activities) can be applied next. This entails checking that both the discrete and the continuous conditions and effects of the event are consistent with the current state. Since the discrete state is fully described by the search state, the discrete check can be done directly (Line 3 of Algorithm 8.1). However, we cannot do the same with the continuous conditions and effects, since the execution times, durations, state variables, and control variables are tightly coupled throughout the plan skeleton. In order to reject infeasible successor states, we construct a mathematical program containing all the constraints defined by the activities in the plan skeleton. Invalid states are those whose associated mathematical programs do not have a feasible solution (Algorithm 8.2).

As explained in Chapter 6, we use Second Order Cone Programs (SOCPs) for this check for the following reasons. First, SOCPs can effectively represent all the continuous conditions and dynamics that our planner requires (as described in Chapter 4). Second, SOCPs are a class of convex optimization problems and are commonly solved with complete algorithms that will return no solution only when the program does not have a solution. This is an important characteristic since it ensures that only infeasible states are pruned. Third, SOCPs can be solved very efficiently with interior-point methods available in off-the-shelf solvers. Finally, being convex optimization problems, the solutions of the SOCPs returned by the solvers are guaranteed to be optimal, which is important for reasons that we explain later. Gradient-based algorithms solving general non-linear programs do not present these characteristics. They are orders of magnitude slower, which would directly translate into the planning time since it is, by far, the dominant factor. They are not complete due to local minima, and not optimal. As we describe in in Chapter 6, formulating all state conditions and dynamics as a convex program is not straightforward and is one of the key innovations of our planner.

Solving the optimization for every candidate successor state that meets the discrete conditions is expensive, and dominates the running time of our planner. For this reason, we compute the feasible lower and upper bounds for each state variable independently with our mathematical program. We do that using the same method that COLIN uses: we solve the optimization problem twice per state variable to minimize and maximize the state variable at the state (Lines 4-7 in Algorithm 8.2). These bounds constitute an over-approximation of the reachable space at the last event of the plan skeleton. We use the bounds to prune activities whose state conditions are necessarily not compatible with the state variable bounds, and, therefore, can never be satisfied (`CONTINUOUS-CONSISTENT` method in Algorithm 8.1). For linear inequality conditions in the form of $\sum k_i x_i \leq c$, this can be done by computing the lower bound of the expression according to the state variable bounds, and checking whether it satisfies the condition. The lower bound on the expression can be computed with the following equation:

$$l_b = \sum \min(k_i x_{imin}, k_i x_{imax}) \tag{8.1}$$

, where k_i are the constant coefficients of the expression and x_{imin} and x_{imax} are the bounds of state variable x_i. The linear inequality can only be satisfied if $l_b \leq c$. Activities having at least one linear constraint where this condition is not met are pruned from the list of applicable activities at that point in the search. Note that, since the optimization problem is convex and its solution is optimal, the computed bounds of each state variables are guaranteed to be the maximum and minimum values that the state variable could reach. Therefore, this method only prunes infeasible activities. Note, however, that this test using the variable bounds does not guarantee that the linear condition can be satisfied in practice, as each variable lower and upper bound is computed independently. This is not a problem, since all search states that pass this pruning test are later checked with the optimization model.

Unfortunately, the pruning test described above is not straightforward in the case of general convex quadratic constraints, as it involves checking the intersection of arbitrary conic shapes (the convex quadratic constraints) and hypercubes (given by the bounds of each state variable). In practice, we use linear over-approximations for those conditions and handle them as in the linear case. This linear over-approximations can be entered manually or computed automatically using the region system described in Section 7.4. Again, using linear over-approximations only affects the efficiency of our pruning method, but not the completeness of the algorithm.

Finally, the state variable bounds are also needed to define the first layer of the heuristic, as we explain in the next section.

8.3 Relaxed Hybrid Plan Heuristic

The heuristic that ScottyActivity uses is based on the Temporal Relaxed Planning Graph (TRPG). The TRPG assigns to each layer in the planning graph a timestamp corresponding to the earliest time when each layer could be reached [32]. In order to handle continuous effects with control variables, such as vehicle dynamics, our heuristic uses two ideas. First, we use *flow tubes* to represent all possible state trajectories resulting from the application of activities with continuous effects. Second, each fact layer is annotated with the minimum and maximum values that each state variable could independently take in that layer. This idea of tracking the bounds for each state variable is borrowed from MetricFF [45].

A flow tube is a compact encoding that describes all possible trajectories resulting from the application of a continuous effect on a state variable, that is, its reachability region. Figure 8-3 shows a flow tube that represents the reachability region of state

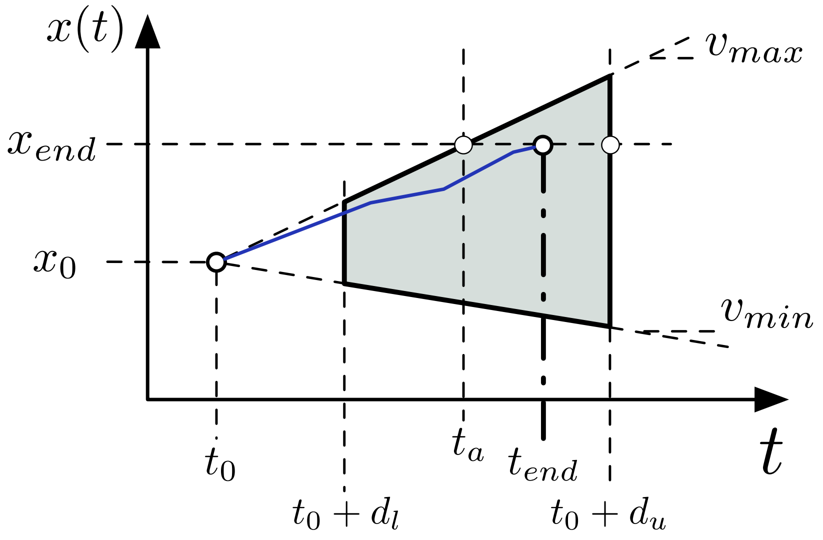


Figure 8-3: The shaded region is a flow tube that represents the reachable region for state variable x when subject to a linear time-varying effect($\Delta x(t) = v(t) \cdot t$) for a duration between d_l and d_u

variable x when subject to first order dynamics $\dot{x}(t) = v(t)$ from $x_0 = x(t_0)$ for a duration between d_l and d_u. Under first order dynamics, as is the case in Scotty problems, the velocity, $v(t)$, is a control variable that is continuously controllable within its actuation bounds of v_{min} and v_{max} for the duration of the activity. The shaded region in Figure 8-3 is the flow tube and is computed by propagating the initial point with the extremal actuation and temporal constraints. This region represents the values that x can take at the end of the activity. The blue line shows one possible trajectory. Note that the example final value, x_{end}, can be reached as soon as at t_a if the maximum velocity value v_{max} is chosen, or as late as $t_o + d_u$ if a lower value is used. Flow tubes have been used successfully for temporally and spatially flexible execution of hybrid plans, in applications such as biped walking [49, 50]. Moreover, flow tubes are the basic building block used by Kongming [64] to represent continuous behaviors in its Hybrid Flow Graph. In our heuristic, we use flow tubes to express how the reachable bounds of state variables grow between the consecutive layers of the temporally relaxed planning graph. In our case, the minimum and maximum actuation bounds are obtained by combining the minimum and maximum bounds of all the continuous effects acting on each state variable at a given time.

In practice, our heuristic works in a similar way to COLIN's [24], except for some differences that we describe in this section. As is the case for many other planners, the heuristic value is the number of start or end events needed to reach the goal in the relaxed graph. We follow the standard procedure to generate the relaxed planning graph that we summarize next. The initial fact layer is defined by the state at which the heuristic is being computed. The relaxed graph is generated by adding alternating activity and fact layers. All activities that could be applied at a given fact layer are added to the current activity layer. The procedure finishes when the goal conditions are contained in a fact layer, or when no more activities can be added to the graph. The delete relaxation procedure ensures that at the next fact layer, new activities may become applicable, but activities that were previously applicable always remain applicable in posterior layers. In order to do that, the delete relaxation only adds the discrete *add effects* of an activity to the next fact layer, but it ignores its *delete effects* [47].

In the spirit of MetricFF, the continuous conditions and effects are handled by tracking the minimum and maximum bounds of each state variable at each fact layer. These bounds are used to test whether an action is applicable at a given layer. The continuous effects active between layers grow these bounds between consecutive layers. We now describe how the bounds for each state variable are grown while creating the relaxed graph, since the presence of control variables in ScottyActivity problems requires a slightly different approach compared to COLIN. We later describe how these bounds are used to test whether a continuous condition can be satisfied at a given layer.

When computing the heuristic for a state, the bounds of all state variables at the first layer are known in advance. Recall from Section 8.2 that these bounds are computed by solving the convex optimization problem twice per state variable in order to find its minimum and maximum possible values. In order to compute the state variable bounds in the next layers, COLIN keeps track of the maximum positive and lowest negative rates of change (gradients) that each state variable could be subject to at each layer. These rates of change are additive and come from the continuous effects of ongoing activities at the current layer. The minimum and maximum possible bounds for each state variable in layer $i+1$ are computed by extending the bounds at layer i with the maximum positive and minimum negative gradients multiplied by the time elapsed between layers $i+1$ and i, $\Delta t = t_{i+1} - t_i$. In the spirit of delete relaxations, the bounds can only grow from one layer to the next. Contrary to COLIN, the rates of change in ScottyActivity are not fixed, as they depend on the continuous controllable control variables, and these can take any value within their actuation bounds. As explained in the beginning of this section, we use flow tubes to represent how the bounds of the state variables grow in the presence of continuous effects depending on continuous control variables. In practice, we use the minimum and maximum actuation bounds of each control variable to compute the minimum negative and maximum positive rates of change at a given layer. The growth of the state variable bounds from a layer to the next are then computed in the same way as COLIN does.

As an example, consider a layer in which an activity *navigate* with linear time-varying continuous effects $\dot{x}(t) = v_x(t)$ and $\dot{y}(t) = v_y(t)$ is active. Assume that the control variable v_x has actuation limits of $(-1, 2)$ while v_y is constrained to be within its $(1, 3)$ bounds. The minimum negative and maximum positive gradients operating on x at that layer are then $\nabla x^- = -1$, $\nabla x^+ = 2$ respectively. The gradients on y are $\nabla y^- = 0$ and $\nabla y^+ = 3$. Note that $\nabla y^- = 0$ since the bounds of state variables from one layer to the next are only allowed to grow. These gradients define how the boundaries of the flow tubes grow with time (as shown in Figure 8-3). If an activity *turbo-boost-x* that gave an additional boost to the x velocity of $(-2, 2)$ became active at a later layer, the minimum and maximum possible gradients for x would become $\nabla x^- = -3$ and $\nabla x^+ = 4$ from that layer on, as effects are additive. The lower and upper bounds for state variable x at layer $i+1$, x_{Li+1} and x_{Ui+1} are then computed as:

$$x_{Li+1} = x_{Li} + \nabla x^{-} \cdot (t_{i+1} - t_i) \tag{8.2}$$

$$x_{Ui+1} = x_{Ui} + \nabla x^{+} \cdot (t_{i+1} - t_i) \tag{8.3}$$

The bounds computed in the previous manner are used to test whether each continuous condition of an activity can be satisfied at a given layer using the same method employed by MetricFF. We now describe how this test is performed for linear inequalities. Any linear inequality can be expressed as:

$$\sum_j k_j \cdot x_j + c \leq 0, \tag{8.4}$$

where k_j and c are constants and x_j are the state variables. A linear equality condition can be satisfied at a given layer i if the intersection between the half-space defined by the inequality and the hypercube $R_i = \{\mathbf{z} \in \mathbb{R}^n \mid x_{j_{Li}} \leq z_j \leq x_{j_{Ui}}\}$ whose sides are the bounds of each state variable at layer i is non null. For inequalities expressed in the form of Equation (8.4), this condition is equivalent to asserting that the lower bound of the left hand side of the inequality at layer i, B_i, is smaller or equal than 0. We can compute this lower bound B_i using the bounds of the state variables with the following expression:

$$B_i = \sum_i \min(k_i \cdot x_{j_{Li}}, k_i \cdot x_{j_{Ui}}) + c, \tag{8.5}$$

Activities having linear inequality conditions that cannot be satisfied at a given layer according to the previous test may become satisfiable in a future layer if there are active continuous effects that expand the state variable bounds sufficiently in the right direction. In order to ensure that the relaxed graph is fully expanded, COLIN iteratively computes the next future time when one such unmet condition will become satisfiable and adds a layer at that time until all activities become applicable or all linear conditions that could ever be satisfied become satisfiable. The future point in time, if any, when an unsatisfiable linear inequality will become satisfiable is computed as follows. The linear condition is unsatisfiable because its lower bound B_i is greater than 0. By using the negative and positive gradients of each state variable at that layer, we can compute the negative gradient of the left hand side of the inequality, ∇e_i^- at layer i:

$$\nabla e_i^- = \sum_j \min(k_j \cdot \nabla {x_j}_i^-, k_j \cdot \nabla {x_j}_i^+) \leq 0 \tag{8.6}$$

If $\nabla e_i^- < 0$, the lower bound of the left hand side of the inequality, B, decreases with time, and the linear inequality will become satisfiable when B becomes negative, which will happen first after Δt_e units of time.

$$\Delta t_e = \frac{B_i}{|\nabla e_i^-|}$$

The next layer can then be created at time $t_i + \Delta t_e$, where t_i is the time of the current layer, with the hopes that satisfying this previously unmet linear condition may make some new activity applicable at that time. If $\nabla e_i^- = 0$ the linear inequality will never become satisfiable in the future, unless another continuous effect that makes this gradient negative becomes active in a later layer.

Also new in our heuristic is that we need to take into account convex quadratic conditions, that COLIN does not support. As we have just described, computing the future times when linear inequalities will become satisfiable can be done in a straightforward manner using Section 8.3. However, computing the future times when an unmet arbitrary convex quadratic condition will become satisfiable cannot be done with an analytical expression in an efficient manner. Our solution is to use linear over-approximations to the quadratic constraints in the heuristic, which are then handled as explained previously. In other words, our heuristic operates on a relaxed problem, where the relaxation replaces convex quadratic constraints with their linear over-approximation, on top of using the delete relaxation. If the user specifies the convex quadratic conditions as primitives, such as ellipsoidal regions, our planner computes the linear over-approximations automatically. For example, for ellipsoidal region conditions, we compute the axis aligned bounding box that contains the region.

Otherwise, the user can specify the approximations directly by providing lists of linear inequalities. These linear over-approximations constitute valid relaxations since they ensure that actions can always be executed earlier than they would be if the actual quadratic constraints were used.

The final difference in our heuristic compared to COLIN's is that *resource-constrained norm effects* (RNE) also need to be considered. These effects only reduce the availability of constrained resources and their application can only make activities infeasible (and never new activities possible). Therefore, in the spirit of delete relaxations, these effects are ignored in the heuristic. The optimization-based consistency check, that accurately computes these effects, rejects states that become infeasible due to this and the search follows through a different route. However, there is room to improve the current heuristic and make it aware of these interactions. We leave handling these effects more accurately in the heuristic for future work.

8.4 Search strategies

ScottyActivity implements two search algorithms. The first is Enforced Hill-Climbing (EHC) , which has been widely used with success by satisficing planners [47]. The second is a variation of EHC that we call *obj-EHC* that guides the search to plans that produce better objectives more efficiently. Additionally, ScottyActivity can optionally fall back to A* when these incomplete search algorithms fail. However, this is not discussed further in this thesis since our A* implementation is the standard one commonly used in other planners.

EHC (Algorithm 8.3) is a very popular greedy search algorithm that drops the open list every time it finds a state with a lower heuristic value. As a consequence, EHC is not complete. Since the heuristic is the estimated number of start or end activities to reach the goal, our EHC algorithm completely ignores the problem objective to guide the search. The `TEST-GOAL` procedure (Line 15) checks that the goal has been reached and, in that case, solves the optimization with the problem objective in order to generate the solution plan with the activity schedule and control trajec-

Algorithm 8.3: ScottyActivity-Plan-Ehc

```
Input: A PDDL-S planning problem $PP$.
Output: A valid PDDL-S plan or $nil$ if no plan could be found.
Algorithm
$S_0 \leftarrow$ MAKE-STATE($PP.I$)
$p_0 \leftarrow \{\}$
$h_{best} \leftarrow$ GET-HEURISTIC($S_0$)
$Q \leftarrow [<S_0, p_0, h_0>]$
while not IS-EMPTY($Q$) do
    $S, p \leftarrow$ POP($Q$)
    has-valid-descendants $\leftarrow nil$
    $A_H, A_{A \setminus H} \leftarrow$ GET-ACTIVITIES($PP$, $S$)
    for $a$ in $A_H + A_{A \setminus H}$ do
        $S_{new} \leftarrow$ APPLY($S$, $a$)
        $p_{new} \leftarrow p + \{a\}$
        $S_{new} \leftarrow$ TEST-CONSISTENCY($PP$, $S_{new}$, $p_{new}$)
        if $S_{new}$
            has-valid-descendants $\leftarrow$ true
            $p_{sol} \leftarrow$ TEST-GOAL($S_{new}$, $p_{new}$)
            if $p_{sol}$
                return $p_{sol}$
            $h_{new} \leftarrow$ GET-HEURISTIC($S_{new}$)
            if $h_{new} < h_{best}$
                $Q \leftarrow \{<S_{new}, p_{new}>\}$
                $h_{best} \leftarrow h_{new}$
                break
            else
                if $h_{new} < \infty$
                    PUSH($Q$, $<S_{new}, p_{new}>$)
        if all $a \in A_H$ explored
         and has-valid-descendants
            /* Do not explore non-helpful                        */
            break
return $nil$
```

tory. This plan is guaranteed to be optimal conditioned on the sequence of events chosen by the search.

While EHC is fast, ignoring the problem objective often leads to EHC making misguided choices that result in highly suboptimal plans. This happens because states that have the same heuristic value (in term of the number of activities to reach the goal) may have very different costs as specified by the problem objective. In order to improve the quality of the plans found by ScottyActivity, we introduce the *obj-EHC* algorithm (Algorithm 8.4), that is a variation of EHC that attempts to improve this issue. The obj-EHC algorithm uses a priority queue to sort the open states by the heuristic value (first) and, in case of a tie, by the cost incurred in that state so far according to the problem objective (second). Contrary to EHC, obj-EHC computes the heuristic value and the cost-so-far of all the helpful descendants of the current state, as opposed to dropping the queue and descending into a state as soon as a heuristic lower than the prior best one is seen. In the spirit of EHC, obj-EHC drops the queue when a state with a lower heuristic than the incumbent is removed from the priority queue. This provides a reasonable compromise between finding better quality plan and preserving the speed of EHC. However, dropping the queue makes obj-EHC also an incomplete search algorithm.

In order to compute the cost so far for a given state, we solve the same optimization problem that we use to test consistency with the objective of minimizing the cost as specified by the problem objective (Line 19). Note that the cost computed this way only provides an indication of the cost incurred so far, but does not provide any estimate of the future cost that may be incurred in by future activities. That is, this is the current cost of the plan skeleton and not a heuristic. Computing an estimate of the future cost would probably help the search considerably. However, this computation is not straightforward since it involves solving an optimization over possible future choices of activities with their conditions and effects. We leave this extension for future work.

Note that state expansion in obj-EHC is more computationally expensive than in EHC, since obj-EHC requires solving one additional optimization problem (the one

Algorithm 8.4: SCOTTYACTIVITY-PLAN-OBJ-EHC

```
Input: A PDDL-S planning problem $PP$.
Output: A valid PDDL-S plan or $nil$ if no plan could be found.
Algorithm
1  $S_0 \leftarrow$ MAKE-STATE($PP.I$)
2  $p_0 \leftarrow \{\}$
3  $h_{best} \leftarrow$ GET-HEURISTIC($S_0$)
4  $Q \leftarrow$ MAKE-PQUEUE()
5  PUSH($Q$, $< h_{best}, 0 >$, $< S_0, p_0 >$)
6  while not IS-EMPTY($Q$) do
7      $S$, $p \leftarrow$ POP($Q$)
8      has-valid-descendants $\leftarrow nil$
9      if $S.h < h_{best}$
10         $h_{best} \leftarrow S.h$
11         CLEAR($Q$)
12     $A_H$, $A_{A \setminus H} \leftarrow$ GET-ACTIVITIES($PP$, $S$)
13     for $a$ in $A_H + A_{A \setminus H}$ do
14         $S_{new} \leftarrow$ APPLY($S$, $a$)
15         $p_{new} \leftarrow p + \{a\}$
16         $S_{new} \leftarrow$ TEST-CONSISTENCY($PP$, $S_{new}$, $p_{new}$)
17         if $S_{new}$
18             has-valid-descendants $\leftarrow$ true
19             $obj_{new} \leftarrow$ MINIMIZE-OBJECTIVE($PP$, $p_{new}$)
20             $p_{sol} \leftarrow$ TEST-GOAL($S_{new}$, $p_{new}$)
21             if $p_{sol}$  return $p_{sol}$
22             $h_{new} \leftarrow$ GET-HEURISTIC($S_{new}$)
23             if $h_{new} < \infty$
24                 PUSH($Q$, $< h_{new}, obj_{new} >$, $< S_{new}, p_{new} >$)
25         if $all\ a \in A_H\ explored$
            and has-valid-descendants
               /* Do not explore non-helpful                                   */
26             break
27 return $nil$
```

minimizing the cost) on top of the optimization problems used to compute the bounds for each state variable. Moreover, obj-EHC maintains a priority queue, which is not needed in EHC. Finally, obj-EHC requires expanding all children of each state as opposed to stopping right away when a state with a better heuristic value is found. In Chapter 9 we compare both, discuss the advantages and disadvantages of each and show that the quality of the plans returned by obj-EHC is significantly higher in general.

Both search algorithms explore by default only the helpful activities of each state. These are the activities that appear in the first layer of the relaxed planning graph [47]. However, we cannot model accurately some of the non-linear effects in our heuristic (i.e. *resource-constrained norm effects*, RNEs). As a result, the heuristic sometimes fails to identify activities that need to take place as helpful when RNEs are used. To alleviate this, we allow our search to explore the applicable, but non-helpful, descendant activities if the current state does not have any valid successors (Line 25).

Finally, ScottyActivity can, like other planners, fall back to A* search if EHC or obj-EHC do not find a plan. In practice A* search is very slow in non-trivial problems and we do not show the performance of our planner using this search method.

Chapter 9

ScottyActivity Experimental Results

In this chapter we evaluate the scalability of ScottyActivity in both synthetic domains and real expressive robotic scenarios. First, we use synthetic domains to illustrate why maintaining both continuous time and continuous control variables, as ScottyActivity does, is essential in order to plan efficiently over long horizons. We then present three new benchmarking domains that represent robotic missions with coordination constraints and show the scalability of our approach in these problems.

We evaluated the empirical performance of our planner on an Intel Core i7-3770 3.40 GHz using Gurobi 7.0.1 as ScottyActivity's internal convex optimization solver.

9.1 Synthetic Benchmarks

In our first test, we compare the performance of ScottyActivity against Kongming in a robotic sampling scenario that shows why Kongming's time discretization can be a problem in common applications and how ScottyActivity overcomes this issue. We then compare ScottyActivity's performance against POPF [22], a state of the art planner capable of planning with continuous effects. Recall that POPF does not support control variables and therefore needs to represent different rates of change with multiple discretized actions. This example shows that representing control variables continuously is essential in robotic applications and that the alternative discretization of control variables is not a scalable approach.

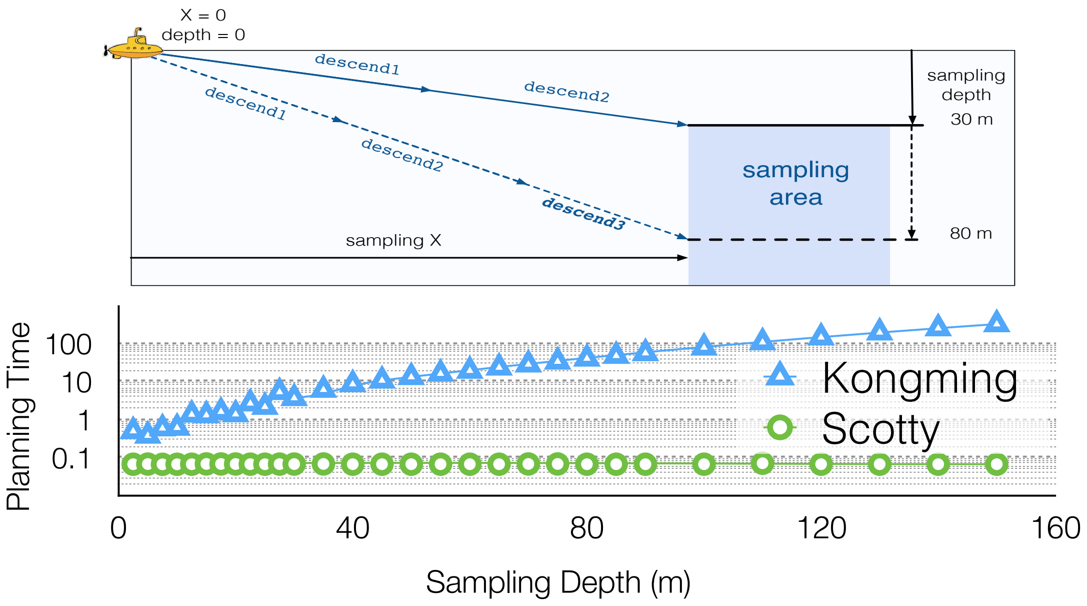


Figure 9-1: Example scenario that shows the problems of discretizing time. Planning time is shown in seconds.

9.1.1 Discretization of time

Kongming is the first mixed discrete-continuous activity planner that performs simultaneous activity and motion planning as a tightly coupled problem. It does so by allowing continuous control variables and by using an efficient representation of all possible trajectories with flow tubes. However, while Kongming's approach is very innovative, we argued in Chapter 2 that its fixed time discretization hinders its performance in problems that require long planning horizons and where both short and long lived activities coexist. In this section we present a simple robotic scenario that highlights this issue and we show that ScottyActivity's continuous time formulation avoids this problem.

The problem consists of a simple autonomous underwater vehicle (AUV) sampling mission as shown in Figure 9-1. In this problem the AUV needs to reach a certain depth range in order to take a sample. We parameterize this scenario in terms of such sampling depth. The AUV can use the action descend to modify its depth according to the bounded control variable *descent-rate*. Because Kongming discretizes time in constant time steps, increasing the target sampling depth forces Kongming

	2D AUV 1	**2D AUV 2**	**3D AUV**	**Firefighting 1**	**Firefighting 2**
Kongming	3.633	9.736	13.063	1.505	20.202
ScottyActivity	0.054	0.025	0.192	0.03	0.372

Table 9.1: Comparison between Kongming and ScottyActivity in several domains. Results show planning time in seconds.

to create additional fact and action layers and, additionally, more variables that the MILP solver needs to consider. As a result, the performance of Kongming degrades very quickly with the target sampling depth as shown in Figure 9-1. On the other hand, ScottyActivity's performance is constant (and orders of magnitude better than Kongming's). This is an expected result. Because ScottyActivity does not discretize time, it solves essentially the same problem regardless of the target sampling depth. Only one *descend* activity is needed and it is only the parameters of the mathematical optimization that change in each instance of the problem. For the sake of completeness we also benchmarked ScottyActivity in other domains in which Kongming was used. These domains, described in detail in [63], typically showcase one or more mobile robots moving in a 2D or 3D environment and completing objectives that involve visiting different locations. Table 9.1 shows that ScottyActivity exhibits a large performance advantage over Kongming in these domains as well.

9.1.2 Discretization of control variables

In this section we argue that continuous control variables are essential to efficiently solve robotic planning problems. In order to do this, we compare the performance of ScottyActivity and POPF in a synthetic robotic domain. While POPF is a very efficient planner, it only supports linear continuous effects with fixed rates of change defined in advance, as opposed to the continuously controllable rates of change (control variables) that ScottyActivity supports. In this synthetic domain a mobile vehicle needs to visit six regions. For the sake of simplicity the order of the regions is fixed in this problem. The objective of this problem consists in minimizing the plan makespan. In order to move, the vehicle uses the *navigate* activity. This activity has two CLTE effects, each operating in state variables x and y respectively:

	Planner	A	t	S	L	O	l	v_{avg}
ScottyActivity (EHC)		1	0.81	24	24	151.11	203.65	1.35
POPF	4 directions	4	1.07	326	38	200.22	376.40	1.88
	8 directions	8	1.77	531	36	192.22	415.90	2.16
	5 steps	24	11.84	3530	36	277.17	413.92	1.49
	7 steps	48	22.66	6168	32	342.56	413.01	1.21
	9 steps	80	59.62	15037	32	469.92	459.06	0.98
	11 steps	120	133.35	31160	32	430.13	446.35	1.04

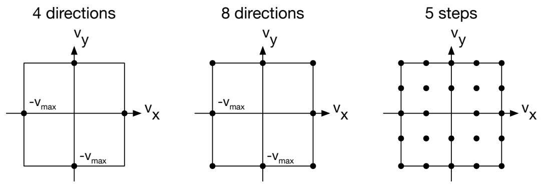


Table 9.2: Discretization example results. **A**: number of navigate actions in the domain; **t**: Planning time in seconds; **S**: Number of nodes expanded; **L**: Plan length in number of actions; **O**: Makespan (objective value) of the plan returned; ***l***: Length of the path traveled; v_{avg}: Average speed of the vehicle throughout the plan. The diagrams on the right show the discretization performed in some of the problem instances. Each black dot represents a *navigate* activity with its given v_x and v_y. For ScottyActivity, any velocity value within the square is allowed.

```
(increase (x) (* (vx) #t))
(increase (y) (* (vy) #t))
```

For ScottyActivity, v_x and v_y are continuous control variables that can take any value between -2 and 2. Since POPF only supports fixed rates of change, the values of v_x and v_y have to be fixed in advance. Therefore, we create multiple *navigate* activities with different discretization values for the velocities. We compare the performance of ScottyActivity using EHC to POPF in problem instances using different discretization values (from 4 navigate activities to up to 120 navigate activities).

The results are shown in Table 9.2. The diagrams on the right show the discretizations performed for some of the problem instances. Since the visit order of the regions has been fixed, the plan returned by ScottyActivity is optimal. Given that ScottyActivity only needs one *navigate* activity to represent all possible velocities, this problem can be solved very quickly. As expected, the results show that as the number of *navigate* activities increases, it becomes harder for POPF to find a solution since it needs to explore more states.

Figure 9-2 shows the optimal trajectory returned by ScottyActivity and the trajectories returned by POPF for different discretizations. A finer discretization (more *navigate* activities), provides the chance of finding better plans, since more fine grained control of the velocities is possible, which in turn allows better navigation headings to be selected. However, the results show that the plans returned by POPF are not

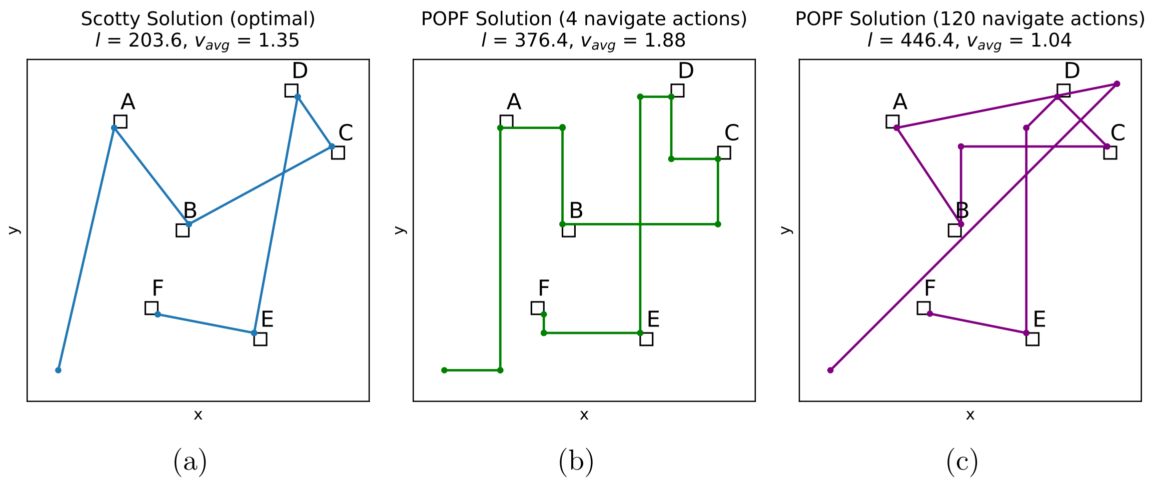


Figure 9-2: Trajectories of the discretization example. In this domain, the solution returned by ScottyActivity is optimal (a). With 4 navigate actions, the solution is worse than optimal and harder to find (b). Adding more actions, the problem becomes much harder to solve, and the solution returned gets worse (c)

only much harder to find, but they also do not get better with more *navigate* activities. We hypothesize that this is due to the greedy nature of the Enforced Hill Climbing algorithm that POPF uses, and the fact that the number of actions to the goal and not the objective is what guides the search. As an example consider the two first actions of the plan shown in Figure 9-2c, which are the navigate activities with velocities $(0.8, 0.8)$ and $(-2, -0.4)$ respectively. Because EHC commits to those first two actions with their fixed velocities and headings, POPF's linear program has no other option than taking the vehicle all the way to the top right corner in order to be able to reach region A afterwards.

This discretization example shows that even in simple domains, continuous control variables provide a large advantage over discretized rates of change.

9.2 Evaluation in Robotic Domains

To the best of our knowledge, there are currently no prior benchmarks available that can exploit the features of our planner. Therefore, in order to showcase the new capabilities of our planner and to show that our optimization framework is fast and scalable, we present three new expressive robotic domains and benchmark our planner against them. Since no other planner can solve these domains, we also provide simplified, linear version of some of these domains that we use to compare our planner to POPCORN [85]. We compare against this planner since it supports control parameters, which are essential for these domains. The simplified domains do not capture the full expressivity of these problems. However, because POPCORN supports control parameters, we can still represent some aspects of these problems within its formalism. We also compare against the first version of our planner, Scotty1 [36], since it uses a much simpler optimization model than the one presented in this work.

We describe these domains in this section. The PDDL description of these that ScottyActivity takes as input is provided in Appendix C.

9.2.1 The AUV Domain

In this domain an Autonomous Underwater Vehicle (AUV) needs to visit and take samples at multiple regions. This domain is similar to POPCORN's 2D-AUV-Power domain, that is based on prior Kongming and ScottyActivity domains. There are two main differences between POPCORN's domain and ours. First, since POPCORN does not support controllable rates of change, the effects of the *glide* action are modeled as discrete numeric displacements on the x, y variables at the end of the action, whereas we model the motion as a continuous effect that takes place while the action is being executed. Moreover, since POPCORN only supports linear constraints, its authors model the maximum power of the vehicle as a simple linear constraint on the displacements at the end of the action (e.g. $3d_x + 4d_y \leq 60$), while we can use the new features of our SOCP model to limit the magnitude of the velocity ($v_x^2 + v_y^2 \leq v_{max}^2$). The objective of this domain consists in minimizing the plan makespan. The simplified linear version of this domain is similar except that we place no constraints on the v_x, v_y velocities other than their simple independent bounds.

9.2.2 The ROV Domain

This domain is based on the motivating example presented in Section 3.1 but without an AUV. As in the motivating example, the Remotely Operated Vehicle (ROV) needs to take samples in multiple regions and end, together with the ship, in the destination region. Note that our planner decides where to station the ship while the ROV is taking samples, and that good selections of that position may allow the ROV to visit several regions without having to be recovered by the ship first. The objective for this domain minimizes a linear combination of the plan makespan and the distance traveled by the ship. Figure 9-3 shows an example solution plan returned by our planner for problem 6 of this domain.

In the simplified linear version of this domain we remove the velocity norm constraints. Furthermore, the maximum distance constraints, which are modeled with the convex quadratic constraints of being inside a circle, are replaced by a simpler linear polygonal over approximation of such a circle (an octagon in this case). Since we cannot model the distance traveled by the ship without quadratic constraints, the simplified version only optimizes the makespan of the plan.

9.2.3 The Air Refueling Domain

In this domain an autonomous Unmanned Aerial Vehicle (UAV) needs to take pictures of several regions before landing at the destination location. Since the UAV has limited fuel, it needs to refuel in-air from a tanker plane. While refueling, both planes can keep moving but they need to stay within a maximum distance. The UAV fuel decreases as a function of the distance traveled and the square of the velocity ($\dot{f} = -k_1 v - k_2 v^2$). As in the ROV domain, the objective for this domain is to minimize a linear combination of the plan makespan and the distance traveled by the tanker plane. In instances 11-20 there is an additional UAV, but only one UAV can refuel from the tanker plane at a time. Figure 9-4 shows an example solution plan returned by our planner for instance 15 of this domain.

This domain is challenging for several reasons. First, the planner needs to con-

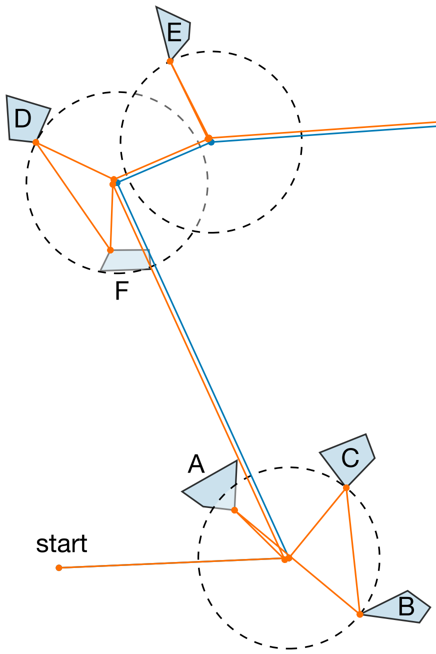


Figure 9-3: Trajectories of ship (blue) and ROV (orange) in problem 06 of the ROV-regions domain. Note how the planner selects ship positions so that the ROV can take samples at multiple regions without having to reposition the ship and while observing the tether range constraint.

sider the simultaneous trajectories of multiple vehicles and also their fuel levels. Second, and more importantly, while our optimization model supports the *resource-constrained norm effects* (such as the fuel decrease depending on the norm or squared norm of the velocity), the heuristic does not consider these effects directly. Therefore, our planner only chooses the refuel activities when reaching other regions becomes infeasible due to having insufficient fuel.

We do not present a simplified version of this domain because POPCORN would not be able to solve a linear alternative. The reason is that the *refuel* activity requires continuous effects since both the tanker and the UAV have to be flying simultaneously while staying close to each other. POPCORN cannot model this since the numeric change can only be applied at the beginning or end of an activity and not continuously in time. This domain also requires that the fuel of each UAVs decreases as a function of the magnitude of their velocities, which POPCORN does not support either.

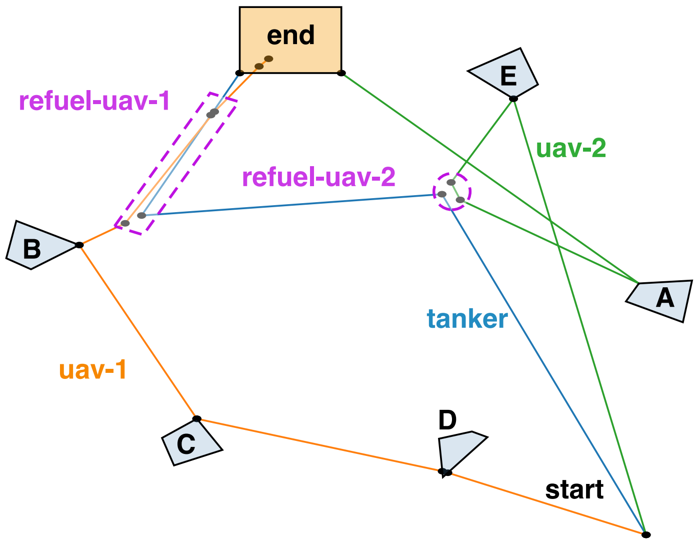


Figure 9-4: Example solution for instance 15 of the refueling domain with a tanker plane (blue) and two UAVs (orange and green). Note that the planner finds a trajectory for the tanker that allows it to refuel both UAVs as needed.

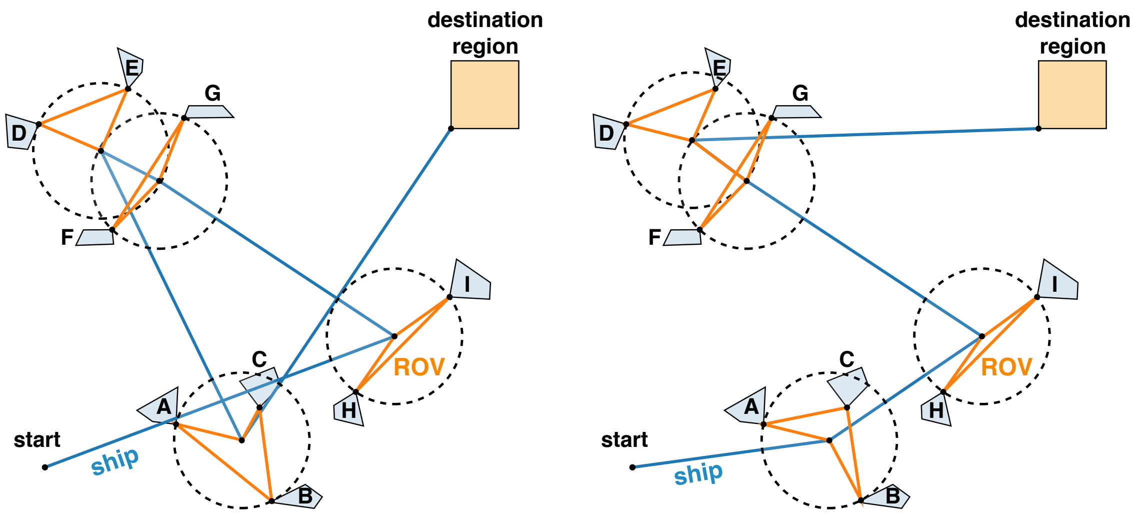
9.2.4 Results

We evaluated ScottyActivity with the two search approaches discussed in Section 8.4 in these robotic domains. The results are shown in Table 9.3. As seen in column T, our convex optimization model, a key contribution of our work, is solved very quickly. The mean optimization time per problem grows for more complicated instances since these have more state variables and require more activities, which results in far more decision variables and constraints at later stages of the search. However, most optimization problems are solved in less than 10 ms in average for small to medium domain instances and in less than 50 ms for larger ones. This is important since large domain instances require solving tens of thousands of optimization problems, as seen in the table (column N). This kind of performance would not be possible if we used a non-convex non-linear optimization model with a general purpose non-linear optimizer.

Table 9.3 also illustrates the performance characteristics of the EHC search algorithm compared to the obj-EHC, that breaks heuristic ties based on the objective of each state, as discussed in Section 8.4. Column $O_{\%}$ shows the relative value of the objective of the obj-EHC approach compared to EHC. Negative values indicate

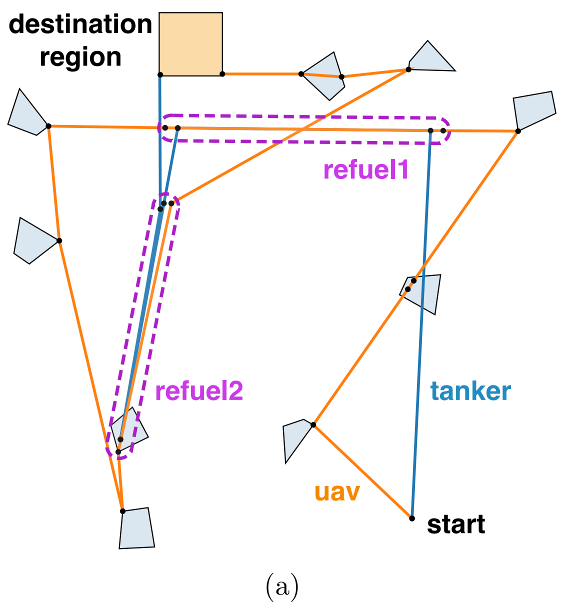
	AUV											**ROV**											**Air Refueling**										
	EHC					**obj-EHC**						**EHC**					**obj-EHC**						**EHC**					**obj-EHC**					
	t	L	S	N	T	t	L	S	N	T	$O_{\%}$	t	L	S	N	T	t	L	S	N	T	$O_{\%}$	t	L	S	N	T	t	L	S	N	T	$O_{\%}$
01	0.53	4	4	17	2	0.54	4	4	21	2	0.0	1.16	16	19	153	3	1.20	16	19	172	3	0.0	0.97	8	11	111	3	1.08	8	12	133	3	-0.0
02	0.55	8	10	41	2	0.64	8	9	46	1	0.0	1.74	20	35	281	4	1.52	20	25	226	4	-0.0	1.68	12	19	191	5	1.93	12	20	221	6	-0.0
03	0.66	12	18	73	2	0.80	12	15	76	2	0.0	2.79	24	57	457	4	1.90	24	32	289	4	-0.1	2.44	16	30	301	6	2.72	16	28	309	6	-0.0
04	0.84	16	28	113	2	0.90	16	22	111	2	-5.7	4.34	36	79	633	5	3.51	36	48	433	6	-0.2	5.78	18	74	669	7	3.52	18	36	397	7	26.1
05	0.94	20	40	161	2	0.98	20	30	151	2	-4.0	7.26	40	119	953	6	4.32	40	55	496	7	-14.7	3.82	20	45	433	7	4.74	20	43	464	8	-8.5
06	0.89	20	40	161	2	0.97	20	30	151	2	-13.3	10.71	52	157	1225	7	7.24	52	74	651	9	-21.2	5.68	24	60	583	8	5.57	22	50	541	8	-12.1
07	1.18	24	54	217	2	1.11	24	39	196	2	-9.5	16.38	56	213	1665	9	9.05	56	83	734	11	-15.5	11.18	28	98	927	11	5.69	-	55	585	8	-
08	1.24	24	54	217	2	1.12	24	39	196	2	-31.4	19.75	68	236	1822	9	14.84	68	108	950	14	-12.1	10.93	32	96	916	11	11.13	30	77	808	12	-30.1
09	1.46	28	70	281	2	1.34	28	49	246	2	-19.1	32.24	72	338	2607	11	17.48	72	119	1053	15	-21.0	12.03	32	105	997	11	13.29	32	94	945	12	-77.0
10	1.49	28	70	281	2	1.50	28	49	246	3	-20.4	35.09	84	350	2659	12	21.96	84	136	1203	16	-4.7	17.14	38	115	1124	14	15.05	34	100	1041	13	-63.0
11	1.87	32	88	353	3	1.90	32	60	301	3	-14.0	40.10	88	392	2993	12	25.47	88	150	1313	17	-23.5	2.14	10	18	289	5	3.60	10	29	494	6	-0.5
12	1.86	32	88	353	3	1.62	32	60	301	3	-33.5	56.63	100	451	3410	15	31.83	100	176	1490	20	-21.7	10.77	14	64	1025	10	-	-	-	-	-	-
13	2.29	36	108	433	3	1.92	36	72	361	3	-33.1	52.82	96	412	3094	15	34.41	104	186	1556	20	-23.2	15.43	16	80	1281	11	-	-	-	-	-	-
14	2.30	36	108	433	3	2.03	36	72	361	3	-23.4	68.63	108	497	3683	17	39.55	108	207	1699	21	-25.9	21.33	18	98	1569	13	-	-	-	-	-	-
15	2.85	40	130	521	3	2.35	40	85	426	3	-26.1	87.77	120	586	4315	18	45.00	120	224	1827	22	-16.6	49.61	22	165	2581	19	-	-	-	-	-	-
16	2.75	40	130	521	3	2.34	40	85	426	3	-19.9	95.51	124	630	4659	18	48.94	124	237	1930	23	-17.6	60.67	24	191	2907	20	-	-	-	-	-	-
17	3.57	44	154	617	3	3.10	44	99	496	4	-32.0	119.82	136	712	5301	20	58.95	136	259	2128	25	-21.9	78.57	26	222	3373	23	-	-	-	-	-	-
18	4.48	48	180	721	4	3.42	48	114	571	4	-38.4	151.10	140	885	6531	21	64.24	140	272	2245	26	-19.7	658.30	32	1147	17003	38	-	-	-	-	-	-
19	5.04	52	208	833	4	4.04	52	130	651	4	-35.9	161.05	144	923	6802	21	70.79	144	288	2373	27	-19.6	563.87	34	906	13972	39	-	-	-	-	-	-
20	6.15	56	238	953	4	4.69	56	147	736	4	-40.8	218.39	156	1181	8658	22	82.21	156	314	2591	29	-20.5	249.25	36	419	6210	39	-	-	-	-	-	-

Table 9.3: Benchmarking results. **t**: Planning time in seconds; **L**: Plan length in number of actions; **S**: Number of nodes expanded; **N**: Number of optimization problems solved; **T**: Mean optimization time for each optimization problem in milliseconds; **O**$_{\%}$: Relative objective value achieved by obj-EHC compared to EHC. Values with '-' denote problems that could not be solved in 1200 seconds.



(a) (b)

Figure 9-5: Plans found by ScottyActivity for problem 9 of the ROV domain using EHC search (a) and obj-EHC (b). The objective is a combination of the plan makespan and the distance traveled by ship. obj-EHC finds a better plan than EHC, with a 21% improvement in the objective, by taking samples at closer regions first.



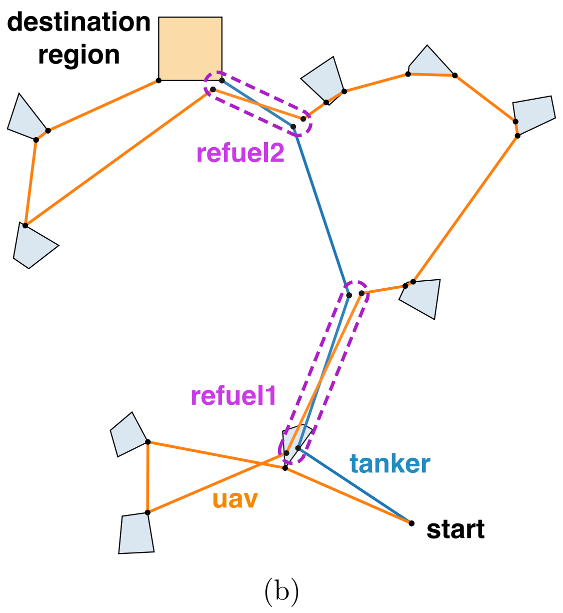


Figure 9-6: Plans found by ScottyActivity for problem 9 of the Air Refueling domain using EHC search (a) and obj-EHC (b). The objective is a combination of the plan makespan and the distance traveled by tanker plane. Note how the plan found by obj-EHC is much better (77% improvement in the objective).

an improvement in the objective. As expected, obj-EHC produces better plans in general. This improvement is very significant in some instances, showing a reduction in the objective of more than 50% (Figure 9-5). As explained in Section 8.4, obj-EHC is more computationally expensive as it requires an extra optimization problem minimizing the objective per state and it checks all the helpful descendants of each expanded node, as opposed to immediately picking the one with lower than the incumbent best heuristic. Therefore, we expected that obj-EHC would take longer than EHC to find plans. However, the results show that this is not the case, and that obj-EHC explores less states, finds better plans and takes less time in general than EHC. In particular, in the ROV domain, obj-EHC takes less than half the time to find plans than EHC for more difficult instances. We believe the objective guidance is being very effective in these domains, in which taking a sample in a nearby region or a further one looks the same in number of actions, but very different in terms of the objective.

However, obj-EHC is not always better than EHC. In particular, obj-EHC strug-

	AUV-simplified									**ROV-simplified**								
	ScottyAct(EHC)					**Scotty1**		**POPCORN**		**ScottyAct**(EHC)					**Scotty1**		**POPCORN**	
	t	L	S	N	T	t	L	t	L	t	L	S	N	T	t	L	t	L
01	0.54	4	4	17	2	0.48	4	0.05	4	0.86	16	19	153	2	0.89	16	0.57	16
02	0.55	8	10	41	1	0.49	8	0.15	8	1.25	20	35	281	2	1.48	20	1.95	20
03	0.61	12	18	73	1	0.56	12	0.30	12	1.74	24	57	457	2	2.33	24	3.59	24
04	0.72	16	28	113	1	0.64	16	0.58	16	2.54	36	79	633	3	5.58	36	6.53	36
05	0.80	20	40	161	1	0.77	20	0.94	20	4.07	40	119	953	3	7.54	40	16.34	40
06	0.83	20	40	161	1	0.74	20	0.91	20	5.46	52	156	1214	3	12.79	52	24.77	52
07	0.92	24	54	217	1	0.92	24	1.50	24	7.85	56	213	1621	4	16.73	56	49.84	56
08	0.99	24	54	217	1	0.94	24	1.52	24	9.06	68	233	1753	4	55.40	84	77.21	68
09	1.15	28	70	281	1	1.24	28	2.22	28	14.36	72	328	2478	5	86.75	96	107.27	72
10	1.13	28	70	281	1	1.14	28	2.23	28	15.53	84	345	2565	5	119.23	100	150.69	84
11	1.43	32	88	353	1	1.44	32	3.29	32	18.34	88	396	2952	5	96.59	96	175.83	88
12	1.42	32	88	353	1	1.43	32	3.26	32	24.79	100	450	3335	6	142.33	108	242.64	92
13	1.49	34	92	369	2	1.53	34	3.89	34	23.36	96	411	3023	6	126.13	104	278.46	96
14	1.48	34	92	369	2	1.57	34	3.90	34	30.35	108	495	3639	7	180.93	116	343.66	108
15	1.80	38	114	457	2	2.12	38	4.88	38	38.62	120	578	4261	7	254.07	128	460.56	112
16	1.75	38	114	457	2	2.19	38	5.60	38	45.26	124	660	4826	8	158.47	108	525.74	116
17	2.22	42	138	553	2	2.50	42	6.58	42	56.49	136	743	5469	8	204.34	120	617.31	128
18	2.81	46	164	657	2	3.53	46	9.91	46	68.51	140	890	6484	8	271.64	132	783.90	140
19	3.31	50	192	769	2	4.12	50	13.01	50	73.40	144	926	6737	9	391.01	152	834.64	136
20	3.93	54	222	889	2	5.30	54	17.18	54	98.11	156	1177	8479	9	500.96	164	1028.76	148

Table 9.4: Benchmarking results for simplified domains **t**: Planning time in seconds; **L**: Plan length; **S**: Number of nodes expanded; **N**: Number of optimization problems solved; **T**: Mean optimization time for each optimization problem in milliseconds.

gles in the Air Refueling domain with two UAVs. This is a challenging domain since the need to refuel is not accurately captured in the heuristic. Since the refueling activity requires the tanker plane to move to wherever the UAV that wants to refuel is, this activity is expensive in terms of the objective, but is not deemed to be useful by the heuristic. Therefore, the algorithm tends to prefer other irrelevant actions that are not useful (such as taking the same sample over and over again).

Finally, we evaluate ScottyActivity in the simplified linearized versions of the domains as described earlier (Table 9.4). Recall that our optimization model degrades gracefully to an LP when the problem solved does not require quadratic constraints. Therefore, these results let us answer the question of what is the performance penalty of switching from a linear program formulation to a SOCP one. As seen in the table, the difference between the mean optimization time for the linear problems solved in the simplified domains and the SOCP ones from the full domains is very small for the simpler instances and significant for more difficult instances. However, this difference is always well within an order of magnitude. Moreover, we should highlight that the linearized version of the domains is significantly simpler, as it not only linearizes some constraints (such as the ROV tether range ones) but also drops many other constraints, like the norm ones or the ones required to minimize the traveled distances. We can conclude that using SOCPs for consistency checking is not only practical, but that the performance trade-off is well worth it considering the added expressivity that they provide. Finally, we compare our planner in these simplified domains against Scotty1 and POPCORN. Since our optimization model is significantly more complex than theirs, even in these linear domains, given the extra variables and constraints that we require, we expected that our planner would be slower. However, Table 9.4 shows that this is not the case and our planner performs significantly better. We hypothesize that this is due to the superior performance of the Gurobi solver compared to the solvers used by Scotty1 (CPLEX 12.4) and POPCORN (lpsolve 5.5). Additionally, POPCORN's test were kindly run on a slower i5-M540 2.53GHz processor by its authors, since they could not share the planner binary with us.

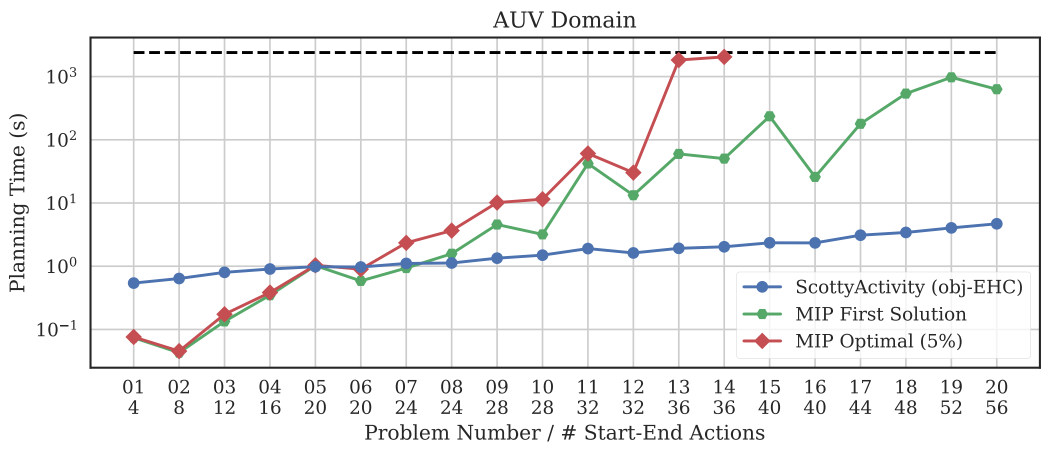
9.3 Comparison With a Mixed Integer Approach

As we explain in Chapter 6, when the plan skeleton that describes the chosen starts and ends of activities and their order is known, the state and control trajectories can be found efficiently by solving a convex optimization problem. The hybrid activity and trajectory planning problem reduces to finding the plan skeleton. This is hard due to the highly combinatorial nature of the problem. In this thesis, we argue that this problem is best solved using heuristic forward search with a delete relaxation heuristic, as this method has been proven to be very effective in activity planning problems. However, this problem could also be solved using a mixed-integer optimization approach. Mixed-integer solvers use branch and bound search algorithms. The branch and bound search is often guided by the solution to relaxed optimization problems in which the integer variables are allowed to take continuous values. We show in this section that our approach based on heuristic forward search performs at least two orders of magnitude better than an alternative mixed-integer approach.

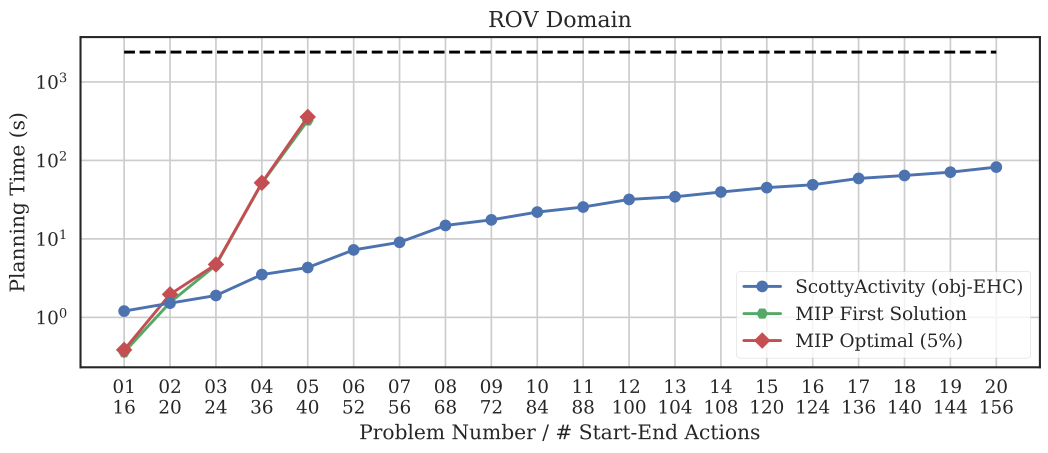
The mixed-integer formulation that we use in this section employs the same decision variables and constrains for the state and control variables and for the activity durations, continuous conditions and effects, as presented in Chapter 6. An important difference between the mixed-integer approach that we use in this section and ScottyActivity is that the former can only find the solution to a planning problem when it is provided with a fixed length for the plan. This length indicates the maximum number of start and end activities (events) that the plan can have. The reason for this limitation is that the decision variables and constraints are fixed in the optimization problem. ScottyActivity, on the other hand, determines the required number of start and end activities as part of its search process. A well-known approach to solving planning problems with a mixed-integer approach when the plan length is not known consists in iteratively solving the problem with increasing plan lengths until a solution is found or the maximum plan length or time limit is exceeded. However, in the results shown in this section we provided the MIP planner with a fixed plan length, that matched, for each problem, the length of the plan found by ScottyActivity using the obj-EHC algorithm.

In order to select the start and end activities and their order, the MIP approach uses integer variables. For each step in the fixed length plan, we create a binary variable for each start or end activity that indicates whether the activity is selected for such step in the plan. The fixed plan length indicates the maximum number of start or end activities, but the MIP planner is allowed to find a shorter plan by selecting no-op activities at the last steps of the fixed length plan. Additional binary variables are created to represent the propositions that hold at each step in the plan. We use the standard big-M method to enable or disable constraints when an activity is selected.

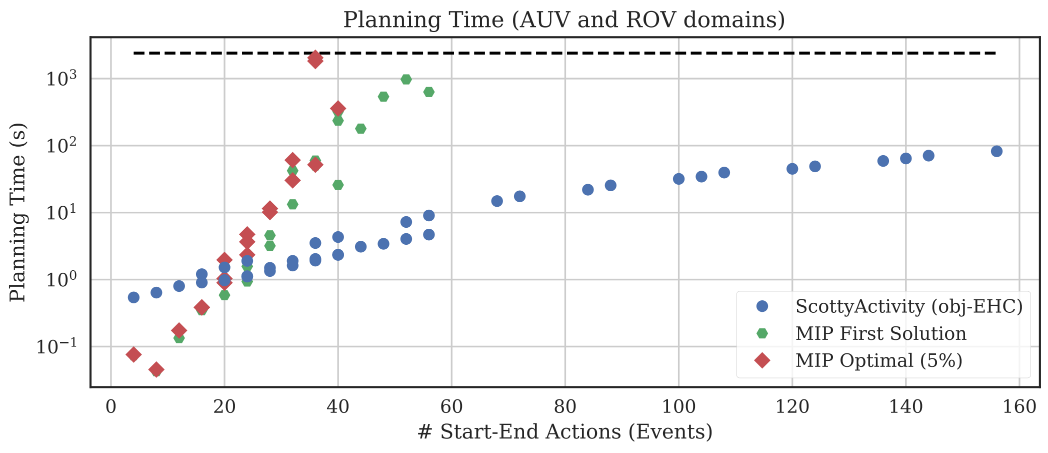
We compared the performance of this MIP planner against ScottyActivity using the obj-EHC algorithm in the AUV and ROV domains from Section 9.2. We used a



(a)



(b)



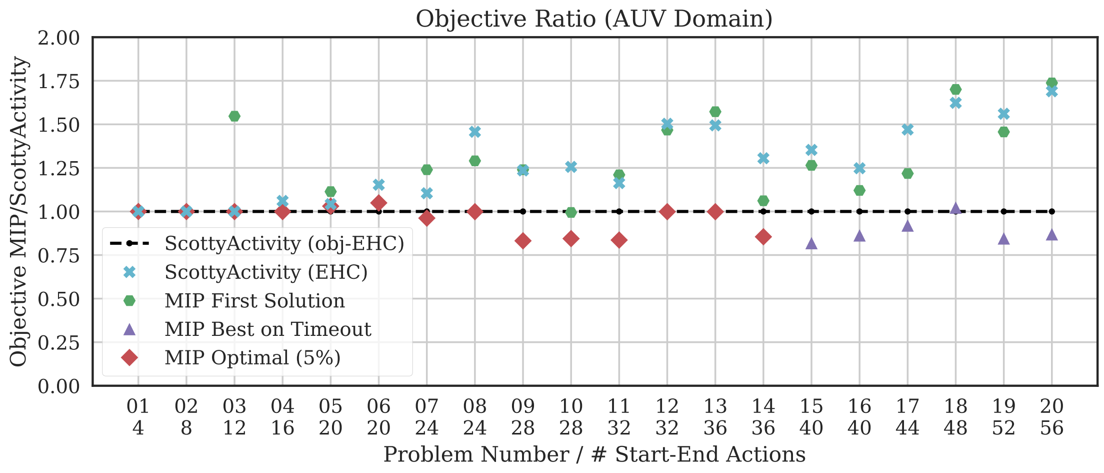
(c)

Figure 9-7: Planning time for ScottyActivity (obj-EHC algorithm) and the MIP approach in the AUV (a) and ROV (b) domains. The planning time for the first MIP solution (green) and optimal to 5% tolerance (red) are shown. Figure (c) shows the planning time for both domains as a function of the required number of start/end actions to solve the problem. A time limit of 2400 seconds was used.

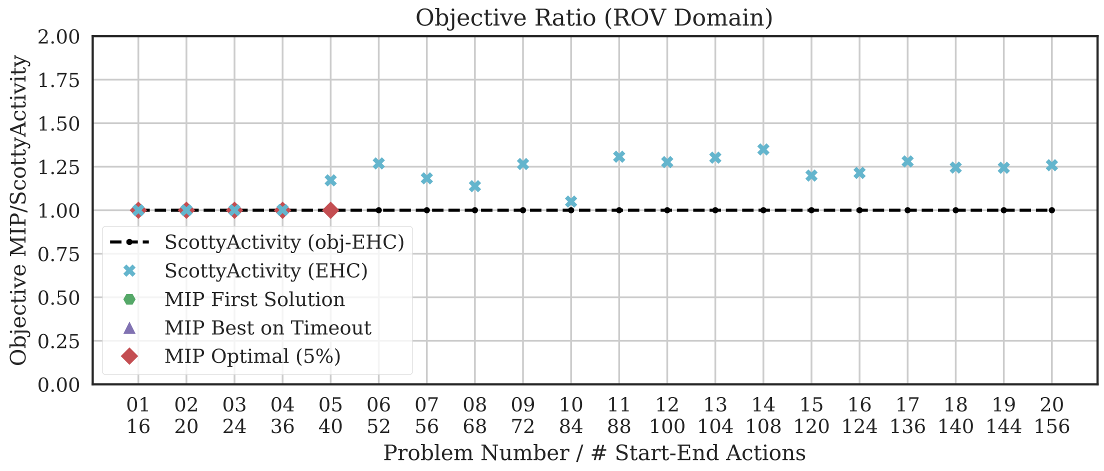
time limit of 2400 seconds for the MIP solver for each problem, and we used Gurobi 7.5 as the MIP solver. The MIP solver was able to solve all the problems in the AUV domain (Figure 9-7a). For small problems, up to instance 5, that require less than 20 events, the MIP planner was able to find a solution faster than ScottyActivity. In many cases, this solution was also the optimal (to 5% tolerance) for the given fixed length. However, the performance of the MIP solver degrades quickly as the complexity of the problem increases. Problems needing more than 40 events where solved two orders of magnitude slower than ScottyActivity. For example, the MIP planner found a feasible solution for instance 19 of the AUV domain in 970 seconds, while ScottyActivity solved the same problem in about 5 seconds. The MIP planner was able to find optimal solutions (proved to 5 % tolerance) for instances 14 and lower, which need fewer than 36 events.

The ROV domain proved to be more challenging for the MIP solver (Figure 9-7b). In this case, the MIP solver was only able to solve the first five problems within the maximum time limit of 2400 seconds. Instance 5, requiring 40 events was solved by the MIP planner in 320 seconds. The same problem was solved by ScottyActivity using the obj-EHC algorithm in about 7 seconds. One of the reasons why the ROV domain is more challenging for the MIP planner is that the problems are more complex and require a higher number of activities. However, we speculate that this is not the only reason. In the AUV domain, the MIP planner was able to find solutions for problems requiring up to 56 events, while it was not able to find a solution for a problem requiring 52 events in the ROV domain. The reason this may happen is that the ROV domain not only requires longer plans, but the number of possible activities is also significantly higher in this domain than in the AUV domain. Furthermore, there are several activities that are mostly discrete, such as the deployment and recovery activities. For this activities the planner is probably not finding useful relaxations.

Another interesting comparison between the ScottyActivity planner and the MIP approach is the quality of the returned plans. We report, in Figure 9-8, the objective of the plans found by ScottyActivity and the MIP planner. For the MIP planner we are interested in the objective of the first solution found, the best solution found



(a)



(b)

Figure 9-8: Objective ratio between the MIP approach and ScottyActivity (obj-EHC algorithm) in the AUV (a) and ROV (b) domains. The figures show the ratio achieved by the first MIP solution (green), the optimal to 5% tolerance (red), and the best MIP solution found before the timeout (purple). A time limit of 2400 seconds was used.

within the time limit and the optimal solution for the fixed plan length. The results show the ratio between the objective of the MIP solutions and the ScottyActivity solution for the AUV (Figure 9-8a) and ROV (Figure 9-8b) domains when using the obj-EHC algorithm. As seen in the figures, the obj-EHC algorithm returns fairly high quality solutions. In particular, the solutions found by ScottyActivity with the obj-EHC algorithms were always better than the first solution found by the MIP planner, even when this solution was often found two orders of magnitude slower. This results show the improvement of the obj-EHC algorithm when compared to the standard EHC algorithm, whose solutions are often comparable to the first solutions found by the MIP planner. One of the benefits of the MIP approach is that it is able to find optimal solutions for a fixed plan length. However, these solutions are often computationally expensive to find.

These results indicate that the MIP approach does not scale to large planning problems. Moreover, it is important to highlight that the MIP planner was provided with the number of required events for each planning problem. The runtime performance would be significantly worse if the MIP planner had to solve problems of increasing fixed-length until a solution was found. Additionally, the Gurobi solver is designed to use multiple cores simultaneously, while our search algorithm is single-core. While we do not believe that a mixed-integer approach is the best method to solve large hybrid activity and trajectory planning problems, it is a useful approach for solving small planning problems that require optimal solutions. Moreover, it may be possible to use a mixed-integer formulation to improve plans that are first found using a heuristic forward search approach. We leave this work for future research.

Part III

ScottyPath

Table of Contents

Chapter 10

Geometric Path Planning Through Convex Obstacle-free Regions

In order to effectively plan missions for mobile robots over long horizons, the Scotty Planning System needs to be able to deal with obstacles. Unfortunately, the disjunctive constraints required to avoid obstacles are non-convex and, therefore, cannot be incorporated in our convex optimization model in a straightforward manner. In this chapter we present a standalone geometric motion planner that uses an approach based on informed search and convex optimization. In order to find obstacle-free paths, this planner generates paths that are guaranteed to be contained in the union of a set of convex obstacle-free safe regions, which are generated in advance. This planner is not part of the Scotty Planning System. However, we use this chapter for pedagogical purposes, as it provides us with the opportunity to introduce the core ideas of the ScottyPath planner using a simpler problem. On that note, the planner presented in this chapter is purely a geometric path planner. As such, it does not consider multiple vehicles, dynamics, temporal constraints or any state conditions other than visiting goal regions. All these are considered in ScottyPath, which we describe in Chapter 11.

The disjunctive constraints required for obstacle avoidance can be handled with mixed-integer optimization approaches. There exist available solvers capable of handling the mixed-integer version of the cone programs that we use, known as mixed-integer second order cone programs (MISOCPs). However, using MISOCPs solvers to handle obstacles in Scotty is not an effective approach for two reasons. First, mixed-integer problems are NP-complete, and therefore not known to be solvable in polynomial time, unlike cone programs. As a consequence, solving MISOCPs takes, in practice, orders of magnitude longer than SOCPs, and performance degrades quickly the size of the problem increases. This is a significant problem for Scotty since, as we explained before, our optimization program is solved thousands of times for a typical robotics planning problem.

However, this issue is not the only reason why a MISOCP approach is not an appropriate solution to handle obstacle avoidance in the Scotty Planning System. Since we do not discretize time, each segment of the piecewise linear trajectory that we allow for robot motions can take an arbitrarily large amount of time. A typical mixed-integer approach using the well-known *bigM method* ensures that the extremes of such linear segments are obstacle free. However, it cannot guarantee that the robot does not collide with obstacles in between, as intermediate points are not subject to obstacle avoidance constraints.

In this chapter, we propose a path planning approach that relies on convex optimization and informed search. We restrict obstacles to convex polytopes, since this greatly simplifies the problem. Non-convex obstacles can be decomposed into convex components using well-known methods [65]. Instead of avoiding obstacles directly, our planner reformulates the motion planning problem to find, instead, paths over convex obstacle free safe regions. These *safe* regions are convex regions that are guaranteed to be obstacle free and are computed with the IRIS algorithm [30]. This type of motion planning over safe regions has been performed in the past to compute dynamically feasible trajectories for quadrotors using a mixed-integer approach [28]. In this chapter, we present, instead, an approach based on informed search that finds shortest piecewise linear paths going through safe regions. We show in Section 10.4 that our approach is more than two orders of magnitude faster in medium to large environments than mixed integer approaches.

Our motion planner finds obstacle free paths from an initial position to one or

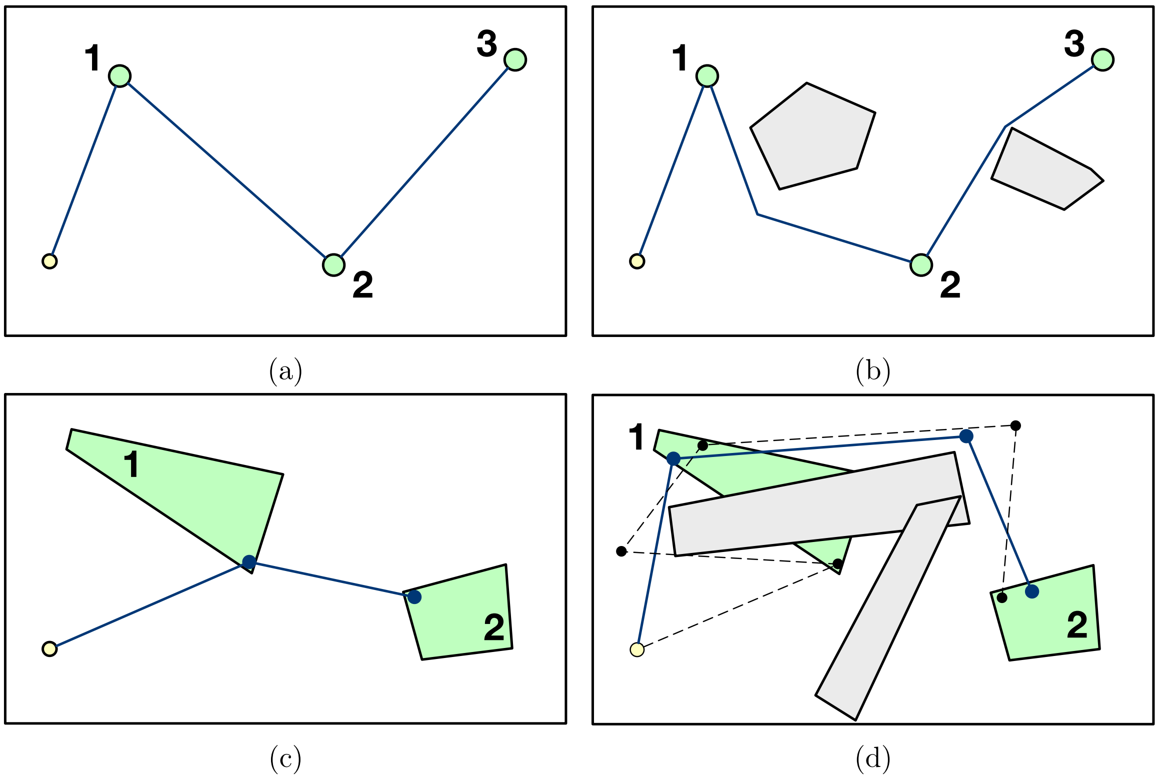


Figure 10-1: Motion planning for multiple goal regions can be done independently when these goal regions are small (a and b). However, when goal regions are large, better plans can often be found by considering all goals jointly. Figure (d) shows the suboptimal path to goals 1 and 2 when planning independently (in black) and a better plan, in blue, when considering both goals jointly.

more goal regions. The standard approach for motion planning when multiple goal regions need to be visited is to plan for each consecutive pair of region goals independently. This works well when the goal regions are small (Figures 10-1a and 10-1b). However, for the typical robotic problems that Scotty solves, these regions can be arbitrarily large. In these cases, planning for pairs of goal regions independently does not work well, as better plans can often be found when considering all the goals jointly (Figures 10-1c and 10-1d). For this reason our motion planner is designed to consider all region goals jointly. As we will explain later, these goals can be ordered or unordered.

Compared to other motion planning approaches, the algorithm presented in this chapter exhibits multiple advantages for our purposes. By remapping the obstacle avoidance problem into the problem of finding paths through convex safe regions, we avoid non-convex disjunctive constraints. As we explain in Chapter 11, this allows ScottyPath to use the same convex SOCP model from Chapter 6 that we use for hybrid activity and motion planning. This is the main reason why this approach has been developed, as it allows us to integrate our Scotty dynamics, continuous effects and temporal constraints within this motion planning framework.

Instead of relying on integer programming to select the obstacle-free safe regions, we use informed search on a graph that describes how these regions are connected. The performance of this approach is orders of magnitude better than the alternative mixed-integer programming approach, as we show in Section 10.4.

As a standalone path planner, our approach does present some disadvantages compared to other standard motion planning approaches. First, we need the convex safe regions to be generated in advance, as will be explained in Section 10.5. Moreover, the shortest paths returned by our approach are only optimal with respect to the coverage of the safe regions. That is, there could be shorter paths avoiding all obstacles and going through areas not covered by the safe regions.

This chapter is structured as follows. We first define our problem statement. We then explain the overall approach informally with an example, and we continue with the formal motion planning algorithm. Next, we introduce alternative mixed-integer approaches for comparison and show in the experimental results section that they are more than two orders of magnitude slower than our planner in medium to complex environments. We end this chapter by discussing methods for generating the convex safe regions in the environment.

10.1 Problem Formulation

As indicated before, the motion planner presented in this chapter solves the problem of finding piecewise-linear shortest paths through convex safe regions from an initial position to one or more goal regions in the environment. We now proceed to introduce the definitions we need before describing our problem statement in detail.

Definition 10.1 (*Bounded convex polytope*)**.** A *bounded convex polytope* is the set resulting from the intersection of a finite number of half-spaces, and is given by

$$R = \{\, \mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} \leq \mathbf{b} \,\} \tag{10.1}$$

, where $A \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^{m \times 1}$.

▲

Definition 10.2 (*Obstacle*)**.** An obstacle, R_o, is a *bounded convex polytope* that our robot cannot intersect with. In order to simplify the planning problem, we follow the popular approach of representing the robot by a point, $\mathbf{x} \in \mathbb{R}^n$ and growing the obstacles according to the robot radius. ▲

Definition 10.3 (*Occupied space*)**.** The *occupied space*, $R_{obstacles}$, is the set defined by the union of all *obstacles.*

$$R_{obstacles} = \bigcup_{R_{oi}} R_{oi} \tag{10.2}$$

▲

Definition 10.4 (*Safe region*)**.** A *safe region*, R_s is a *bounded convex polytope* that does not intersect with any obstacles. ▲

Definition 10.5 (*Safe space*)**.** The *Safe Space* is the set resulting from the union of all the *safe regions.*

$$R_{safe} = \bigcup_{R_{s\,i}} R_{si} \tag{10.3}$$

▲

Note that the intersection between the occupied space and the safe space is null, $R_{safe} \cap R_{obstacles} = \varnothing$.

Definition 10.6 (*Goal region*)**.** A goal region is a *bounded convex polytope* that the robot needs to visit. A goal region may or may not intersect with one or more obstacles or safe regions. ▲

Definition 10.7 (*Environment*)**.** An *environment*, E, is a *bounded convex polytope* that contains all obstacles, all safe regions and all goal regions.

$$\bigcup_{R_{g\,i}} R_{g_i} \cup R_{obstacles} \cup R_{safe} \subseteq E \tag{10.4}$$

▲

Definition 10.8 (Free space)**.** The *free space*, R_{free}, is the part of the environment that does not intersect with any obstacles.

$$R_{free} = E \setminus R_{obstacles} \tag{10.5}$$

Note that $R_{safe} \subseteq R_{free}$. ▲

With these definitions in place, we can now proceed to define the problem that the motion planner presented in this chapter solves.

Definition 10.9 (*Safe-region shortest path problem*)**.** A *safe-region shortest path problem* (SSP) is a tuple $\langle E, O, S, \mathbf{x}_0, G \rangle$ where

- E is an environment.
- O is the set of all obstacles in the environment.
- S is the set of all safe regions in the environment.
- x is the starting location of the robot.
- G is a list of ordered or unordered goal regions that the robot needs to visit.

The solution to a safe-region shortest path problem is a *safe region constrained shortest path* that visits all goal regions (in order or not) and has minimum length. ▲

Definition 10.10 (*Safe-region shortest path problem with ordered goals*)**.** A *safe-region shortest path problem with ordered goals*, SSPo, is a SSP where the goal regions need to be visited in the given order. ▲

Definition 10.11 (*Safe-region shortest path problem with unordered goals*)**.** A *safe-region shortest path problem with unordered goals*, SSPu, is a SSP where the goal regions that need to be visited are unordered. ▲

The motion planner presented in this chapter solves both SSPos and SSPus. The solution to SSP problems are *shortest safe-region constrained paths.* These are defined next.

Definition 10.12 (*Valid path*)**.** A *valid path*, p_{valid}, is a piecewise linear path given by the sequence of points denoting the ends of its linear segments, $\mathbf{x}_1 \ldots \mathbf{x}_n$, that never intersects with any of the obstacles. That is, a valid path is fully contained in the free space, $p_{valid} \in R_{free}$. ▲

Definition 10.13 (*Shortest path*)**.** Given a SSP, a *shortest path* is a valid path that visits all the goal regions (in the given order of not, depending on whether the SSP is an SSPo or an SSPu) and that has minimum length. ▲

Definition 10.14 (*Safe-region constrained path*)**.** A *safe-region constrained path*, p_{safe} is a path that is fully contained in the safe space, $p_{safe} \in R_{safe}$. Since $R_{safe} \subseteq R_{free}$ every safe-region constrained path is also a valid path. ▲

Definition 10.15 (*Shortest safe-region constrained path*)**.** Given a SSP, a *shortest safe-region constrained path* is a safe-region constrained path that visits all the goal regions and that has minimum length. Since every safe-region constrained path is also a valid path but not every valid path is a safe-region constrained path, shortest paths have, in general, shorter length than shortest safe-region constrained paths. ▲

10.2 Approach

In this section we first describe how our motion planner works with an example and then we proceed to describe our planning algorithm in detail.

10.2.1 In a Nutshell

Our motion planner takes a *safe-region shortest path problem* (Figure 10-2a) and returns a *shortest safe-region constrained path.* We assume that the safe regions have already been constructed and placed in the environment (Figure 10-2b), but we discuss in Section 10.5 different methods for constructing such regions. To motivate our solution method, we note that, if we knew an (ordered) sequence of safe and goal regions we want to visit, then the problem of finding the shortest paths reduces to solving a second order cone program, which can be done very efficiently with state of the art commercial solvers. The difficulty of the motion planning problem lies, then, in finding what safe regions to visit and in what order. We find this sequence using informed search, although we also compare our approach with mixed integer

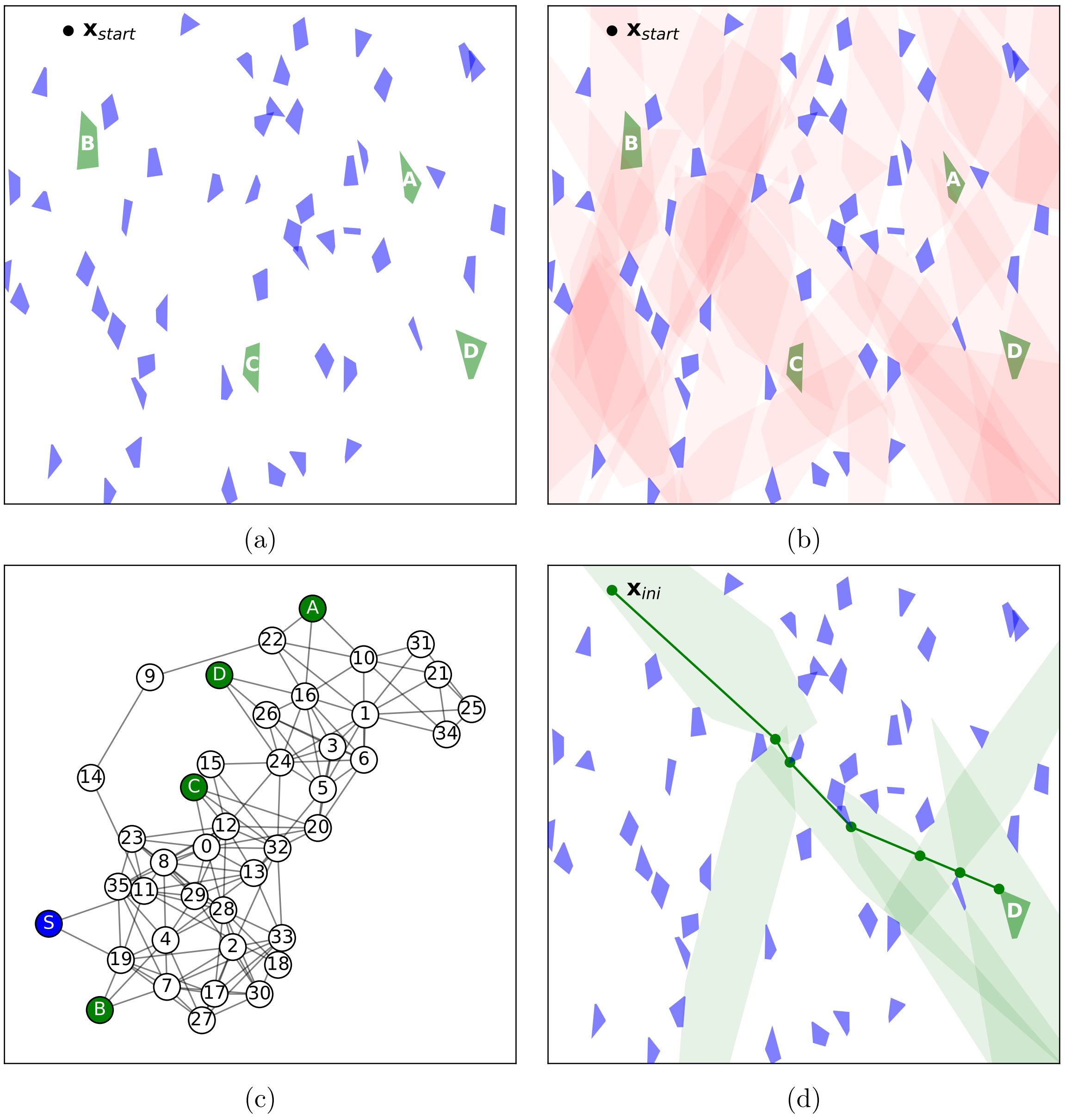


Figure 10-2: Figure a shows the environment with obstacles (in purple) and goal regions (in green). Safe regions (shown in red) are generated in advance (b). The connectivity graph is then computed. This graph (c) shows the connections between safe regions (white nodes), goal regions (green nodes) and the starting position (blue). Using A^* we find the optimal sequence of safe regions to reach goal region D with the shortest safe-region constrained path (d).

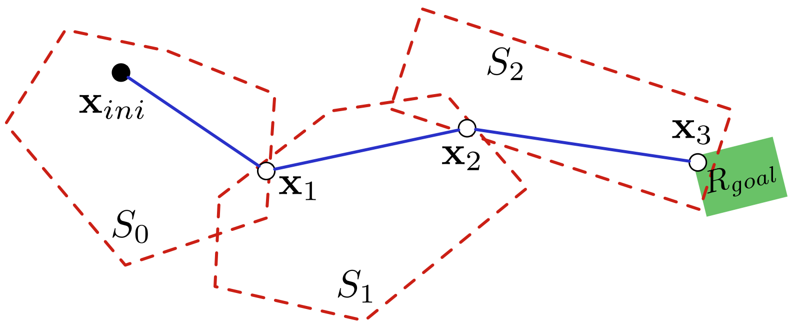


Figure 10-3: Shortest path going through a sequence of safe regions.

approaches. In order to find the best sequence of safe regions, we first create the connectivity graph of safe and goal regions (Figure 10-2c). The connectivity graph is an unweighted graph that indicates how regions are connected: i.e. the graph contains an edge between two regions if these have a non-null intersection. This graph, which is precomputed before the start of the search, prevents us from trying impossible paths between regions that are not connected. In order to find the best sequence of safe regions, we use the A^* algorithm. In order to compute the cost of the selected sequence of paths and the heuristic (*cost-to-go*), we solve a SOCP for each search state. This SOCP computes the shortest path going through the selected safe regions in the current search state (cost), and the straight line paths going through the remaining goals and ignoring obstacles (heuristic). Since our heuristic is admissible, the returned path is the shortest path through safe regions (Figure 10-2d).

10.2.2 Shortest Path through Convex Regions

Given a starting point, a goal region and an ordered sequence of safe regions to go through, the shortest path can be found by solving a second order cone program (SOCP). The shortest path is a piecewise-linear path, where each segment is fully contained in each of the safe regions in the sequence. A path going through n safe regions and reaching a goal region at the end consists of n segments and $n+1$ points. These points are the solution to the SOCP problem. Given that the segments are straight lines and that the safe regions are convex, it suffices to impose that the beginning and the end of each segment are inside its associated safe region in order to guarantee that the full segment remains inside the safe region. This is achieved with the following optimization program in which the decision variables are the $n+1$ points (Figure 10-3), $\mathbf{x}_i \in \mathbb{R}^n$:

$$\min_{\mathbf{x}_i} \sum_{i=0}^{n-1} \|\mathbf{x}_{i+1} - \mathbf{x}_i\| \tag{10.6}$$

$$\mathbf{x}_0 = \mathbf{x}_{ini} \tag{10.7}$$

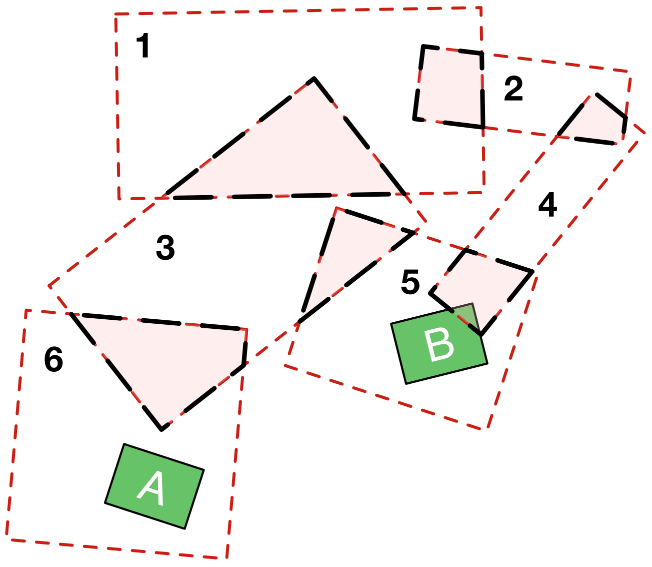
$$\mathbf{x}_n \in R_{goal} \tag{10.8}$$

$$\mathbf{x}_i, \mathbf{x}_{i+1} \in S_i \quad \forall i = 0 \dots n-1 \tag{10.9}$$

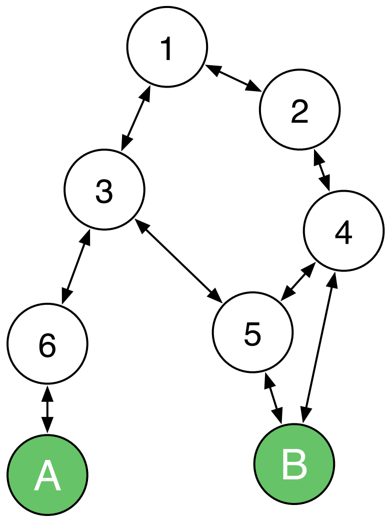
, where $S_0 \dots S_{n-1}$ is the sequence of the n safe regions visited, and the constraint ensures that each of the linear segments of the path is fully contained in each safe region, as discussed earlier. Note that all points except the first and the last are constrained to be in two safe regions simultaneously. This is equivalent to imposing that these points lie in the intersection between the consecutive safe regions. In order to find the shortest path, we minimize the sum of the lengths of all the segments in the paths. These lengths are expressed as Euclidean norms and can be represented with cone constraints. Therefore, the mathematical program presented before is a second order cone program (SOCP). Since SOCPs are convex optimization problems, our optimization problem can be solved very efficiently and the solution returned by the solver is optimal. As indicated earlier, the difficulty in the motion planning problem that we are trying to solve lies in finding what safe regions to visit and in what order. Once that is determined, we can extract the shortest path as indicated in this section optimally and very efficiently.

10.2.3 Connectivity Graph of Safe Regions

As explained earlier the first step in our motion planning algorithm involves creating the connectivity graph that expresses how the safe and the goal regions are connected.



(a)



(b)

Figure 10-4: Safe regions and resulting connectivity graph

Algorithm 10.1: CONNECTIVITY-GRAPH

Input: A set of convex safe regions (S) and a set of convex goal regions (R_g).
Output: A bidirectional graph, $G = \langle V, E \rangle$, with bidirectional edges between pairs of regions whose intersection is not null.

Algorithm

```
1  $V \leftarrow S + R_g$
2  $E \leftarrow \{\}$
3  $n_S \leftarrow |S|$
4  $n_g \leftarrow |R_g|$
5  for $i$ in $0 \dots n_S - 1$ do
6      for $j$ in $i + 1 \dots n_S - 1$ do
7          if INTERSECTS?($S_i$, $S_j$)
8              $E \leftarrow E + \langle S_i, S_j \rangle$
9      for $k$ in $0 \dots n_g - 1$ do
10         if INTERSECTS?($S_i$, $R_{g_k}$)
11             $E \leftarrow E + \langle S_i, R_{g_k} \rangle$
12 return $\langle V, E \rangle$
```

This can be done offline and reused in an environment with constant safe and goal regions where multiple paths may be queried with either different starting positions or different goal regions.

In order to create the connectivity graph, we test the pairwise intersections of all safe regions among each other, as well as the intersections between safe regions and goal regions (Algorithm 10.1). Note that we do not need to compute the actual intersection between the regions, but only determine whether this intersection is null or not (Line 7 and 10). This can be done efficiently with state of the art computational geometry algorithms [43]. The complexity of the connectivity graph algorithm is quadratic in the number of safe regions. However, due to the efficiency of modern computational geometry algorithms we find that the connectivity graph can be created very fast even in environments with hundreds of safe regions, and that the computation time is very small compared to the time that it takes to find a path later through this regions. This will be shown later in our experimental results section.

10.2.4 Informed Search over Convex Safe Regions

As discussed earlier we use informed search (the A^* algorithm) and the connectivity graph to find the optimal sequence of safe regions that the robot needs to go through in order to visit the goal regions (Algorithm 10.2). The algorithm presented in this section solves SSPo problems (in which the regions goals are ordered). Section 10.2.4.2 discusses the changes needed to solve SSPu problems (with unordered region goals).

We frame this problem as the search for the optimal sequence of safe regions over the connectivity graph until all goal regions have been visited. Each search node in the search tree contains the sequence of regions visited up to that point. These regions can be either safe or goal regions. The search finishes when we find a search node that visits all goal regions. We start the search with a priority queue containing a search node for every safe region that contains the initial position $\mathbf{x}_{ini}$ (lines 2 to 6). Each search node has its associated cost (g), which is the length of the path going through the visited regions up to that node. Nodes also have their heuristic value (h), which is an estimate of the length of the remaining path in order to visit all remaining

Algorithm 10.2: PLAN-REGION-PATH

Input: The starting point $\mathbf{x}_{ini}$, the connectivity graph G, a sequence of goal regions to visit R_g and a heuristic weight $\alpha \geq 0$.

Output: A safe-region constrained path $\mathbf{x}_{ini}, \mathbf{x}_1 \ldots \mathbf{x}_n$ visiting the goal regions, and the sequence of the safe regions that contain the path, if it exists. This path is a shortest safe-region constrained path if $\alpha \leq 1$.

Algorithm

```
1  $Q \leftarrow$ CREATE-PRIORITY-Q()
2  $R_{containing} \leftarrow$ GET-CONTAINING-SAFE-REGIONS($\mathbf{x}_{ini}$)
3  for $r$ in $R_{containing}$ do
4      $N_{new} \leftarrow$ MAKE-NODE($\mathbf{x}_{ini}$, $R_g$, $r$);
5      $\langle g, h \rangle \leftarrow$ COMPUTE-G-H($N_{new}$)
6      PUSH($Q$, $g + \alpha \cdot h$, $N_{new}$)
7  while not IS-EMPTY($Q$) do
8      $N \leftarrow$ POP($Q$)
9      if not VISITED?($N.r_{last}$)
10         MARK-VISITED($N.r_{last}$)
           if $N.remaining\_goals = \varnothing$
11             return $\langle N.path, N.region\_sequence \rangle$
12         $R_{reach} \leftarrow$ SAFE-REGION-NEIGHBORS($G$, $N.r_{last}$)
13         if REACHABLE($G$, $N.r_{last}$, $N.next\_goal$)
14             $R_{reach}$ +=$N$.next_goal
15         for $r$ in $R_{reach}$ do
16             $N_{new} \leftarrow$ EXTEND($N$, $r$)
               if $N_{new}$ and not VISITED?($r$)
17                 $\langle g, h \rangle \leftarrow$ COMPUTE-G-H($N_{new}$)
18                 PUSH($Q$, $g + \alpha \cdot h$, $N_{new}$)
19 return $\langle nil, nil \rangle$
```

Algorithm 10.3: Extend

```
Input: A search node, $N$, and a new safe or goal region to visit, $r$.
Output: A new search node, $N_{new}$, going through region $r$.
Algorithm
1  $N_{new} \leftarrow$ CLONE($N$)
2  if $r$ is SAFE-REGION
3      APPEND($N_{new}$.visited_regions, $r$)
4      $N.r_{last} \leftarrow r$
5  elif $r$ is GOAL-REGION
6      $r_{prev} \leftarrow N.r_{last}$
7      APPEND($N_{new}$.visited_regions, $r$)
8      APPEND($N_{new}$.visited_regions, $r_{prev}$)
9      REMOVE($N_{new}$.remaining_goals, $r$)
10 return $N_{new}$
```

goal regions after this node. Usually graph search problems assign fixed costs to the edges of the graph and these costs are accumulated when traversing the graph in order to determine the cost of reaching a given state through a path. Our problem is more complicated in that we cannot assign fixed costs to the edges of the connectivity graph. The reason is that the edge weights along a path are not independent of their sibling edges. Each edge weight is a function of the points selected within the two regions, and these points are shared by adjacent edges. Edges represent intersections of regions and these intersections can be arbitrarily large. The cost of reaching some region through a sequence of regions changes heavily depending on the path taken. That is, the same edge that connects two safe regions in the connectivity graph can have very different costs depending on the path before getting to that edge.

In order to solve this issue, we use an optimization based approach that jointly computes the cost g and heuristic h for each search node by solving a single second order cone program (line 17). This optimization problem, described in Section 10.2.4.1, finds the minimum length path going through the visited safe regions in the search node, and reaching the remaining goals while ignoring obstacles from that point onward. Since the heuristic computed this way ignores obstacles, it represents the minimum possible length of the path through the remaining goal regions, and it constitutes an admissible heuristic. The paths returned by our algorithm are, therefore, optimal when the heuristic weight, α, is smaller or equal than 1.

In order to perform informed search nodes are expanded as follows. Each node can be extended to visit neighboring safe regions (those connected to the last visited safe region) or the next goal region if it is connected to the last visited safe region in the search node (lines 12 to 14). This is shown in Algorithm 10.3. When a search node is extended with a safe region, this algorithm simply appends the safe region to the list of visited regions and updates the node's last region field. However, goal regions are handled differently. First, whenever a new goal region is visited, this algorithm removes the goal from the list of remaining goals to satisfy (line 9). Moreover, since goal regions may intersect with obstacles, this algorithm not only appends the goal region to the list of visited regions, but it also appends the last visited safe region as well. This ensures that the segment visiting the goal region will also be contained in the previous safe region, which ensures that the path will go through the goal region without intersecting with any obstacles.

Finally, we avoid exploring regions that have already been reached by using a visited list. In order to preserve the optimality of our algorithm, regions are only marked as visited once they are pulled from the priority queue (line 12). Since our planner is designed to find shortest paths visiting multiple goal regions, the same safe region may have to be visited more than once within the full plan (e.g. going back through some safe regions after visiting a goal region in a corner). Therefore, safe regions are only marked as visited with respect to the sequence of goals already satisfied. That is, the same safe region can be visited again after visiting some goal region. For example, safe region R_7 may be first reached and marked as visited by a search node that describes a path that has already visited goals G_1 and G_2. Other search nodes containing paths that have also visited goals G_1 and G_2 will not be allowed to visit R_7, since it has already been marked as visited for those goals. However, a search node that has visited only G_1 or another that has visited goals G_1, G_2 and G_3 will be allowed to visit R_7, since this safe region has not been marked as visited for those goals.

We now proceed to describe how the cost (g) and heuristic (h) are computed for

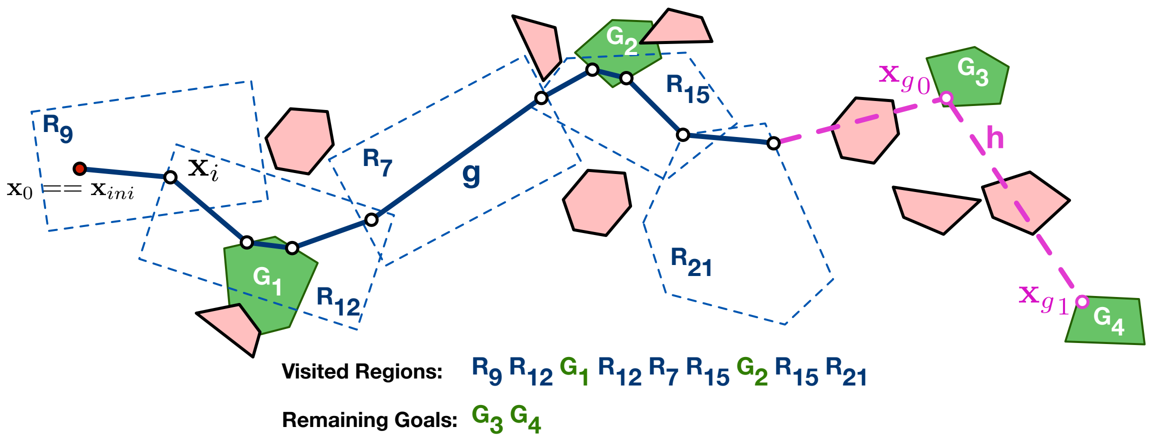

R_9
$\mathbf{x}_0 == \mathbf{x}_{ini}$
$\mathbf{x}_i$
G_1
R_{12}
R_7
g
G_2
R_{15}
R_{21}
$\mathbf{x}_{g0}$
G_3
h
$\mathbf{x}_{g1}$
G_4
Visited Regions: R_9 R_{12} G_1 R_{12} R_7 R_{15} G_2 R_{15} R_{21}
Remaining Goals: G_3 G_4

Figure 10-5: Computation of g and h for a search node.

each node by solving an optimization program and the changes that are required to plan paths visiting unordered goal regions.

10.2.4.1 Computing the Cost and Heuristic through Convex Optimization

As described earlier, we use a single second order cone program to compute both the cost and the heuristic of each search node. In this optimization problem we minimize the sum of the length of the path going through the visited regions (i.e. the node cost, g) and the lengths of the straight line segments going to the remaining goal regions while ignoring obstacles (i.e. the heuristic, h). An example is shown in Figure 10-5 and Algorithm 10.4 describes how this program is built. The path going through the visited regions is a piecewise-linear path that has as many segments as visited regions. Each segment is constrained to be contained inside its associated visited region by constraining its extreme points (lines 5 to 8). As explained earlier, our node expansion function (Algorithm 10.3) forces the same previous safe region to be visited again after a goal region is visited. Therefore, the extremes of each segment visiting a goal region are also contained in that safe region. This ensures that the segments going through goal regions are fully contained within a safe region and, are therefore, obstacle free. This is the case even when goal regions intersect with obstacles. This allows us to treat goal and safe regions equally when building the

Algorithm 10.4: COMPUTE-G-H

```
Input: A search state, $S$.
Output: The path cost through the visited regions in state $S$, $g$, and the
        cost-to-go through the remaining goal regions, $h$.
Algorithm
$P \leftarrow$ BUILD-PROGRAM()
1  $n_r \leftarrow |S.\text{visited\_regions}|$
2  for $i$ in $0 \ldots n_r$ do
       $\mathbf{x}_i \leftarrow$ADD-VAR($P$, $d$)
3  ADD-CONSTR($P$, $\mathbf{x}_0 = \mathbf{x}_{ini}$)
4  $g \leftarrow 0$
5  for $i$ in $0 \ldots n_r - 1$ do
6      $R_i \leftarrow S.\text{visited\_regions}(i)$
7      ADD-CONSTR($P$, $\mathbf{x}_i \in R_i$)
8      ADD-CONSTR($P$, $\mathbf{x}_{i+1} \in R_i$)
9      $g \leftarrow g +$ LENGTH($\mathbf{x}_i$, $\mathbf{x}_{i+1}$)
10 $h \leftarrow 0$
11 $n_g \leftarrow |S.\text{visited\_regions}|$
12 $\mathbf{x}_{prev} \leftarrow \mathbf{x}_n$
13 for $j$ in $0 \ldots n_g - 1$ do
14     $G_j \leftarrow S.\text{remaining\_goals}(j)$
15     $\mathbf{x}_{g_j} \leftarrow$ADD-VAR($P$, $d$)
16     ADD-CONSTR($P$, $\mathbf{x}_{g_j} \in G_j$)
17     $h \leftarrow h +$ LENGTH($\mathbf{x}_{prev}$, $\mathbf{x}_{g_j}$)
18     $\mathbf{x}_{prev} \leftarrow \mathbf{x}_{g_j}$
19 SET-OBJ($P$, $g + h$)
20 SOLVE($P$)
21 $S.\text{path} \leftarrow$ GET-VALUE($\mathbf{x}_0 \ldots \mathbf{x}_n$)
22 return $\langle$GET-VALUE($g$), GET-VALUE($h$)$\rangle$
```

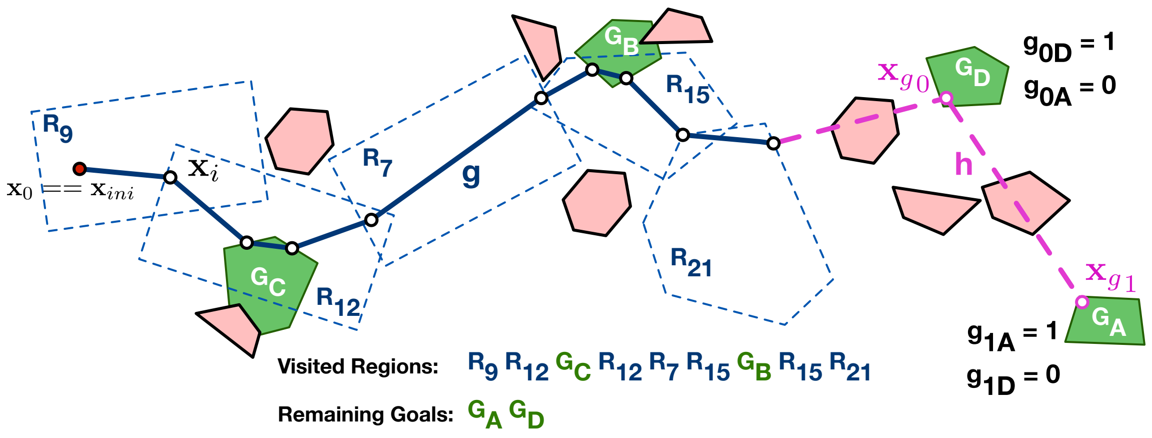


Figure 10-6: Computation of g and h for a search node in the case of unordered goals. The MISOCP program finds that the optimal order to visit the remaining goals is D and then A.

optimization problem that computes g and h.

In order to compute the heuristic part of the path, we place points within the remaining goal regions (lines 12 to 18). Recall that we are now discussing SSPo problems in which the goals are ordered, and therefore we now the order in which these points have to be placed. We describe later in Section 10.2.4.2 how to deal with the unordered goals of SSPu problems. The minimization objective of the optimization problem is the sum of the lengths of all the segments of the path built as indicated before. These lengths are Euclidean norms and can therefore be represented with cone constraints. As a result, the program is, again, a SOCP and can be solved optimally very efficiently.

10.2.4.2 An Extension to Shortest Paths through Unordered Goal Regions

Generalizing Algorithm 10.2 to the unordered problem (SSPu) only requires changes to the node expansion and heuristic computation.

First, the modification to the expansion is minor. In Algorithm 10.2 instead of allowing node extensions to the next goal, we allow extensions to any goal region not visited yet.

Second, the modification to the heuristic computation is more involved. In the case of SSPos, the heuristic portion of the optimization can be computed in a straightforward manner, since the order of the remaining visit goals is known. This is no longer the case for SSPus. In order to compute a heuristic value that is admissible and informative, we use a mixed-integer programming encoding to determine the order of the remaining goal regions that provides the shortest path while ignoring obstacles. In the beginning of this chapter we argued that we wanted to avoid mixed-integer approaches to select what safe regions to visit and in what order. The reason for this is that our algorithm presented in this chapter can handle the combinatorial problem of selecting the safe regions more efficiently by using an informative heuristic and A*. However, we now use mixed-integer programming for a very different purpose. We use integer variables to select the order of the remaining goals while ignoring obstacles, which is a much smaller problem to solve.

To compute the values of the committed cost (g) and heuristic (h) for a search state in SSPus, we construct the optimization program for the part of the path that goes through the visited regions in the same way as in the case of SSPs, as described earlier. For the goals that still need to be visited at that search state, we define one point $\mathbf{x}_{Gi}$ for each of the $i \in 0 \ldots n_G - 1$ remaining goals (Figure 10-6). Since we do not know the order in which the remaining goals need to be visited, we use boolean indicator variables g_{ik} to assign each of the n_g goal points to the remaining goal regions. When boolean variable g_{ik} takes value 1, goal point $\mathbf{x}_{Gi}$ visits goal region k.

$$g_{ik} \implies \mathbf{x}_{Gi} \in G_k \tag{10.10}$$

$$\sum_{k=0}^{n_G-1} g_{ik} = 1 \tag{10.11}$$

$$\sum_{i=0}^{n_G-1} g_{ik} = 1 \tag{10.12}$$

$$\tag{10.13}$$

, where i and k range from 0 to $n_G - 1$. The summation constraints ensure that all goal points have to be assigned to one goal region (10.11), and that all goal regions are visited by some goal point (10.12). As in the case of ordered SSPs, the objective is given as the sum of lengths between consecutive points in the path. The boolean indicator variables take the necessary values to ensure that the order of the remaining goals (ignoring obstacles) is optimal, which provides an admissible heuristic. The presence of the boolean variables turns the previous SOCP program into a mixed-integer second order cone program (MISOCP). In order to speed up the solver, the best order for the remaining goals returned by the solver for a search node is used as a starting solution for the optimization problem of its children search nodes.

10.3 Alternative Mixed-Integer Approaches

Finding shortest safe-region constrained paths involves selecting what safe regions to visit and in what order. This is a highly combinatorial problem in which an objective, the path length, has to be minimized. For these problems, mixed-integer approaches have proved to be very successful. Therefore, we present here two mixed-integer encodings that find shortest safe-region constrained paths and we use them to benchmark our informed search approach. We show that even the best mixed-integer approach presented here is more than two orders of magnitude slower than our method.

We first present a simple encoding and we the proceed to describe a more advanced encoding. The more advanced encoding uses the same connectivity graph described in Section 10.2.3 and a starting guess for a solution to the problem. In order to find shortest paths, these approaches need to minimize segment lengths which are represented with cone constraints. Therefore, these optimization problems are mixed-integer second order cone programs (MISOCPs).

10.3.1 A simple MISOCP encoding

Both mixed-integer methods presented in this section find shortest safe-region constrained paths by solving a one-off optimization program. Optimization problems have fixed size, and it needs to be fixed in advance. We parameterize the size of the optimization problem in terms of the number of segments of the piece-wise linear path, n_s. Since each segment is constrained to be fully contained in a safe region, paths with a larger number of segments can have a shorter length than those with a smaller number of segments. Having to pick the number of segments in advance is an important drawback of this methods compared to our informed search approach that automatically finds the number of required segments automatically. A popular approach to handling this problem involves iteratively solving problems of increasing size until a solution is found. However, in this section we only describe the encodings when the number of segments is chosen in advance.

Given n_s segments, there are $n_s + 1$ extremes of each segment. These points are expressed as $\mathbf{x}_i \in \mathcal{R}^d, \forall i \in 0 \ldots n_s$ and constitute $(n_s+1) \times d$ of the decision variables of the mathematical program. The first extreme of the first segment is constrained to be the initial point of the trajectory, $\mathbf{x}_0 = \mathbf{x}_{ini}$.

We further define boolean variables r_{ij} that assign safe regions to segments. A value of 1 for r_{ij} implies that segment i is contained in safe region j. This is achieved with the following constraints:

$$r_{ij} \implies \mathbf{x}_i \in R_j \tag{10.14}$$

$$r_{ij} \implies \mathbf{x}_{i+1} \in R_j \tag{10.15}$$

$$r_{ij} \in \{0,1\} \quad \forall i \in 0 \ldots n_s - 1, \forall j \in 0 \ldots n_R - 1 \tag{10.16}$$

, where n_R is the number of safe regions. Constraints (10.14) and (10.15) ensure that segment i is fully contained in safe region j. These constraints can be enforced using the well known *bigM method* or using indicator constraints, which most modern solvers support. We enforce that each segment is assigned to a safe region with the following constraint:

$$\sum_j r_{ij} = 1 \tag{10.17}$$

We also define another set of boolean variables g_{ik}. When boolean variable g_{ik} takes value 1, the end of segment i visits a goal region k.

$$g_{ik} \implies \mathbf{x}_{i+1} \in G_k \quad \forall i \in 0 \ldots n_s - 1, \forall k \in 0 \ldots n_g - 1 \tag{10.18}$$

, where n_g is the number of goals.

We additionally impose goal reachability constraints, that only allow g_{ik} to take value 1 if segment i is contained in one of the safe regions that intersects with goal region k. These constraints are not needed, but they facilitate the job of the solver.

$$g_{ik} \leq \sum_{l \in R_{G_k}} r_{il}, \quad \forall i \in 0 \ldots n_s - 1, \forall k \in 0 \ldots n_g - 1 \tag{10.19}$$

, where R_{G_k} is the set of safe regions that intersect with goal region k.

We also impose that all goal regions need to be visited:

$$\sum_i g_{ik} = 1, \quad \forall k \in 0 \ldots n_g - 1 \tag{10.20}$$

For SSPo problems, we further require that the goal regions are visited in the given order using precedence constraints:

$$g_{i,k} \leq \sum_{l=0}^{i-1} g_{l,k-1}, \quad \forall i \in 0 \ldots n_s - 1, \forall k \in 0 \ldots n_g - 1 \tag{10.21}$$

These constraints ensure that in order to satisfy goal k at the end of segment i, the previous goal, $k-1$, must have been satisfied in one of the previous segments. We additionally force that some goal is reached at the end of the last segment:

$$\sum_k g_{n_s-1,k} = 1 \tag{10.22}$$

This constraint is not necessary but we found that it helps the solver significantly, especially in problems with only one goal region.

The objective is the sum of the lengths of each segment:

$$\min \sum_{i=0}^{n_s-1} \|\mathbf{x}_{i+1} - \mathbf{x}_i\| \tag{10.23}$$

These lengths are represented with cone constraints.

The encoding provided in this section can be used to find optimal shortest paths (as good as the ones found with our informed search approach) as long as a sufficiently large number of segments is allowed. Note that using a larger number of segments than the minimum required for an optimal shortest path does not affect the optimality of the path found, since additional segments can be placed in repeated safe regions and segments can have zero length if needed. Increasing the number of segment does increase the difficulty of the problem very significantly as the number of boolean variables increases and mixed-integer problems are known to be NP-complete problems.

10.3.2 A MISOCP encoding using the connectivity graph and a warm start

The simple encoding presented in the previous section can be used to find shortest safe-region constrained paths, and therefore solves the same problem that our informed search approach solves. However, this simple encoding is at a disadvantage compared to our informed search approach since it does not use the connectivity graph that indicates how the safe regions are connected. As we will see in Section 10.4, this simple encoding does not scale well as the number of safe regions increases. In order to better compare our informed search approach with a mixed-integer solution, we present here a more advanced encoding whose performance is greatly enhanced compared to the simple encoding. We do this through two additions. First, we use the common approach of providing an initial suboptimal solution to the solver. This initial solution often helps the solver to find the optimal solution faster. We describe how this initial solution is computed later. Second, we include in this advanced encoding the information of how safe regions are connected from the connectivity graph, as computed in Section 10.2.3. This additional information also helps the solver and increases performance as shown in the experimental results section at the end of this chapter.

Explicitly stating how safe regions are connected is not necessary, since the constraints (10.14) and (10.15) that force points to remain inside safe regions will allow the solver to discover what safe regions are connected. This, however, places the burden of discovering the connections on the solver. In order to free the solver from trying infeasible sequences of safe regions, we add the constraints

$$r_{i,j} + \sum_{l \in \overline{R_j}} r_{i+1,l} \leq 1 \quad \forall i \in 0 \ldots n_s - 2, \forall j \in 0 \ldots n_R \tag{10.24}$$

, where $\overline{R_j}$ is the set of safe regions that does not intersect with safe region j. This constraint ensures that if segment i is contained in safe region j, segment $i+1$ cannot be contained in a safe region that does not intersect with safe region j.

To speed up the search, we also add reachability constraints that state that a safe region j cannot be selected for segment i if it is not possible to reach region j using i segments from the starting point.

$$r_{il} = 0, \forall l \in \overline{D_i} \quad \forall i \in 0 \ldots n_s - 1 \tag{10.25}$$

, where D_i is the set of safe regions that can be reached from the start point in i segments or more and $\overline{D_i} = R \setminus D_i$ is the set of regions that cannot be reached in i segments. In order to compute the set D_i we compute the all the single source shortest lengths from the start position to all the safe regions in the connectivity graph, considering that all edges in the graph (that represent connections between safe regions) have length 1. This can be done very fast for the typical size of problems that we solve, and its computation time is very small compared to the time that it takes to solve the mixed-integer problem.

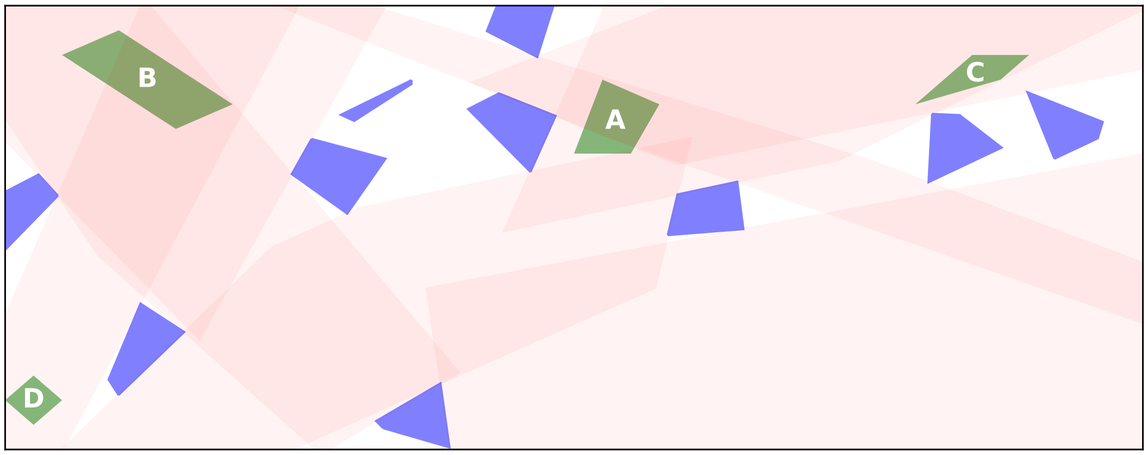
Finally, we compute an initial suboptimal solution to the problem and provide it to the solver in order to warm start its search. In order to compute an initial solution, we simply perform search on the connectivity graph by considering that all the edges have cost 1. Recall that the nodes in the graph represent the convex safe regions and edges between nodes represent that the regions overlap. The solution that we obtain using this method is the one that provides a path going through the smallest number of regions, which is often significantly different from the solution with the smallest length.

The extra constraints added to the advanced encoding described in this section are numerous. Adding these constraints to the program takes a considerable amount of time, as shown at the end of this chapter. However, we show in Section 10.4 that this version of the mixed-integer program performs much better than the version that does not use the information from the connectivity graph or the warm start.

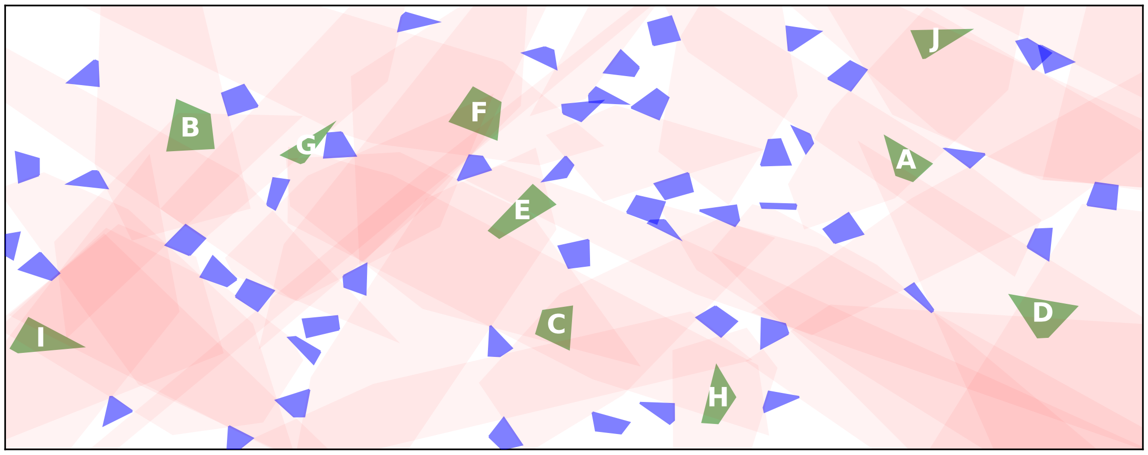
10.4 Experimental Results

In this section we show the results of our path planner as described in Section 10.2 and compare its performance against the mixed-integer approaches shown in Section 10.3. As the results show, our informed search path planner performs over two orders of magnitude faster than the best mixed-integer approach for medium to complex size problems. All the tests were run on an Intel Core i7-3770 3.40 GHz using Gurobi 7.5 as the SOCP and MISOCP solver.

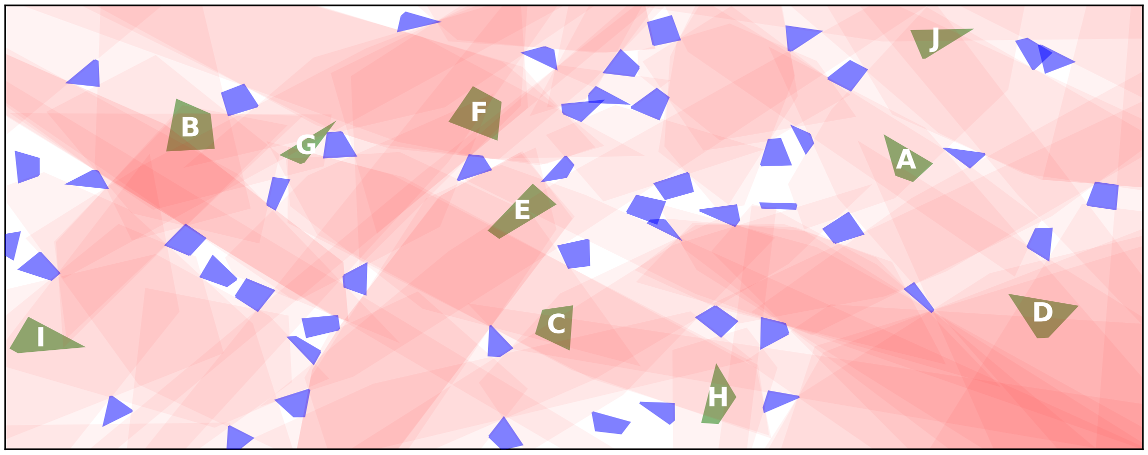
In order to evaluate the planners, we use three scenarios (simple, medium and complex) with different numbers of obstacles and safe regions (described in Figure 10-7). The medium and complex scenarios have the same number of obstacles, and they



(a)



(b)



(c)

Figure 10-7: Test environments, where obstacles are drawn in blue, goal regions in green and safe regions in red. The simple map (a) has 10 obstacles and 8 safe regions covering 90.1% of the free space. The medium map (b) has 50 obstacles and 36 safe regions covering 88.7% of the free space. The complex map (c) is the same as the medium one, but with 75 regions that cover 95% of the free space.

	MISOCP		**MISOCP (adv)**		**A*$_{5\%}$**		**A*$_{0.5\%}$**	
Map	**T**	**L**	**T**	**L**	**T**	**L**	**T**	**L**
Simple Map								
2	0.01	17.82	0.03	17.82				
3	**0.03**	**15.79**	0.05	15.79	0.08	15.79	0.06	15.79
Medium Map								
4	0.30	33.96	0.34	33.96				
5	7.52	33.24	0.69	33.24				
6	(0.6%)	32.84	5.05	32.84	0.28	32.93	**0.34**	**32.84**
Complex Map								
3	0.33	35.68	0.81	35.68				
4	(4.3%)	34.24	1.39	33.92				
5			16.36	33.09				
6			43.26	32.84	**1.19**	**32.84**	1.40	32.84

Table 10.1: Single goal results. Each row shows the planning results for the specified number of segments. Informed search approaches find the needed number of segments for the optimal path automatically. Column **T** shows the planning time in seconds and **L** the length of the returned path. For the MISOCP approaches, results in parenthesis show the MIP optimality gap that the solver was able to prove before the timeout.

are arranged in the same way. However, the complex scenario has over twice the number of safe regions (75 vs 36) than the medium one, which makes the problem harder, since we are finding paths over safe regions. We evaluate the performance of our planner and the mixed-integer planners in these scenarios using single goals, multiple ordered goals and multiple unordered goals.

For every test, we used a timeout of 600 seconds. As indicated in Section 10.3, the mixed-integer approaches need to know in advance how many segments are needed, which is an important drawback over our informed search approach. Paths with lower number of segments are easier to find, at the cost of having a longer length than paths with more segments. In order to evaluate the scalability of the mixed-integer approaches, we try to solve each problem multiple times with different number of segments. We start with the lowest possible number of segments that provides a feasible path and we end with the number of segments needed to find the optimal shortest path. We determine such number of segments for the optimal shortest path by solving

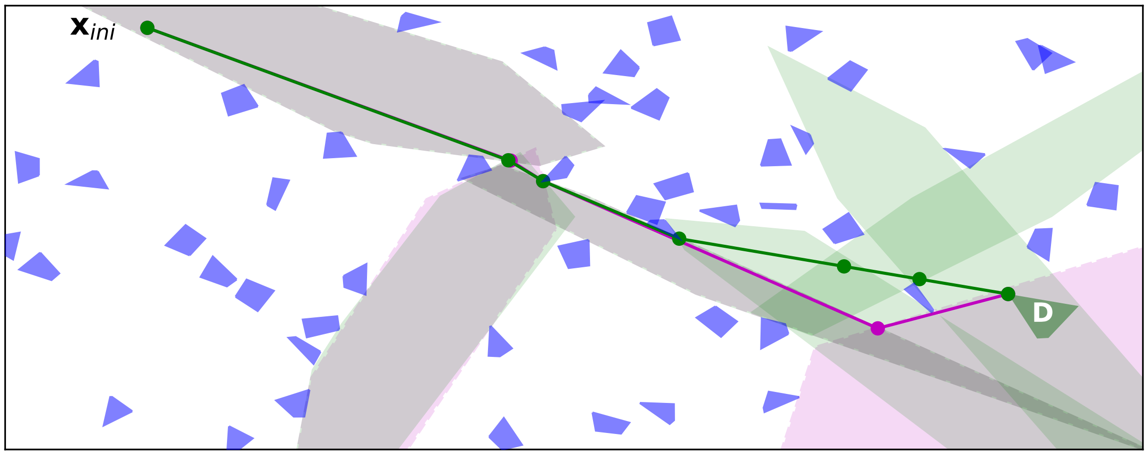


Figure 10-8: Example results of a single goal problem in the complex map. The optimal A* approach finds the shortest path consisting of six segments in 1.40 seconds (green). Within the same time, the advanced MISOCP approach can only find a (longer) path with four segments (magenta). The regions selected by each approach are shown in the same color.

the problem with our informed search approach in advance. For the mixed-integer approaches, we stop trying to solve problems with a certain number of segments when the same problem with one fewer segment times out in 600 seconds. The MISOCP planners are stopped as soon as the solver finds a solution with a proven optimality gap of 0.5%, which means that the solution returned has a length within 0.5% of the optimal shortest length for the given number of segments. In our results we show the performance of our informed search approach using two optimality settings. The first one (labeled $\mathbf{A^*}_{0.5\%}$) uses weighted A* with a weight of 1.005 ($f = g + 1.005h$). This means that the returned path is guaranteed to be within 0.5% of the optimal shortest path. We also instruct the MISOCP solver to return a solution as soon as it proves that its length is within the same 0.5% of the optimal shortest length. The difference is that our informed search approach does not need to know the number of segments in advance and finds the shortest path regardless of how many segments are needed. For the purposes of this section, we will call this our optimal informed search approach. The second setting that we use for our informed search approach (labeled $\mathbf{A^*}_{5\%}$) uses a more aggressive heuristic weight of 1.05. This planner finds paths faster at the expense of only being able to guarantee returned paths whose length is within

5% of the optimal shortest one. We show the performance of this planner to illustrate that by sacrificing some optimality we can greatly improve the performance of our planner.

The results for a single goal are shown in Table 10.1. As we can see, for small problems (e.g. Simple Map) the basic mixed-integer approach (labeled as MISOCP) is the fastest method. However, this is only the case for very small problems. For medium and large problems the performance of the basic mixed-integer approach does not scale well. In the medium map, the basic mixed-integer approach is not even able to find the shortest path with six segments before the timeout. As seen in the table, the more advanced mixed-integer approach presented in Section 10.3.2 (labeled MISOCP (adv)) scales much better than the basic approach. This is due to the fact that this more advanced encoding uses the connectivity graph and a starting MIP solution that guides the search. In this same map, our optimal informed search approach is able to find the shortest path more than an order of magnitude faster. In the complex map, the basic MISOCP approach can only find the path with three segments within the 600 second timeout. The optimal path has six segments and, again, our optimal informed search approach can find the optimal solution more than an order of magnitude faster than the advanced MISOCP approach. Within the time that it takes our optimal informed search approach to find the optimal path (that has six segments), the advanced MISOCP can only solve the problem with 4 segments, which provides a significantly longer path (Figure 10-8).

The results for ordered multiple goals are shown in Table 10.2. The difference in performance between the informed search approaches and the mixed integer approaches is more pronounced in this case. The informed search approaches perform significantly faster even in the simple map that only has a few safe regions. The basic MISOCP approach cannot even find any solution within the timeout in the case of four goals or higher in the complex map. The advanced MISOCP approach cannot find any solution within the timeout in the case of the complex map with six goals. The mixed integer approaches can only find optimal solutions (within the time limit) having the number of segments required for the shortest path in the case of the sim-

	MISOCP		**MISOCP (adv)**		**A*$_{5\%}$**		**A*$_{0.5\%}$**	
Map	**T**	**L**	**T**	**L**	**T**	**L**	**T**	**L**
Simple Map								
(3 goals)								
8	1.59	47.44	0.16	47.44				
9	2.14	45.41	0.84	45.41				
10	1.11	44.82	6.24	44.82			**0.37**	**44.82**
11					0.37	44.86		
Complex Map								
(2 goals)								
6	4.16	59.13	1.48	59.13				
7	(6.8%)	58.49	9.76	58.49				
8			18.70	57.89				
9			78.55	57.20				
10			346.02	56.67				
11			(73.9%)	60.10				
14					2.35	56.56		
16							**5.72**	**56.47**
Complex Map								
(4 goals)								
11	-	-	210.72	78.65				
12			(43.5%)	82.63				
20							**14.95**	**75.70**
21					8.26	75.87		
Complex Map								
(6 goals)								
15	-	-	(57.3%)	120.00				
24							**18.65**	**105.81**
29					13.06	106.01		

Table 10.2: Results for multiple ordered goals. Each row shows the planning results for the specified number of segments. Informed search approaches find the needed number of segments for the optimal path automatically. Column **T** shows the planning time in seconds and **L** the length of the returned path. For the MISOCP approaches, results in parenthesis show the MIP optimality gap that the solver was able to prove before the timeout. Results with '-' denote that the MISOCP solver was not even able to find a feasible solution within the allowed time of 600 seconds.

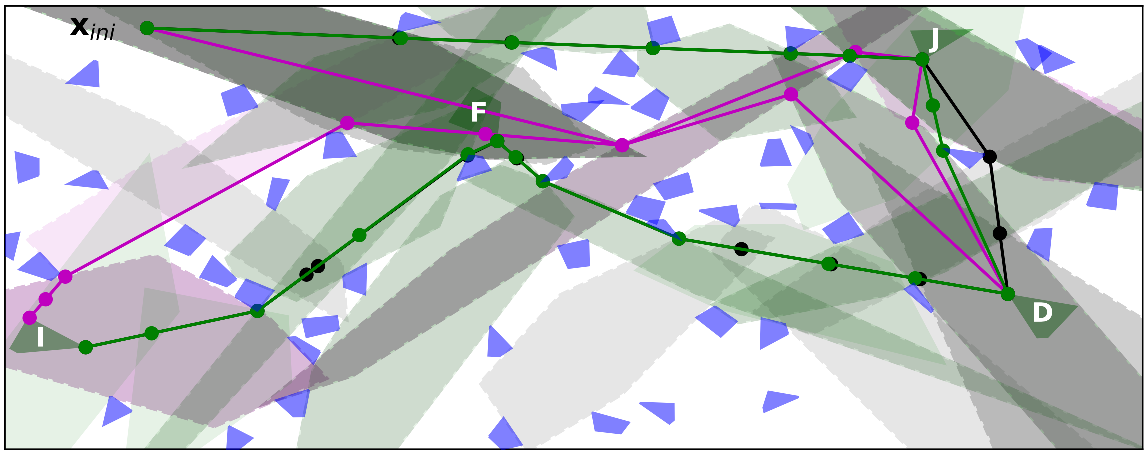


Figure 10-9: Paths for visiting ordered goals J, D, F and I. Within the timeout, the advanced MISOCP planner can only find a path with 11 segments and a length of 78.65 in 210 seconds (magenta). The optimal informed search approach finds the optimal path consisting of 20 segments and a length of 75.70 in 14.95 seconds (green). The weighted A* approach finds a path with 21 segments and a length of 75.87 in 8.26 seconds (black).

ple map. Even when solving problems with a significantly lower number of segments, the advanced MISOCP approach performs more than two orders of magnitude slower than the informed search approaches. For example, it takes the advanced MISOCP approach more than 200 seconds to find a suboptimal path with 11 segments for the complex map with four ordered goals. In the same problem, the optimal informed search approach finds the shortest path with 20 segments in about 15 seconds. The weighted A* approach finds a marginally worse path in just over 8 seconds (Figure 10-9).

Finally, we present the results for the case of multiple unordered goals (Table 10.3). This constitutes the hardest case, and the performance difference between the mixed integer approaches and the informed search one accentuates even more. For the case of two goals in the complex map, the advanced MISOCP can only find a path with 9 segments within the timeout (in over 400 seconds), while the shortest path found by the optimal informed search approach consists of 14 segments and is found in 7.5 seconds (Figure 10-10). For the case of six unordered goals, the MISOCP approaches cannot even find the optimal solution with 15 segments within the timeout. The optimal informed search approach finds the shortest path consisting of 19 segments

	MISOCP		MISOCP (adv)		A*-MIP$_{5\%}$		A*-MIP$_{0.5\%}$	
Map	**T**	**L**	**T**	**L**	**T**	**L**	**T**	**L**
Simple Map								
(3 goals)								
8	11.32	24.76	0.95	24.76	**0.50**	**24.76**	0.57	24.76
Complex Map								
(2 goals)								
6	(3.7%)	45.32	6.45	45.38				
7			23.15	44.89				
8			61.09	44.74				
9			409.91	44.45				
10			(0.9%)	44.45	1.92	44.77		
14							**7.51**	**44.30**
Complex Map								
(4 goals)								
11	-	-	455.10	58.92				
12			(0.9%)	58.90				
14					8.36	58.81		
15							**25.61**	**58.80**
Complex Map								
(6 goals)								
15	-	-	(61.0%)	68.83				
19					29.03	**66.99**	**179.73**	67.00

Table 10.3: Results for multiple unordered goals. Each row shows the planning results for the specified number of segments. Informed search approaches find the needed number of segments for the optimal path automatically. Column **T** shows the planning time in seconds and **L** the length of the returned path. For the MISOCP approaches, results in parenthesis show the MIP optimality gap that the solver was able to prove before the timeout. Results with '-' denote that the MISOCP solver was not even able to find a feasible solution within the allowed time of 600 seconds.

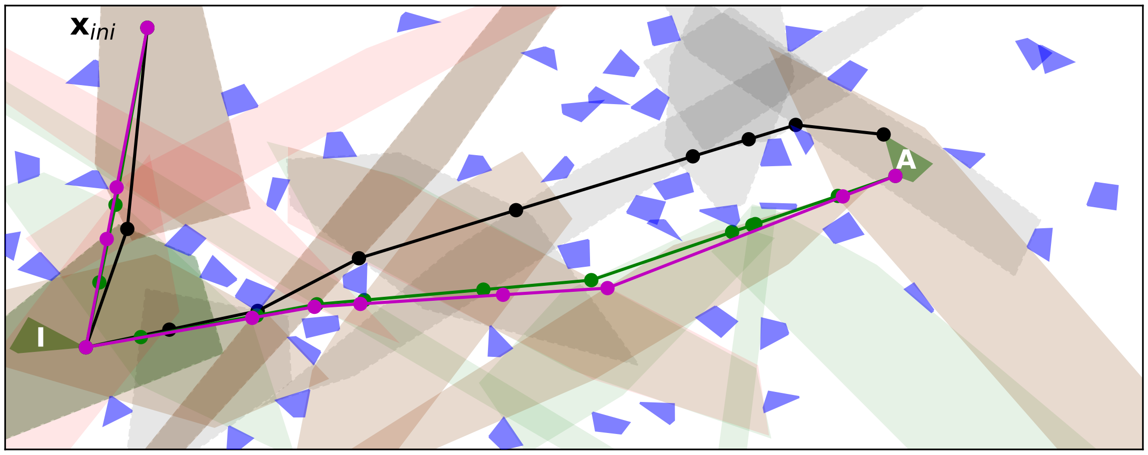


Figure 10-10: Paths for visiting unordered goals A and I. The planners discover that the optimal order is I and then A. Within the timeout, the advanced MISOCP planner can only prove an optimality gap of 0.9% with a path with 10 segments and a length of 44.45 (magenta). The optimal informed search approach finds the optimal path consisting of 14 segments and a length of 44.30 in 7.51 seconds (green). The weighted A* approach finds a path with 10 segments and a length of 44.77 (within 5% optimality) in just 1.92 seconds (black).

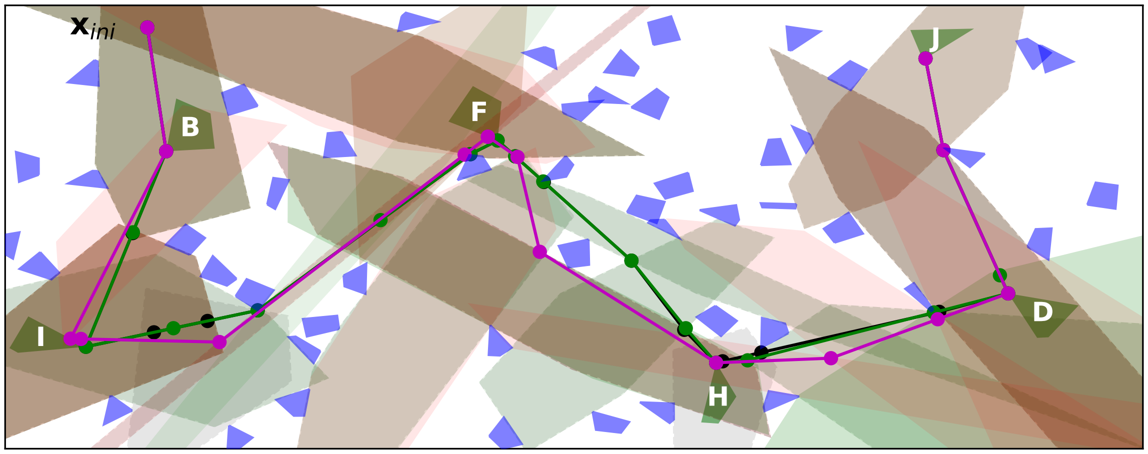


Figure 10-11: Paths for visiting six unordered goals. The planners discover that the optimal order is B, I, F, H, D and J. Within the timeout, the advanced MISOCP planner can only prove an optimality gap of 61% with a path with 15 segments and a length of 68.83 (magenta). The optimal informed search approach finds the optimal path consisting of 19 segments and a length of 67.00 in 179.73 seconds (green). The weighted A* approach finds a different path with 19 segments and and the same length as the optimal in just 29 seconds (black).

in 180 seconds. The weighted A* solution finds, in this case, a different solution of the same length much faster, in 29 seconds (Figure 10-11).

The results presented in this section show that our informed search approach, as presented in Section 10.2, performs more than two orders of magnitude faster than the best mixed integer approach that we have described in Section 10.3.2. Moreover, recall that the mixed integer approaches require that we indicate in advance the number of segments that we allow the path to have. Determining the lowest number of segments needed in order to find a feasible path is not complicated, as we can find such number by running a shortest path algorithm in the connectivity graph considering that all edges in the graph have the same weight. Determining the number of segments needed for the optimal shortest path is harder, though. In general, we can only say that the MISOCP will find the optimal shortest path through the safe regions if we allow the path to have as many segments as safe regions are in the environment. In practice, the shortest path can be found with much fewer segments, as safe regions are typically heavily interconnected, but this cannot be known in advance. Our informed search approach does not present this problem of determining the number of segments needed, as it will explore the connectivity graph in a best-first manner until the optimal path is found. While our optimal informed search approach performs much better than the mixed-integer counterparts, we have also shown with these results that it is often possible to increase the performance very significantly by trading some optimality with an increased heuristic weight. This can be useful in situations in which a 'good enough' path suffices but planning time is critical.

10.5 Generation of Convex Safe Regions

The path planning algorithm presented in this chapter finds collision free paths going through convex obstacle-free regions. So far, we have assumed that these convex obstacle-free regions were already present in the environment. In this section we discuss some possible ways to generate such regions.

We use the IRIS algorithm[30] to generate our safe regions, that are obstacle-free bounded convex polytopes. The IRIS algorithm alternates between finding separating hyperplanes and largest area inscribed ellipsoids until it converges to the largest discovered obstacle free bounded polytope. The IRIS algorithm needs a seed point to start its iterative approach for each new safe region to be generated. Although the seed point is not guaranteed to be inside the final region returned by the algorithm, we have found in our tests that this point is often either inside or close to this region. Therefore, the problem of generating convex safe regions in the environment reduces to selecting the desired seed points in order to then generate the safe regions with the IRIS algorithm.

There are multiple ways to generate these seed points. A straightforward approach consists in having a human operator select the seed points using a point and click interface. While this could be labor intensive in large environments with many obstacles, it is often a fast and precise method for generating safe allowed regions for robotic vehicles. This is important for missions with very expensive or hard to replace assets in which operators want to ensure that robots are only allowed to be in regions that human experts have deemed to be safe. This is the method that the MIT team used in the DARPA Robotics Challenge[35]. In the next section we present an alternative approach that we can use to automatically generate the safe regions.

10.5.1 Automatic Generation of Convex Safe Regions

Since the IRIS algorithm only needs a seed point to generate a safe region, a basic approach for the automatic generation of safe regions consists in randomly sampling obstacle-free points in the environment and using them as seed regions one by one, until the desired safe region coverage is as large as desired. Unfortunately this method is not very efficient. As more safe regions are added to the environment, it becomes increasingly likely that the next randomly sampled seed point is already contained in one of the safe regions. As a consequence, the newly generated region is likely to have a large overlap with other safe regions already in the environment and less likely to cover obstacle-free areas of the environment that were not already covered by other safe regions.

Algorithm 10.5: GENERATESAFEREGIONS

```
Input: An environment $E$ with obstacles and a desired coverage goal, $c_{goal}$
Output: A list of SafeRegions, $S$, that achieves the coverage goal.
Algorithm
1  $P_{\text{boundary}} \leftarrow$ POLY($E.bounds$)
2  $R_{\text{free}} \leftarrow$ SUBTRACT($P_{\text{boundary}}$, UNION($E.obs$))
3  $R_{\text{uncovered}} \leftarrow R_{\text{free}}$
4  $A_{\text{free}} \leftarrow$ AREA($R_{\text{free}}$)
5  $S \leftarrow \{\}$
6  $c \leftarrow 0$
7  while $c < c_{goal}$ do
8      $P \leftarrow$ FIND-LARGEST-BY-AREA($R_{\text{uncovered}}.polys$)
9      $\mathbf{x}_{\text{seed}} \leftarrow$ SAMPLE-POLY($P$)
10     $S_{\text{new}} \leftarrow$ GROW-IRIS($\mathbf{x}_{\text{seed}}$, $E.obs$)
11     if $S_{\text{new}}$ not in $S$
12         $S \leftarrow$ APPEND($S$, $S_{\text{new}}$)
13         $R_{\text{uncovered}} \leftarrow$ SUBTRACT($R_{\text{uncovered}}$, $S_{\text{new}}$)
14         $c \leftarrow 1 - \frac{\text{AREA}(R_{\text{uncovered}})}{A_{\text{free}}}$
15 return $S$
```

This problem can be mitigated in part with Algorithm 10.5. Instead of sampling seed points randomly in the environment, this method keeps track of the obstacle-free areas of the environment that are not covered by safe regions yet, and samples seed points in those areas. We initially compute the obstacle free region in the environment. This is done with computational geometry libraries [43] that compute the subtraction between the bounding polytope of the environment and the union of all the obstacles (Line 2). The result of this operation is often a list of one or more disjoint polytopes. Initially the entirety of the obstacle free region is uncovered by safe regions (uncovered region). We then proceed to find the polytope in the uncovered region with largest area and we sample a seed point within it using rejection sampling. This seed point is used to grow a convex obstacle-free region using the IRIS algorithm (Lines 8 to 10). Since the seed point is not guaranteed to be contained in the generated region, it is possible that this step produces a safe region that has already been generated before. If that is the case, we reject this region and generate a new one with a different sampled point. Otherwise, we add the newly generated safe region to the list of safe regions and we subtract it from the uncovered region. Whenever the ratio of the area covered by the safe regions and the free region exceeds the target value, the algorithm stops.

We have found that this algorithm is able to cover a larger free area than the basic sampling algorithm using a lower number of safe regions. This algorithm works well for low dimensional environments, but it is unlikely to scale well for larger dimensions due to the curse of dimensionality and the fact that keeping track of uncovered disjoint free-areas becomes exponentially harder in larger dimensions. However, this is typically not a problem in many common mobile robotic missions. All the safe regions that we used in the benchmarks in this chapter were generated using this method.

Chapter 11

ScottyPath: Path Planning Through Convex Obstacle-free Regions for Qualitative State Plans

The ScottyActivity planner, as presented in Chapter 8, is capable of performing combined activity and trajectory planning for robotic applications by using an approach that interleaves heuristic forward search and convex optimization. Unfortunately, since commonly used obstacle avoidance constraints are non-convex, ScottyActivity cannot handle obstacles. However, most robotic applications require that obstacles are taken into consideration. In this chapter we present ScottyPath, a planning algorithm designed to solve this problem.

Generating obstacle-free trajectories for ScottyActivity plans is not straightforward when they involve arbitrarily long horizons in which state variables, times and control variables are tightly coupled over the full plan by temporal constraints and robot behaviors. Off-the-shelf path planners cannot be used directly for this purpose. This is the case for two reasons. First, the locations that need to be reached by the robots at different times in the plan may not be explicitly defined. For example, a UAV maintaining a communication wireless link with two ground vehicles may need to stay within a maximum distance from either vehicle, and its trajectory will depend on the trajectory of the other vehicles throughout the entire plan. Second, there may be infinitely many feasible intermediate locations that a robot may visit to satisfy the plan constraints independently, but the full plan needs to be considered jointly in order to provide an optimal obstacle free trajectory.

In this chapter we present *ScottyPath*, a planning algorithm that completes the Scotty Planning System. ScottyPath takes a plan specification, a set of convex obstacle-free safe regions for each vehicle, and an objective, and returns an optimal plan and control trajectory that is obstacle free. ScottyPath generates obstacle free paths by ensuring that each robot always remains within one of its convex obstacle-free safe regions. As in Chapter 10, safe regions are convex polytopes that are generated offline in advance using the IRIS algorithm [30]. The plan specification is given as a qualitative state plan (QSP) [60, 61]. QSPs specify robot goal behaviors in terms of temporal constraints and continuous state and control constraints between points in the plan, called events.

We use an approach that is based on insights from earlier chapters. First, we use informed search to assign vehicles to *convex safe regions* using a method that is analogous to the method described in Chapter 10. Recall that the geometric path planner described in the previous chapter finds obstacle-free shortest paths and does not consider other constraints, such as temporal constraints, dynamics, state constraints or multiple robots. All these are handled within ScottyPath. In fact, ScottyPath is much more general than the geometric path planner presented in Chapter 10, and any SSPo, in which the goals are ordered, can be solved with ScottyPath. Second, we use ScottyConvexPath, described in Chapter 6, in order to find the control trajectory of each vehicle using convex optimization. We also use ScottyConvexPath to compute the heuristic and cost value used during the informed search that assigns the regions without using time discretization. The innovations of our planner lie in how the cost and heuristic for each search node are computed jointly with our convex optimization model and in how intermediate events in the plan are connected sequentially until the full plan is connected through safe regions.

While ScottyPath is designed to take the output of ScottyActivity as input, it can also be used standalone. In fact, we will see in our example scenario in Section 11.1

that describing plan specifications with QSPs is straightforward. ScottyPath is useful in many robotic applications in which obstacle free trajectory planning for multiple robots with complex state and time constraints is needed, but where activity planning is not necessary. ScottyPath is validated on a set of realistic robotic problems. The results, presented in Chapter 12, demonstrate the scalability of ScottyPath to long horizons, multiple vehicles and plans of increasing difficulty.

The structure of this chapter is the following. We present an example problem in Section 11.1 that we use to describe our problem statement, in Section 11.2. In Section 11.3 we explain how the output solutions of the ScottyActivity planner and the SSPo problems from Chapter 10 can be expressed as input problems that ScottyPath can solve. Finally, we describe our approach in Section 11.4.

11.1 Example problem

Our example scenario here is similar to the one we introduced in Section 3.1, and is based on a realistic oceanographic science mission, in which multiple vehicles need to be coordinated. For the sake of simplicity, only two vehicles are used in our example scenario: a ship and a remotely operated vehicle (ROV). We assume that the mission that these vehicles need to complete is specified as a qualitative state plan. This qualitative state plan describes the initial conditions of these vehicles, specifically their starting positions, as well as the state conditions that need to hold at each step of the plan. It also specifies the behaviors that each vehicle uses and when these behaviors are enabled and disabled. This qualitative state plan is the input to ScottyPath. Operators can write qualitative state plans manually, when a high level of control is desired. Otherwise, they can use ScottyActivity to generate these plans autonomously, by only providing a high level description of the end goals and a model of the robots and the environment.

We show our example scenario in Figure 11-1a. As in the scenario presented in Section 3.1, the ROV is initially on-board the ship and both are at the starting location. The ship can navigate, while obeying its velocity constraints, until reaching

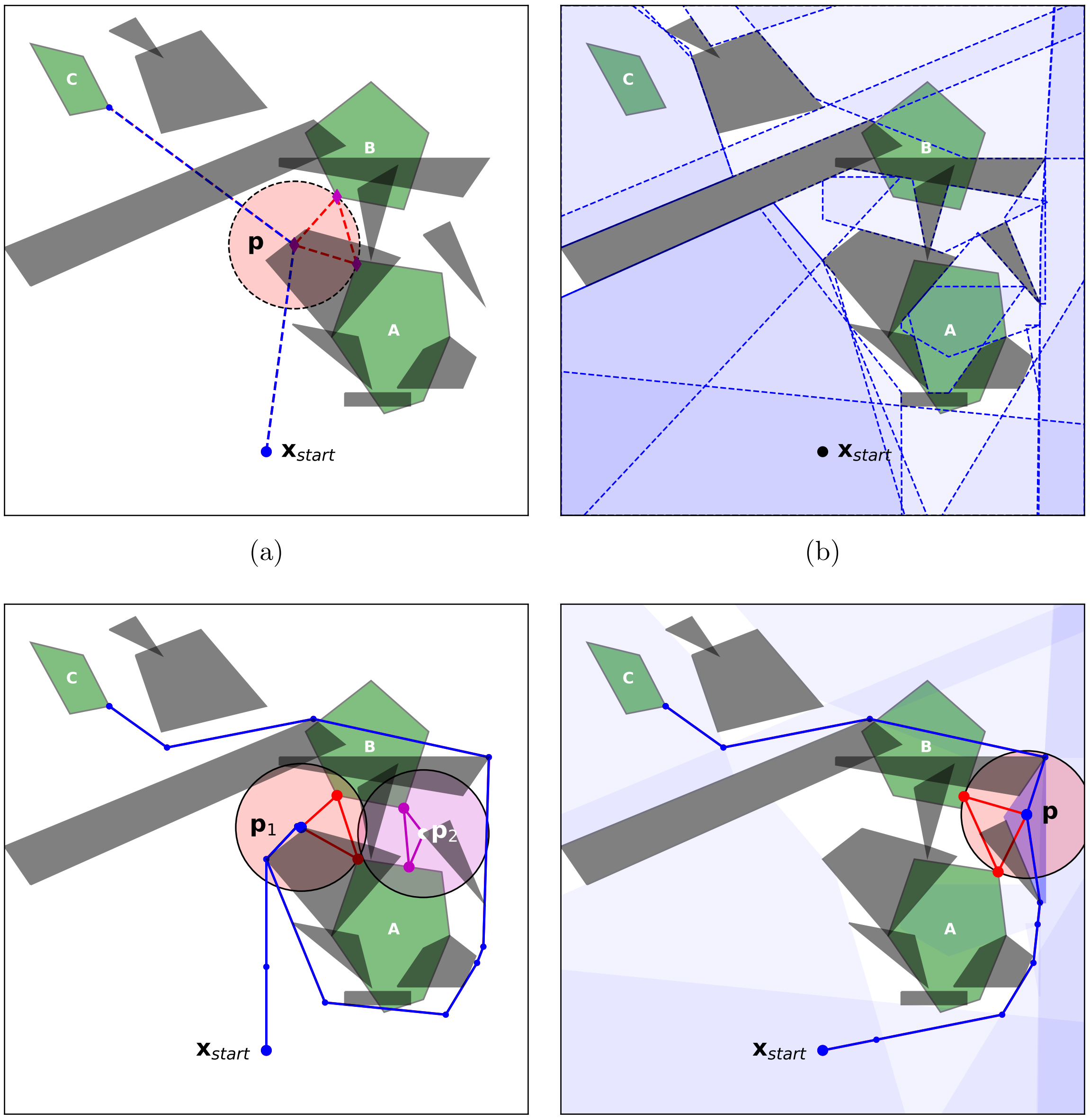


Figure 11-1: Figure (a) shows the example scenario with the surface obstacles for the ship, in black, the ROV sampling regions (A and B) and the end region (C). Figure (a) also shows the optimal solution of this problem when obstacles are ignored. In order to avoid the obstacles, the ship is required to remain in the safe regions (shown in blue) that are constructed offline (b). The problem objective affects the deployment position. In Figure (c), position $\mathbf{p}_1$ is the optimal ROV deployment position that minimizes the time until the first sample is taken. On the other hand, position $\mathbf{p}_2$ is the optimal deployment position in the plan that minimizes the total distance traveled by the ROV. Finally, the plan that minimizes the total distance traveled by the ship and the mission duration is shown in Figure (d). This figure also shows, in blue, the safe regions selected for the ship throughout the mission.

some location where the ROV is deployed. The ship is not allowed to move from the time the ROV deployment starts until the crew finishes recovering the ROV. After the ROV is deployed, the ROV is allowed to navigate submerged on its own, while obeying its velocity constraints and while ensuring that its distance from the ship never exceeds the length of the tether that connects them. During deployment, the ROV must visit region A and then region B to take a sample in each location. While each sample is being taken, the ROV is not allowed to move. Finally, after both samples are taken, the ship must reach region C with the ROV on-board. In order to reduce the cost of the mission, the chief scientist desires a plan that minimizes a linear combination of the distance traveled by the ship and the total time of the mission.

The qualitative state plan does not define specific times when the constraints must be met nor the precise values that the sate variables need to take (such as the precise deployment location). In addition, the qualitative state plan does not define the control trajectory that the vehicles must follow. It defines, however, constraints that must be satisfied by the final plan. Since the qualitative state plan defines what behaviors are used, when they are enabled, and any intermediate constraints that need to be satisfied, activity planning is not required. However, unlike ScottyActivity PDDL-S problems, there are obstacles that need to be considered. In our example the obstacles are surface obstacles that only the ship needs to avoid, since the ROV is always submerged while deployed. Figure 11-1a shows the solution to this problem when obstacles are ignored. This plan, where obstacles are ignored, is the plan that ScottyActivity would return. The planner that we introduce in this chapter finds a plan that meets the same goals and their constraints, while ensuring that the ship avoids the surface obstacles and the mission objective is minimized.

As seen in Figure 11-1a, the plan that ignores the obstacles chooses a deployment position, labeled $\mathbf{p}$, that is inside an obstacle and, therefore, infeasible. Moreover, the ship trajectory goes through obstacles at other times during the plan. Finding an obstacle-free plan is challenging for several reasons. First, an off-the-shelf motion planner cannot be used directly, since the positions that the ship needs to reach are not explicitly defined. For example, the deployment position is affected by the tether range constraint between the ship and the ROV and the fact that the ROV needs to visit both A and B while the ship is stationary. Moreover, there are an infinite number of possible deployment positions that satisfy this constraint, but the space of such positions is not convex due to the existence of obstacles. In order to handle obstacles, we restrict vehicles to remain inside convex *safe regions* that are generated offline. Figure 11-1b shows the *safe regions* that the ship is allowed to remain in. Since the obstacles are in the surface, the ROV only has one safe region, which covers the full scenario. Finally, the problem objective affects the choice for such deployment position. For example, if the chief scientist wanted to minimize the time until the first sample was taken, the resulting plan would look as in Figure 11-1c, where the deployment position is labeled as $\mathbf{p}_1$. Note that while this plan is able to deploy the ROV early, it forces the ship to take a detour that negatively affects the total distance traveled by the ship. Another possible mission objective could aim to minimize the distance traveled by the ROV. The resulting plan in that case is shown in the same figure with the deployment position labeled as $\mathbf{p}_2$. Finally, the plan that minimizes the total distance traveled by the ship and the mission duration is shown in Figure 11-1d.

11.2 Problem Statement

Recall that the problem we solve in this chapter is that of finding a schedule for the qualitative state plan and the control plans for each robot that take them through trajectories that satisfy the goals in the plan, while avoiding obstacles.

The example problem described in Section 11.1 has several characteristics. First, the problem defines a mission qualitative state plan that requires the robots to satisfy certain state constraints at different points in the mission. These are the robot goals. Moreover, this qualitative state plan also describes the behaviors that each robot uses to move around and the times when those behaviors become active. These form a skeleton plan for achieving these goals. Note, however, that this skeleton plan is flexible, as it does not provide the specific times when certain parts of the mission need to happen, nor the precise state trajectories or velocities that the robots need to follow. Instead, it constrains the solution to achieve the goals by using an ordered set of behaviors, and a set of constraints on state, control variables and events. The problem consists in finding a schedule for the events in the plan and the control and state trajectories that allow the robots to satisfy the goals according to the skeleton plan while avoiding obstacles. The problem also specifies the objective that the optimal solution plan needs to minimize.

In this section we provide a formal definition for the ScottyPath problem and its solution.

11.2.1 Qualitative State Plans

Qualitative State Plans (QSPs) specify the goals for the robots along with what behaviors are used to achieve them. It also provides a set of state, control and temporal constraints between events in the plan that solution plans need to satisfy. Additionally, the solution plans need to be obstacle free. ScottyPath achieves obstacle free paths by ensuring that robots remain inside convex safe regions at all times.

QSPs are very expressive, and do not require that the events in the plan are totally ordered. In its current form, ScottyPath solves a restricted form of QSPs in which the events are totally ordered that we call Totally Ordered Qualitative State Plans (tQSPs). We first provide a definition for general QSPs in this section, and we later describe the totally ordered restricted version that ScottyPath solves.

As we explain in detail in the definition, the main element of a QSP are the *episodes*. Episodes are defined between their *start* and *end* events, and specify temporal, state and control constraints that need to be satisfied between those events. They also specify the behavior that a robot uses between those events. These behaviors are defined with *continuous effects*, and can be understood as dynamics that describe how the state variables evolve as a function of the control variables during the episode. Finally, episodes also specify weighted objective terms that need to be minimized. The solution plan that ScottyPath finds minimizes the sum of all weighted objective terms across all the episodes in the QSP.

Recall that ScottyPath takes the output of ScottyActivity as its input. Every activity in a plan returned by ScottyActivity can be represented as an episode in a QSP. However, QSPs are more expressive than what can be represented with the output plans returned by ScottyActivity. We provide in this section the general definition of QSPs and we describe later, in Section 11.3.2, the mapping from the output of ScottyActivity to QSPs.

Our example problem in Section 11.1 is a QSP that defines the goals for the ship and the ROV, the behaviors that are used to achieve these goals and a set of operational state, control and temporal constraints that need to be satisfied between different events in the plan. We now provide a formal definition for QSPs and we show the QSP for our example scenario afterwards. The formal definitions for all the elements in a QSP are provided throughout the rest of this section.

Definition 11.1 (Qualitative State Plan, QSP)**.** A *Qualitative State Plan* (QSP) is given by a tuple $\langle X, C, V, r_v, E_v, E_p \rangle$, where

- X is the set of state variables.
- C is the set of control variables.
- V is a set of n_v *vehicles*.
- $r_v : V \rightarrow R_v$ is a mapping from each vehicle, v, to the set of *convex safe regions* that v is allowed to go through, R_v.
- E is a set of events. Each event e in the plan denotes a point in time in the plan, and is given by the tuple $\langle t_e, S_e \rangle$, where:
  - t_e is a real valued variable that represents the execution time of the event.
  - S_c is a set of state constraints on the values of the state variables at the event, $\mathbf{x}_e = \mathbf{x}(t_e)$.
- E_p is a set of episodes. Each episode connects two events and represents a set

of constraints that need to be satisfied between those events. An episode is given by a tuple $\langle e_S, e_E, d_l, d_u, S_c, C_c, C_e, O\rangle$, where:

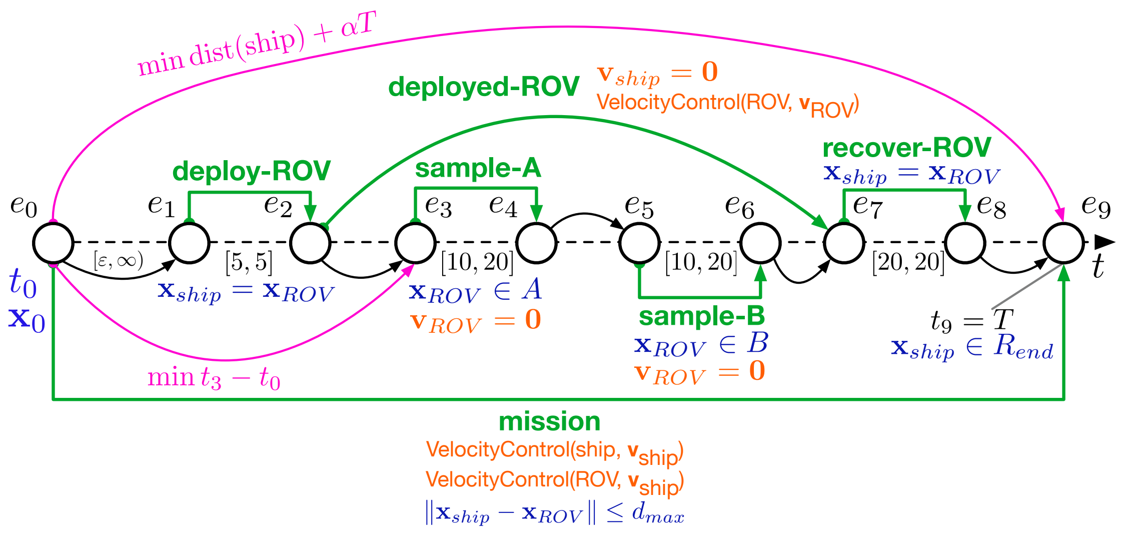
- e_S and e_E are the start and end events of the episode.
- d_l and d_u are the lower and upper bounds on the duration of the episode (i.e. $d_l \leq t_{e_E} - t_{e_S} \leq d_u$).
- S_c is a set of state constraints that the state variables need to satisfy for the duration of the episode.
- C_c is a set of control variable constraints.
- C_e is a set of *continuous effects* that are active for the duration of the episode.
- O is a set of *objective terms* for the episode.

▲

11.2.1.1 Totally Ordered Qualitative State Plans

QSPs, as defined in Section 11.2.1, do not specifically define a total order between events. There may be multiple feasible orderings of events that allow all constraints, temporal or otherwise, to be satisfied. For example, a QSP may specify that a vehicle needs to visit two regions before the end of the mission, but that the order in which those regions are visited is not important.

ScottyPath, in its current form, requires that the events in the QSP are totally ordered. Therefore, we define a restricted form of QSPs called *totally ordered qualitative state plans (tQSPs).* Recall that ScottyActivity explicitly defines the order of the start and ends of the activities in its solution plan. As a result, the solution of ScottyActivity is already a tQSP, and this restriction does not affect the interoperation between ScottyActivity and ScottyPath. We define the input of ScottyPath, a tQSP, as follows.



min dist(ship) + αT
deployed-ROV
$\mathbf{v}_{ship} = \mathbf{0}$
VelocityControl(ROV, $\mathbf{v}_{ROV}$)
deploy-ROV
sample-A
recover-ROV
$\mathbf{x}_{ship} = \mathbf{x}_{ROV}$
e_0
e_1
e_2
e_3
e_4
e_5
e_6
e_7
e_8
e_9
$[\varepsilon, \infty)$
$[5, 5]$
$[10, 20]$
$[10, 20]$
$[20, 20]$
t
t_0
$\mathbf{x}_0$
$\mathbf{x}_{ship} = \mathbf{x}_{ROV}$
$\mathbf{x}_{ROV} \in A$
$\mathbf{v}_{ROV} = \mathbf{0}$
sample-B
$\mathbf{x}_{ROV} \in B$
$\mathbf{v}_{ROV} = \mathbf{0}$
$t_9 = T$
$\mathbf{x}_{ship} \in R_{end}$
min $t_3 - t_0$
mission
VelocityControl(ship, $\mathbf{v}_{ship}$)
VelocityControl(ROV, $\mathbf{v}_{ship}$)
$\|\mathbf{x}_{ship} - \mathbf{x}_{ROV}\| \leq d_{max}$

Figure 11-2: Totally Ordered Qualitative State Plan for the example scenario. Events are represented with circles. The episodes that describe the behaviors of the vehicles and impose constraints on them are shown as green arcs. State constraints are shown in blue, control variable constraints and continuous effects in orange, and episodes with objective terms in pink.

Definition 11.2 (*Totally Ordered Qualitative State Plan, tQSP*)**.** A *totally ordered QSP (tQSP)* is a QSP where the temporal constraints impose that the events are totally ordered. A tQSP has the following additional properties:

- E_l is the event list, given by the ordered sequence of events in the tQSP.
- s is the start event of the tQSP (i.e. the first element of E_l).
- e is the end event of the tQSP (i.e. the last element of E_l) and T is its execution time.
- $\mathbf{x}_{ini} \in \mathbb{R}^n$ is a real-valued vector that denotes the values of all state variables at the start event of the tQSP.
- L is the plan makespan, and is defined as the difference between the execution times of the last event and the starting event in the tQSP: $L = T - t_0$.

▲

Our example scenario, as presented in Section 11.1, is a tQSP problem. The tQSP is shown in Figure 11-2, where the events of the plan are labeled e_0 to e_9 and are represented by circles. The episodes of the plan that impose constraints on the vehicles are shown as green arcs and labels. Episodes that only impose temporal constraints between consecutive events are shown as black arrows. Episodes that only describe objective terms, are shown as pink arcs.

11.2.1.2 State Variables and State Constraints

We use *state variables* to represent the positions of the robots as well as other related variables such as the battery level. ScottyPath operates on the assumption that it cannot change state variables directly. Instead, robot behaviors, that we call *continuous effects*, change the values of the state variables. These continuous effects depend on control variables, whose value is assumed to be directly controllable.

Definition 11.3 (*State Variables*)**.** The vector of *state variables* of the system, $\mathbf{x}$, is given by a vector of real valued variables, $\mathbf{x} = \langle x_1, x_2, \ldots, x_n \rangle \in \mathbb{R}^n$. ▲

As in the case of ScottyActivity, we also use *resource variables* to model values such as the battery level of a vehicle. Recall that, as we describe in Chapter 4, certain continuous effects, such as the ones that depend on norms of control variables, can only be applied to resource variables.

Definition 11.4 (*Resource*)**.** A *resource* is a type of state variable that is subject to special conditions that we describe later in this section. ▲

In the QSP, goal constraints are specified as constraints in the state variables. These constraints can be imposed for the duration of an episode, or at an event. Recall, as described in Chapter 5, that convex constraints can be efficiently imposed over arbitrarily long horizons in our model. For this reason, and in order to use Scotty's Convex Model, as described in Chapter 6, we impose the same restriction that we used in ScottyActivity and limit state constraints to convex quadratic state constraints. However, note that convex quadratic constraints are sufficient to express a wide range of real-world constraints that appear in typical robotic missions, like the ones shown in our example scenario.

Definition 11.5 (*Convex Quadratic State Constraint*)**.** A *convex quadratic state constraint* is a constraint in the form of $g(\mathbf{x}) \leq 0$, where $g : \mathbb{R}^n \rightarrow \mathbb{R}$ is a convex quadratic function operating on the vector of state variables, $\mathbf{x}$. ▲

Additionally, and as described in Chapter 6, we restrict resources to only be subject to *greater or equal than* constraints. This is, again, the same limitation that ScottyActivity imposes.

Our example scenario presents multiple state constraints. The ROV tether constraint, for example, is represented with the convex quadratic constraint: $(x_{ROV} - x_{ship})^2 + (y_{ROV} - y_{ship})^2 - R_{tether}^2 \leq 0$. Additionally, the ROV is required to remain inside each sampling region while the sample is being taken. In this case, the constraints are linear, since the sampling regions are polygons. Recall that any linear constraint is also a convex quadratic constraint [14].

11.2.1.3 Vehicles and Convex Safe Regions

Since ScottyActivity does not consider the obstacle avoidance problem, state variables can be handled independently. In effect, it does not matter whether a state variable denotes the x position of one robot or a different one, as long as it is subject to the right constraints. This is no longer the case in ScottyPath. In order to avoid obstacles, ScottyPath forces each robot to remain within convex safe regions throughout the full plan. We use the term *vehicle* to denote the vector of state variables that defines the position of a robot. Additionally, vehicles can have associated resource variables and a vector of control variables. The former are useful to model, for example, the battery level or the remaining fuel of a vehicle. The vector of control variables is explained in detail in the next section, where we explain that it is used to model the controllable velocities of the vehicle.

Definition 11.6 (*Vehicle*)**.** A *vehicle* v is given by a tuple $\langle \mathbf{x}_v, \mathbf{r}_v, \mathbf{c}_v \rangle$, where

- $\mathbf{x}_v \in \mathbb{R}^d$ is the *position* of the vehicle and is given by a vector of *state variables* where every state variable is one of the state variables of the system.
- $\mathbf{r}_v$ is a vector of *resource variables* associated with the vehicle.
- $\mathbf{c}_v$ is a vector of *control variables* associated with the vehicle.

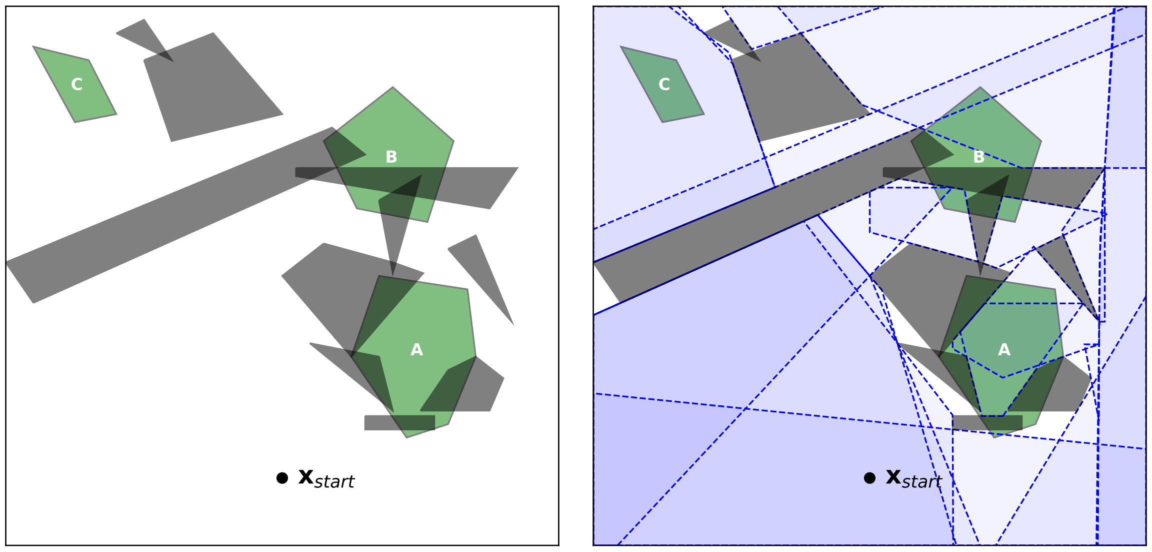
▲

In the example scenario there are two vehicles: the ship and the ROV. The position of each vehicle is a two dimensional vector of state variables (e.g. $\mathbf{x}_{ship} = (x_{ship}, y_{ship})$). In the example, the vector of state variables contains the position variables of both vehicles: $\mathbf{x} = (x_{ship}, y_{ship}, x_{ROV}, y_{ROV})$.

In order to avoid obstacles, vehicles are required to remain within *convex safe regions*. As described in the QSP (Definition 11.1), each vehicle v is assigned a set of convex safe regions R_v through the mapping $r_v : V \to R_v$. The plan returned by ScottyPath ensures that each vehicle is always within one of its assigned convex safe regions. The reason why we allow each vehicle to have a different set of safe regions is that different vehicles may have to avoid a different set of obstacles. In the example scenario, all obstacles are surface obstacles and, therefore, only the ship needs to avoid them. The ROV, on the other hand, is always submerged and can move freely. This represented by the ROV having only one safe region, which represents the full environment.

ScottyPath uses the same convex safe regions introduced in the geometric path planner presented in Chapter 10. Recall that these are convex polytopes that are generated in advance. We provide the definition of convex safe regions here again for clarity purposes.

Definition 11.7 (*Convex Safe Region*)**.** A *convex safe region*, r is a *bounded convex polytope* that does not intersect with any obstacles and that is given by:



(a) (b)

Figure 11-3: Example scenario environment with surface obstacles (in gray) and regions of interest, in green (a). Figure (b) shows, in blue, the convex safe regions for the ship. The ship must always remain within one of those safe regions.

$$R = \{\, \mathbf{x}_v \in \mathbb{R}^d \mid A\mathbf{x}_v \leq \mathbf{b} \,\} \tag{11.1}$$

, where $A \in \mathbb{R}^{l \times d}$ and $\mathbf{b} \in \mathbb{R}^{l \times 1}$, l is the number of hyperplanes (faces) of the polytope, and d is the dimension of the polytope. ▲

Figure 11-3 shows the convex safe regions for the ship in our example scenario. These regions are computed in advance, as described in Chapter 10.

11.2.1.4 Control Variables and Continuous Effects

We assume that a human or an automated planner, like ScottyActivity, has provided us with a behavior-based plan, for achieving goals comprised of a set of active behaviors that are enabled or disabled according to a provided sequence. Each of these behaviors is a continuous effect over time. A continuous effect is active for the whole duration of the episode where it is specified. Recall that ScottyPath can only change the values of state variables by choosing the schedule of the events and the trajectories for the control variables.

Control variables and continuous effects serve the same purpose than in ScottyActivity and, therefore, their definitions in the ScottyPath problem statement are the same. Recall that control variables are bounded continuous controllable parameters.

Definition 11.8 (*Control Variables,* $\mathbf{c}$)**.** The control variables vector, $\mathbf{c}$, is a vector of control variables $\mathbf{c} = \langle c_1, c_2, \ldots, c_m \rangle$, where each control variable c_j is a real valued parameter that is continuously controllable within its fixed lower and upper bounds, c_{j_l} and c_{j_u}. ▲

In the example scenario, each vehicle has its own velocity vector as their associated control variables vector (e.g. $\mathbf{c}_{ROV} = (v_{x_{ROV}}, v_{y_{ROV}})$). In the example, the vector of control variables, is the vector of all these velocities: $\mathbf{c} = (v_{x_{ship}}, v_{y_{ship}}, v_{x_{ROV}}, v_{y_{ROV}})$

As described in Section 11.2.1, episodes in the QSP can define constraints on the control variables. These are useful to model certain operational constraints that may need to be active at certain points in the mission. In the example scenario, there are two convex quadratic control variable constraints that limit the magnitude of the velocities of the ship and the ROV, $\|\mathbf{v}_v\|_2 \leq v_{max_v} \quad \forall v \in \{\text{ship}, \text{ROV}\}$. Moreover, in order to enforce that the ship does not move while the ROV is deployed, there is a control variable constraint that sets the v_x and v_y velocities of the ship to 0 during that time. In order to use the same convex optimization model and solver that ScottyActivity uses, as described in Chapter 6, we restrict control variable constraints to convex quadratic constraints.

Definition 11.9 (*Convex Quadratic Control Variable Constraint*)**.** A *convex quadratic control variable constraint* is a constraint in the form of $g(\mathbf{c}) \leq 0$, where $g : \mathbb{R}^m \rightarrow \mathbb{R}$ is a convex quadratic function operating on the vector of control variables, $\mathbf{c}$. ▲

Continuous effects, as defined in Section 4.1.4, operate on state variables independently. A continuous effect is the model of an engaged behavior. The plan provided by the automated planner or the human describes, qualitatively, when the behavior is engaged and disengaged. A continuous effect is active for the whole duration of the episode where it is specified. A state variable may be influenced by multiple effects simultaneously, and these influences are additive. Recall that we allow three types of continuous effects.

The first type is the continuous controllable linear time-varying effect (CLTE). As described in Definition 4.9, CLTEs are the most common continuous effects and describe a time varying change on a state variable with a rate of change that is a linear combination of control variables. In the example scenario, the *CLTEs* allow the vehicles to move according to their control variable velocities. For example, the two CLTE effects operating each on x_{ship} and y_{ship} allow the ship to move according to its velocity by inducing the rates of change on its position variables: $\dot{x}_{ship} = v_{x\,ship}$ and $\dot{y}_{ship} = v_{y\,ship}$.

As described in Chapter 4, the other two continuous effects that we allow are *resource-constrained norm effects* (RNE) and can only be applied to *resource* variables. RNEs (Definition 4.10) describe a change that depends on the Euclidean norm of a vector of control variables. Similarly to ScottyActivity, ScottyPath supports two types of RNEs: LNEs and LSNEs. Recall that the former describes a change that is proportional to the Euclidean norm of a vector of control variables, while the later describes a change that is proportional to the square of the Euclidean norm. RNEs are useful to model, for example, the battery decrease rate of a vehicle as a function of the norm of its velocity vector.

11.2.1.5 Objective Terms

As described in Section 11.2.1, episodes can also define one or more weighted objective terms. The optimal plan returned by ScottyPath minimizes the weighted sum of all the objective terms across episodes in the QSP.

Each objective term describes some incurred cost throughout the episode where it is specified. We allow multiple types of objective terms. For example, we can define a objective term on the duration of an episode. By placing an episode with such objective term between the start and end events of the QSP, we can instruct ScottyPath to minimize the duration of the solution plan. Other objective terms allow us to minimize the distance traveled by a vehicle or the actuation effort during an episode. The types of objective terms that we allow in ScottyPath mirror those defined for ScottyActivity, in order to be able to use the same second order cone model described in Chapter 6. The objective terms that we allow in the episodes of a QSP are defined as follows.

Definition 11.10 (*Objective Term*)**.** A *objective term* specified in an episode is given by the tuple $\langle k, e_S, e_E, o_t \rangle$.

- $k \in \mathbb{R}$ is the coefficient of the objective term and is given by a real constant.
- e_S and e_E are the start and end events of the episode where the objective term is specified.
- o_t is the function that defines the objective. This function is defined between the start, e_S, and the end event, e_E. It can be one of the following:
  - A *duration objective term*, given as the difference of execution times between the start and end event of the objective term: $o_t(e_S, e_E) = t(e_E) - t(e_S)$.
  - A *differential state variable objective term*, given as the difference of values of a non-resource state variable, x, between the start and end events. The objective term is given by $o_t(e_S, e_E) = x(t(e_E)) - x(t(e_S))$.
  - An *end resource variable objective term*, given by the value of a resource variable at the end event, e_E. As explained in Chapter 6, resource variables can only be maximized. Therefore, we require this objective term to have a negative coefficient ($k < 0$) and the start event to be the start event of the QSP, $e_s = q.s$, where q is the QSP. The objective term is then defined as: $o_t(q.s, e_E) = r(t(e_E)) - r(t_0)$
  - A *path length objective term.* As described in Chapter 4, in order to maintain convexity, these terms can only be minimized. Therefore, we require the term coefficient to be positive, $k > 0$. This term minimizes the distance traveled by a vehicle during the episode. The objective function

is the following:

$$o_t(e_S, e_E) = \int_{t(e_S)}^{t(e_E)} \|\dot{\mathbf{x}}_e(\tau)\| d\tau \tag{11.2}$$

, where $\mathbf{x}_v$ is the vector of state variables that describes the position of the vehicle.

– An *actuation objective term*, that depends on the norms or squared norms of vectors of control variables. Again, this term can only be minimized $(k > 0)$. The objective function is given by:

$$o_t(e_S, e_E) = \int_{t(e_S)}^{t(e_E)} \|\mathbf{c}_e(\tau)\|^{\{1,2\}}\, d\tau \tag{11.3}$$

, where $\mathbf{c}_e$ is a vector of control variables.

▲

The objective that the optimal plan returned by ScottyPath minimizes is then the linear combination of all objective terms of all episodes in the QSP. We call this sum the *objective* of the QSP.

$$\min \sum_{\langle k, e_S, e_E, c_t \rangle \in e.O}^{\forall e \in E_p} k \cdot o_t(e_S, e_E) \tag{11.4}$$

Note that while objective terms are relative in that they are defined between the start and end events of its episode, it is possible to minimize absolute terms by defining objective episodes that start at the starting event in the QSP. The example scenario has two objective terms between the start and the end event of its tQSP. One of them minimizes the total duration of the plan, $T - t_0$. The other is a path length term that minimizes the distance traveled by the ship throughout the mission. Figure 11-2 shows the episodes containing these objective terms as pink arcs in the tQSP. As an example, the figure shows an additional objective that minimizes the time between event e_3 and e_0. This represents the time until the sampling of the first region is started. We use this example objective later in this chapter to illustrate how our algorithm operates.

11.2.2 Solution Plan

Conceptually, the solution to a tQSP is comprised of three elements: the schedule for the events in the tQSP, the trajectories of all the control variables throughout the plan, and a sequence of safe regions, for each vehicle, that describes the obstacle-free path that each vehicle follows.

Definition 11.11 (*tQSP Plan*)**.** A *tQSP plan* is a tuple $\langle f_t, \mathbf{f_c}, \mathbf{r} \rangle$, where

- $f_t : E \to [t_0, T]$ is an assignment of each event in the tQSP to its execution time, where the starting event is constrained to take place at t_0. The execution time of the last event is called T.
- $\mathbf{f_c} : [t_0, T) \to \mathbb{R}^m$ is the control trajectory. The control trajectory assigns a value to all control variables at every time t between the start and the end of the plan.
- $\mathbf{r} : [t_0, T] \to R_{v_1} \times \ldots \times R_{v_{n_v}}$ is an assignment of each time in the plan to a vector that describes the convex safe region that each vehicle is constrained to remain in during that time.

▲

Since state variables only change their values due to the influence of continuous effects, and these only depend on the times they are active for and the values of the control variables, the state trajectory, $\mathbf{x}(t)$, is fully determined in a tQSP plan at all times between the start of the plan, t_0, and the end of the plan, T.

A *valid tQSP plan* satisfies all constraints in the tQSP.

Definition 11.12 (*Valid tQSP Plan*)**.** A *valid tQSP plan* is a *tQSP plan* that satisfies all the constraints in the tQSP. That is,

1. For each event, e_j, the values of the state variables in the tQSP plan at the event, $\mathbf{x}(t(e_j))$, satisfy the state constraints of the event.

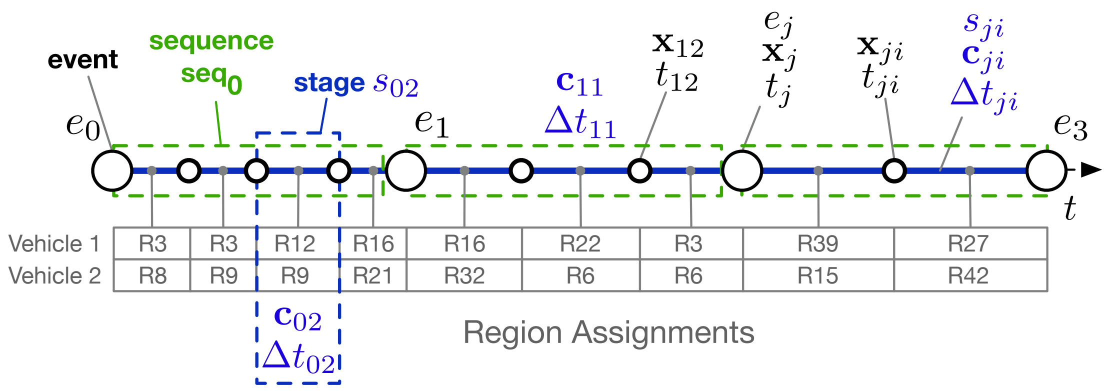


Figure 11-4: Diagram showing a tQSP plan with piecewise constant control with four events. Each consecutive pair of events, e_j and e_{j+1}, are connected by a sequence j (green). Each sequence consists of one or more stages (blue). The control variables vector takes a constant value $\mathbf{c}_{ij}$ during stage i of sequence j. Each vehicle is assigned one of its safe regions at each stage, where it must remain for the duration of the stage.

2. For each episode with start event e_S and end event e_E:

   - The event times of e_S and e_E satisfy the episode temporal constraints.
   - The state variables trajectory, $\mathbf{x}(t)$, satisfies the episode state constraints for the duration of the episode.
   - The control trajectory, $\mathbf{c}(t)$ satisfies the episode control variable constraints for the duration of the episode.

▲

Definition 11.13 (Optimal tQSP Plan)**.** An *optimal tQSP plan* is a *valid tQSP plan* such that the tQSP objective takes the minimum possible value. ▲

11.2.3 Piecewise Constant Solution

Analogous to ScottyActivity, ScottyPath also generates plan with piecewise constant control. Recall, from Chapter 5, that the piecewise constant restriction allows us to efficiently enforce maintain conditions over arbitrarily long durations and to use our second order cone model. Intuitively, in a ScottyPath solution plan, the vector of control variables follows a piecewise constant trajectory. The periods of time in which the vector of control variables takes a constant value are called a *stages*. Each vehicle is constrained to remain inside a convex safe region for the whole duration of each stage in the plan. In other words, during each *stage* all vehicles remain in some convex safe region and the values of all control variables stay constant. The plan is given, therefore, by a set of ordered stages. Recall that events in a tQSP are totally ordered. We call the set of ordered stages between consecutive events in the tQSP a *sequence*.

We call this type of plans that ScottyPath returns *tQSP plans with piecewise constant control*. Additionally, the plans that ScottyPath returns are *optimal*. More formally, *tQSP plans with piecewise constant control* are defined as follows.

Definition 11.14 (*tQSP Plan with Piecewise Constant Control*)**.** A *tQSP plan with piecewise constant control* is a tQSP plan in which the control trajectory and the region assignment are piecewise constant. A tQSP plan with piecewise constant control has the following properties:

- $N - 1$ *sequences*, where N is the number of events in the tQSP. Each sequence, seq_j is the period of time between two consecutive events, e_j and e_{j+1}. Each sequence seq_j consists of N_j *stages*. A stage s_{ji} is the i-th stage of sequence j, with $i = 0, \ldots, N_j - 1$. Each stage has a beginning and an end. The beginning of the first stage in the sequence takes place at the start event of the sequence: $t(s_{j0}.start) = t(e_j)$. The end of the last stage in the sequence, takes place at the end event of the sequence: $t(s_{j,N_j-1}.end) = t(e_{j+1})$. The duration of stage s_{ji}, Δs_{ji} is the difference between its end and start times.

- A control trajectory, $\mathbf{c}(t)$, that is piecewise constant. The control variable vector is constant during each stage in the plan: $\mathbf{c}(t) = \mathbf{c}_{ji}$ for $t \in [t(s_{ji}.start), t(s_{ji}.end))$.

- A region assignment, $\mathbf{r}$, that is piecewise constant. Each vehicle is required to remain in the same region for the duration of each stage in the plan, $\mathbf{r}(t) = \mathbf{r}_{ji}$ for $t \in [t(s_{ji}.start), t(s_{ji}.end))$.

▲

Figure 11-4 shows a diagram that illustrates the relation between events, sequences and stages in a tQSP plan with piecewise constant control.

As in the case of the PDDL-S plans with constant control that ScottyActivity generates (Section 4.2), the fact that ScottyPath generates tQSP plans with piecewise constant control does not affect the completeness or the optimality of ScottyPath.

Theorem 11.1 (Completeness of tQSP Plans with Piecewise Constant Control)**.** *If a tQSP problem has a solution, there always exists a solution that is a tQSP plan with piecewise constant control.*

Theorem 11.2 (Optimality of tQSP Plans with Piecewise Constant Control)**.** *The optimal solution to a tQSP problem, if one exists, is a tQSP plan with piecewise constant control.*

The proofs for Theorems 11.1 and 11.2 are essentially the sames as the proofs for the completeness and optimality of PDDL-S problems (Theorems 4.1 and 4.2). We refer the interested reader to the proofs of the theorems of the PDDL-S plans with piecewise constant control (Appendix A) for additional details.

11.3 Relation Between tQSPs and Other Problems

ScottyPath has been designed as part of the Scotty Planning System. Therefore, it can take the output plan generated by ScottyActivity as its input. However, tQSPs, as described in Section 11.2.1.1, are more expressive than the plans returned by ScottyActivity. In this section, we describe how to express the output plans of ScottyActivity as tQSPs. Additionally, we also describe how to solve the SSPo problems from Chapter 10 with ScottyPath by expressing them as tQSPs.

11.3.1 SSPos and tQSPs

Every *safe-region shortest path problem with ordered goals* (SSPo), as defined in Section 10.1, can be expressed as a tQSP. Recall that a SSPo problem seeks the shortest path contained in safe regions that visits multiple region goals in a given order. Although SSPo problems do not require temporal constraints, dynamics or multiple vehicles, every SSPo can be expressed as a tQSP with:

- One vehicle, v, with as many position state variables as the dimension of the goal regions in the SSPo. The vehicle also has a control variable with bounds $(-1, 1)$ for each of the position state variables.

- An initial state, $\mathbf{x}_{ini}$, that denotes the initial position of the vehicle at $t = 0$. This initial position is the initial position of the SSPo.

- An ordered list of events. The first event is the start event of the tQSP. This events represents the starting position of the SSPo. For each goal region in the SSPo, a corresponding event is added to the list. Each of these events have a state constraint that requires the vehicle to be in the goal region of the SSPo. The last goal region of the SSPo corresponds to the last event in the tQSP.

- A set of episodes. One episode is placed between the start event and the last event of the tQSP. This episode defines, for each position variable of the vehicle, a CLTE effect depending on its associated control variable (e.g. $\dot{x} = v_x$). These effects allow the vehicle to move in the environment. In order to impose the ordering constraints between the goals, additional episodes are placed between each consecutive goal region events with a temporal constraint $(0, \infty)$. Finally, an episode with an objective term minimizing the distance traveled by the vehicle is placed between the start and the end events of the tQSP.

- A region assignment, r_v, that assigns all the safe regions in the SSPo to vehicle v.

Note that the events in a tQSP are fully ordered. Therefore, SSPu problems (in which the goal regions are not ordered), cannot be expressed as tQSP problems.

11.3.2 Scotty Skeleton Plans, PDDL-S Problems and tQSPs

Recall that ScottyActivity plan skeletons, as defined in Section 8.1, represent a partial schedule of totally ordered events, where each event in ScottyActivity is the start or end of an activity. While performing joint activity and trajectory planning, ScottyActivity tests the consistency of plan skeletons by solving a convex model that finds a feasible assignment for the execution times of the events and for the control trajectory.

Since the events in a ScottyActivity plan skeleton are totally ordered, and the continuous effects and state conditions supported by ScottyActivity are similar to that of ScottyPath, every ScottyActivity plan skeleton can be represented as a totally ordered QSP. A tQSP representing a ScottyActivity plan skeleton has the following characteristics:

- For any event in the plan skeleton, an event, e, is created in the tQSP. If the plan skeleton event is a start (end) event of an activity, the *at start* (*at end*) state conditions of the activity are added as the state conditions of event e.
- For every activity in the plan skeleton, an episode is created in the tQSP between the analogous events of their start and end counterparts in the plan skeleton. The temporal constraints, and *over all* state constraints of the activity are added to the episode. Similarly, the continuous effects of the activity are added to the episode.
- Episodes with temporal constraints $[\varepsilon, \infty)$ are added between the events in the tQSP to maintain the same relative order of the events in the plan skeleton and to enforce the epsilon separation of events in plan skeletons.
- An episode between the first and the last events in the tQSP is added. This episode contains all the control variable constraints present in the ScottyActivity PDDL-S problem.
- In order to capture the objective of the PDDL-S problem, another episode between first and the last events in the tQSP is added. For every objective

term in the PDDL-S problem, a similar objective term is added to the newly created episode. This is all that is needed since PDDL-S problems only support objectives at the end of the plan.

Note that the discrete conditions and the discrete effects of activities are ignored and, therefore, not part of the tQSP. This can be done since we require the events to be totally ordered. Because the events are ordered, the semantics of PDDL-s problems ensure that the discrete conditions and effects will always hold.

In order to create a tQSP problem, we further need to define the vehicles of the problem and their relation with the state variables in the PDDL-S problem. This information is not present in PDDL-S problems and, therefore, it needs to be entered externally. Similarly, a mapping from each vehicle to a set of convex safe regions needs to be defined.

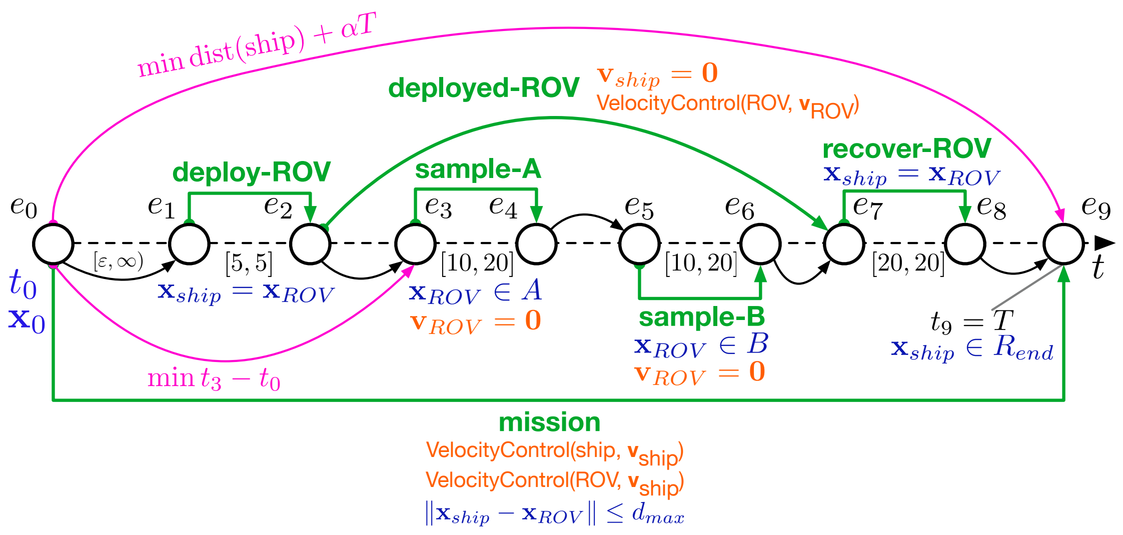
Note that when mapping the solution plan from ScottyActivity to a tQSPpp problem, we only use the information of what activities are selected and the total order of the start and end events in the plan. The actual execution times and control trajectories that ScottyActivity finds are discarded. Instead, ScottyPath finds new execution times and control trajectories that satisfy all the PDDL-S constraints, but also ensure that all the vehicles in the problem always remain within safe regions.

11.4 Planning Approach

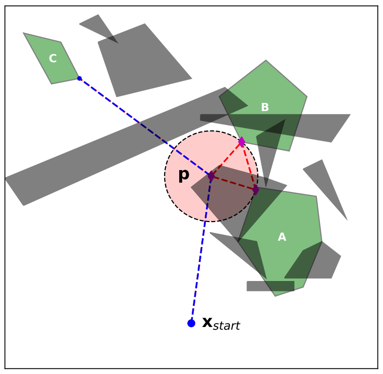
In this section we discuss ScottyPath approach. We first start by providing a high level overview of the algorithm. Afterwards, we provide a detailed description of the search and the convex model used by ScottyPath.

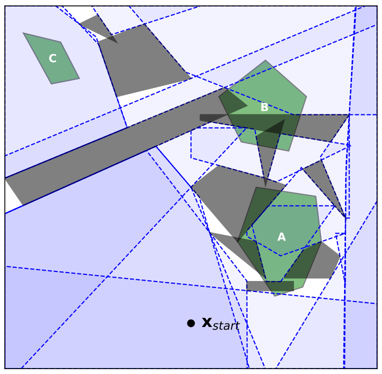
11.4.1 In a Nutshell

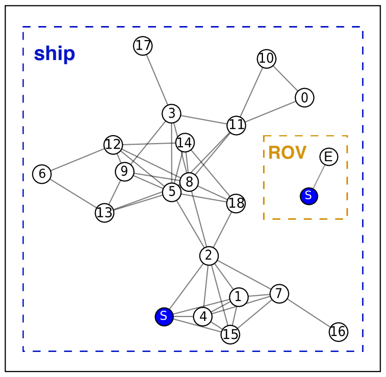
Recall that ScottyPath finds optimal tQSP plans with piecewise constant control. In order to find optimal piecewise constant control trajectories that satisfy all the constraints of the tQSP we use convex optimization. To avoid the non-convex con-



(a)





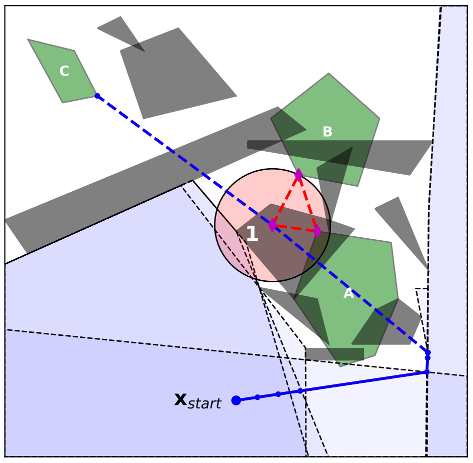


(b) (c) (d)

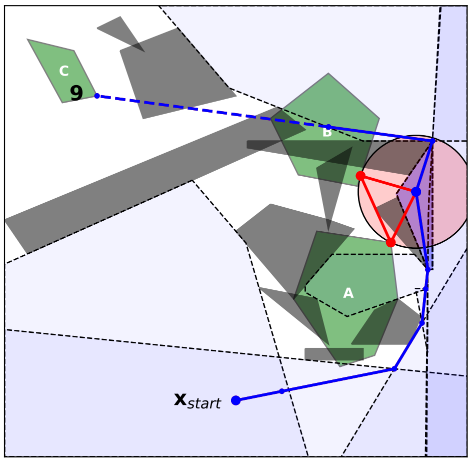
Figure 11-5: Figure (a) shows the tQSP for our example scenario (same as Figure 11-2). Figure (b) shows the optimal trajectories for the ship (blue) and the ROV (red) in the absence of obstacles. The safe regions for the ship are shown in blue in Figure (c). Figure (d) shows the connectivity graph for the ship and the ROV. The starting position is shown in blue and each safe region is shown as a white node. Since all obstacles are on the surface, the ROV only has one safe region, which covers the full environment.

straints that obstacle avoidance requires, we force vehicles to remain within a sequence of overlapping convex safe regions that are guaranteed to be obstacle free. As we did in Chapter 10, we assume that these regions have been generated offline in advance, with one of the methods discussed in Section 10.5.1, based on the IRIS algorithm [30]. Figure 11-5c shows such safe regions for the ship in the example scenario presented in Section 11.1. For each vehicle, we generate a *connectivity graph* that describes how its safe regions are connected. Connectivity graphs are generated with Algorithm 10.1, as described in Section 10.2.3. Figure 11-5d shows the connectivity graphs for the ship and the ROV in the example scenario.

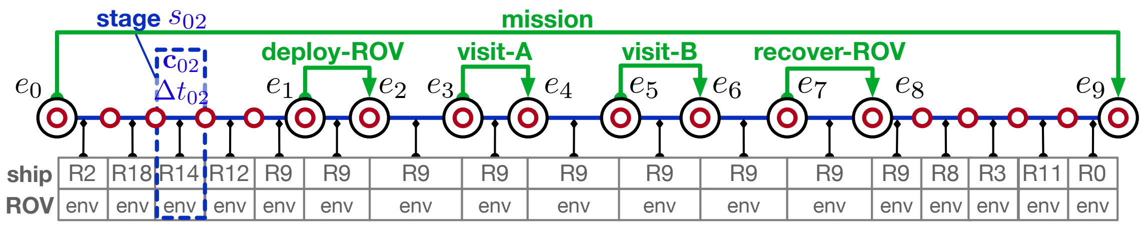
Recall that the tQSP plans with piecewise constant control that ScottyPath finds are comprised of stages, which describe periods of time in which the vector of control variables takes a constant value. Each stage defines: a) a duration, b) a constant value for all control variables, and c) a safe region for each vehicle, such that each vehicle remains inside that safe region for the whole duration of the stage. Recall, as well, that we call a *sequence* the ordered set of stages between consecutive events in the tQSP. In a final solution plan returned by ScottyPath there are $N-1$ sequences, where N is the number of events in the tQSP. Each of these sequences consists of one or more stages. We call a *region assignment*, the composition of each sequence in the plan. That is, the number of stages each sequence contains and the safe regions that each vehicle is assigned in each of those stages. As we describe in Section 11.4.3, when the *region assignment* is known, it is possible to use our convex model to find all the continuous values that minimize the objective of the tQSP for such region assignment. These continuous values are: the schedule of all the events in terms of their execution times, the values of all state variables at each event, and the constant value for the vector of control variables at each stage in the plan. That is, from the region assignment we can use our convex optimization model to find the piecewise constant trajectory of the control variables and the piecewise linear trajectory of the state variables. There are, in general, an infinite number of region assignments that produce feasible solution plans. The problem that ScottyPath solves reduces to finding the optimal *region assignment* such that its associated control and state



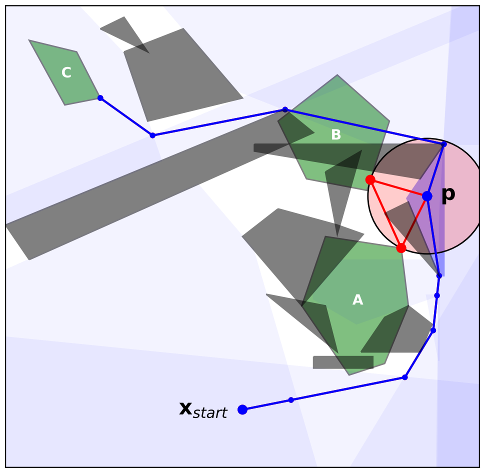
(a)



(b)



(c)



(d)

Figure 11-6: Figure (a) shows the trajectories for the ship (blue) and ROV (red) for a region assignment where the next event that needs to be connected is the deployment event, e_1. The ship safe regions are shown in translucent blue. The trajectories shown minimize $f = g+h$, where g, the committed cost, is the part going through safe regions (in solid lines) and h is the heuristic part ignoring obstacles from that point (in dashed lines). Figure (b) shows the trajectories for a region assignment in which all events have been connected except for the last one, e_9. Figures (c) and (d) show the final optimal region assignment that connects all events and its corresponding trajectories. Since the obstacles do not affect the ROV, it is always inside the region that denotes the full environment, ‘env’.

trajectories minimize the tQSP objective.

ScottyPath decides how many stages are needed for each sequence between consecutive events in the tQSP, and the safe region that each vehicle is assigned for each of those stages. As in Chapter 10, we use A* on the connectivity graphs to find an optimal region assignment. Each search node defines a partial region assignment. The initial search node has an empty region assignment. A child node of a search node adds an additional stage that assigns a safe region to each vehicle. Regions are added until all events in the tQSP are connected. In the case of the geometric planner presented in Chapter 10, the planner adds regions until all goal regions are connected by a path consisting of a sequence of intersecting convex safe regions. ScottyPath operates similarly in that it connects events in the tQSP with sequences of safe regions. The difference, compared to the geometric path planner from Chapter 10, is twofold. First, the region assignment describes a safe region path for each vehicle, as opposed to a single region path. Second, we do not have an explicit representation of the goal regions that need to be visited. In this case, the events in the tQSP can be considered goals that need to be connected by sequences of safe regions. Each sequence consists of an ordered set of stages between consecutive events in the tQSP. Each event has associated a time and values for all state variables at that point in the plan, including the positions of all vehicles. ScottyPath adds stages to the end of a sequence until it becomes possible to connect to the next event. A connection is possible if the time and the values of all state variables at the end point in the last stage match the time and the state variables of the next event. This means that each vehicle has a path through safe regions that leads to the time and the state variable values at the event. The time and the state variables of the event need to satisfy all the constraints in the tQSP. Whenever a connection is made, the sequence connecting that event is closed. Children nodes of the search node where the connection was made have partial region assignments with a new sequence. This new sequence aims to connect the next event in the tQSP. Since we explore region assignments in best-first order, the problem is solved when the last event in the tQSP is connected.

In our example scenario, the first event that needs to be connected by regions is the start event of the ROV deployment episode (labeled as e_1 in Figure 11-5a). The tQSP does not specify the exact point where the ROV should be deployed. However, that point is constrained by the requirement, specified in the tQSP, that the ROV visits sampling regions A and B without repositioning the ship and without violating the tether range constraint. Figure 11-6a shows a node with a partial assignment of safe regions in which the next event that needs to be connected is e_1. The solid lines show the trajectories of the ship and ROV going through the regions in the stages of the region assignment. The dashed lines show the rest of the plan that is not connected by safe regions yet. The trajectories, connected by safe regions or not, satisfy all the constraints in the tQSP. However, the part of the plan connected by safe regions is obstacle free, while the unconnected part ignores obstacles. In Figure 11-6b, event e_1 and all others except for e_9 have already been connected. Finally, Figure 11-6c shows the final optimal region assignment in which all the events are connected by safe regions. Figure 11-6d shows its resulting optimal trajectories.

In order to use A*, we need to compute the cost (g) and heuristic (h) for each search node. In the spirit of our method in Chapter 10, we compute g and h jointly by solving a convex optimization model for each node in the search. Recall that the optimal plan that we are seeking is the one that minimizes the linear combination of all objective terms in the episodes in the tQSP. Each objective term c_t is split into its cost part g_c and its heuristic part h_c. The cost part of the objective is the accumulated value incurred from the start until the end of the last stage going through safe regions. The heuristic part is the accumulated value from that point until the end of the plan, that is, the remaining unconnected section of the plan. In the example scenario, one of the objective terms minimizes the distance traveled by the ship. The g part of that objective, for a given region assignment, is the distance traveled by the ship through the assigned safe regions. The h part is the length of the path connecting the remaining events that ignores the obstacles. As an additional example, imagine another objective minimizing the distance traveled by the ROV. Consider a search node with a region assignment like the one shown in Figure 11-6a where the deployment point, event e_1, has not been connected yet. For this node, the heuristic part of the objective term is the full distance traveled by the ROV (red dotted trajectory) and the cost part is 0, since the ROV does not cover any distance in the part of the plan connected by safe regions. On the other hand, for the search node with the region assignment shown in Figure 11-6b, the cost part of the objective term is the distance traveled by the ROV (solid red line). The heuristic part is 0, since the ROV does not need to move again in the remaining part of the plan not connected by safe regions. This is the case because the ROV is being carried by the ship during that part of the plan.

As discussed earlier, search nodes are expanded by creating child nodes with one additional stage appended at the end. This stage assigns a safe region to each vehicle that is a neighboring region from the region assigned to each vehicle in the last stage. Additionally we need to test, for every search node, whether the its region assignment can connect the next event in the tQSP. We determine whether this connection is possible by solving the optimization model one more time with the objective of minimizing the distance from the last point going through safe regions to the event that needs to be connected. If such distance is 0, the connection is possible. In effect, when the distance is 0, the time and values of the state variables for the event are the same as the time and the values of the state variables for the last point going through regions. If this is the case, we create a child node in which the connection is enforced by increasing the index of the next event that needs to be connected.

The search finalizes once all the events in the plan are connected through safe regions. Since our heuristic ignores obstacles in the part of the tQSP not connected by safe regions, and our optimization problem is convex, the heuristic value is a lower bound on the remaining cost. That is, the cost for the remaining part of the tQSP not yet connected can only be as low as the heuristic value. Hence, our heuristic is admissible. Therefore, our A* search returns an optimal assignment of regions connecting consecutive events that minimizes the linear combination of objectives in the tQSP.

In the rest of this section we describe the search algorithm in more detail, as well as the convex optimization model that we use.

11.4.2 Search Algorithm

```
Algorithm 11.1: ScottyPath-Plan
Input: A tQSP problem, $P$, and a heuristic weight $\alpha \geq 0$.
Output: A solution tQSP plan with piecewise constant control if one exists or
        $nil$ otherwise. The solution is optimal if $\alpha \leq 1$.
Algorithm
1  $Q \leftarrow$ CREATE-PRIORITY-Q()
2  for $v_i$ in $P$.vehicles do
3      $R_{v_i} \leftarrow$ GET-CONTAINING-SAFE-REGIONS($P.\mathbf{x}_{ini}(v_i)$)
4  for $\langle r_{v_1}, \ldots, r_{v_{n_v}} \rangle$ in CROSS-PRODUCT($R_{v_1}, \ldots, R_{v_{n_v}}$) do
5      $N_{new} \leftarrow$ MAKE-NODE($\langle r_{v_1}, \ldots, r_{v_{n_v}} \rangle$)
6      $\langle \text{feasible}, g, h \rangle \leftarrow$ COMPUTE-G-H-DIST($N_{new}$)
7      if $feasible$
8          PUSH($Q$, $g + \alpha \cdot h$, $N_{new}$)
9  while not IS-EMPTY($Q$) do
10     $N \leftarrow$ POP($Q$)
11     if NEEDS-EXPANSION?($N$)
12         MARK-EXPANDED($N$)
13         if ALL-EVENTS-CONNECTED?($N$)
14             return MAKE-PLAN($N$)
15         for $N_{child}$ in MAKE-EXTENSIONS($N$) do
16             $\langle \text{feasible}, g, h \rangle \leftarrow$ COMPUTE-G-H-DIST($N_{child}$)
17             if $feasible$ and NEEDS-EXPANSION?($r$)
18                 PUSH($Q$, $g + \alpha \cdot h$, $N_{child}$)
19 return $nil$
```

The search algorithm that assigns sequences of safe regions to each vehicle in tQSP is shown in Algorithm 11.1. As described earlier, we use A* to find an optimal assignment of regions that minimizes the objective of the tQSP problem. The search expands nodes in order of increasing f $(= g + h)$ values. For every search node, we use a convex optimization model to jointly compute the cost (g) and heuristic (h) values of the region sequences assigned by that node (Lines 6 and 16). We also use our convex model to find the minimum possible distance from the last point in the part of the tQSP already connected by safe regions to the next event that has not been connected yet.

```
Algorithm 11.2: MAKE-EXTENSIONS
Input: A search node, $N$.
Output: A set $C$ of the possible node expansions from $N$.
Algorithm
1  $C \leftarrow \{\}$
2  if $N.next\_distance == 0$ /* Connection to next event is possible.  */
3      $N_{new} \leftarrow N$
4      $N_{new}$.next_event = $N$.next_event + 1
5      ADD-STAGE($N_{new}$, $N$.last_stage_regions)
6      ADD($C$, $N_{new}$)
7  for $v_i$ in $P$.vehicles do
8      $R_{v_i} \leftarrow$ GET-NEIGHBORING-SAFE-REGIONS($v_i$, $N$.last_stage_regions)
9  for $\langle r_{v_1}, \ldots, r_{v_{n_v}} \rangle$ in CROSS-PRODUCT($R_{v_1}, \ldots, R_{v_{n_v}}$) do
10     $N_{new} \leftarrow$ CLONE-AND-ADD-STAGE($N$, $\langle r_{v_1}, \ldots, r_{v_{n_v}} \rangle$)
       ADD($C$, $N_{new}$)
11 return $C$
```

The first step in the algorithm consists in finding, for each vehicle, the safe regions that contain its initial location. We compute the cross product of the possible initial regions for each vehicle in order to create a set of starting search nodes (Lines 2 to 8). Each is placed on the queue as root nodes. The search finalizes when all the events are connected by safe regions (Line 13). At that point, ScottyPath returns a tQSP plan with piecewise constant control with the region assignment, the schedule of the events and the control and state trajectories for each vehicle (Line 14). Since our heuristic is admissible, the region assignment found by the search produces an optimal plan.

Search nodes are expanded by creating child nodes with an additional stage appended at the end (Algorithm 11.2). The newly added stage assigns a safe region to each vehicle. To compute the possible child nodes of a given search node, we first find all the neighboring safe regions of each vehicle's last region. We then compute the cross product of the neighboring regions for each vehicle in order to find all the possible region assignment extensions for the current node. A child node is created for each of these (Lines 9 to 10). The neighboring safe regions of a safe region include the same safe region itself. We do this to allow some vehicles to stay in the same region they were while other vehicles change to a different safe region.

Finally, we test, for each node, whether it is possible to reach the next unconnected event through its region assignment. If that is the case, we create an additional child node in which the index of the next event that needs to be connected is increased by 1. The new sequence that starts at the event that has just been connected contains one stage. This stage assigns, to each vehicle, the same region they were assigned in the previous stage. That is, each vehicle stays in the same region that made the connection possible (Lines 2 to 6). Note that we do not force nodes to connect events when the connection is possible, but that we only enforce this connection in a child node. The reason is that making the connection as soon as it becomes feasible is not always the best option to find an optimal plan. In order to find optimal plans, we need to explore the full search space, and making the connection is a choice that is represented by a child node.

In order to avoid exploring unnecessary region assignments, we use an expanded list. In the expanded list, we mark the regions assigned to each vehicle in the last stage of the region assignment (Line 12). For example, in a problem with two vehicles, v_1 and v_2, the last stage of a region assignment may assign $\langle v_1 \leftarrow R_7, v_2 \leftarrow R_3 \rangle$, where R_7 and R_3 are the regions for the first and second vehicle. The regions assigned to each vehicle are considered jointly. Therefore, another search node with a region assignment whose last stage was $\langle v_1 \leftarrow R_7, v_2 \leftarrow R_{12} \rangle$ would not be considered expanded, and the algorithm would expand it. Note, however, that the same safe regions may have to be visited by the vehicles at the same time at later steps in the tQSP. For example, consider a tQSP problem in which the two vehicles have to return to a certain location after they visit each of their goals. In this case, the assignment $\langle v_1 \leftarrow R_7, v_2 \leftarrow R_3 \rangle$ may be needed multiple times throughout the plan. To avoid this issue, the expanded list records, not only the region that each vehicle is assigned, but also the next event in the tQSP that has to be connected next at that point in the search. For example, the expanded list would record: $\langle e_1, v_1 \leftarrow R_7, v_2 \leftarrow R_3 \rangle$, meaning that v_1 is in region R_7 and v_2 is in region R_3 when the next event that has to be connected by safe regions is e_1. The search would still expand the assignment $\langle v_1 \leftarrow R_7, v_2 \leftarrow R_3 \rangle$, but only if it was associated with a search node in which the next event that had to be connected would be different from e_1, e.g. later in the tQSP. This is analogous to the expanded list used in the geometric path planner in Chapter 10.

11.4.3 Computing Optimal Trajectories Through Convex Safe Regions using Convex Optimization

Our convex optimization model is used for three purposes. First we use it to compute the cost and heuristic value for each search node. Second, we use the same constraints with a different objective to compute the minimum distance from the last point going through the safe regions in a region assignment to the next event in the tQSP not yet connected. This distance is used to check whether the current region assignment can connect the next event. Finally, this model is also used to compute the optimal tQSP plan with piecewise constant control when the complete region assignment connecting all the events is found. In this case the model and objective are exactly the same as when computing the g and h values for a search node. The difference is that, because all events are connected, the heuristic value, h, is 0, since the full plan goes through safe regions.

Since the state constraints, the continuous effects and the control variable constraints are the same as in ScottyActivity, the convex optimization model that we use for ScottyPath is almost the same as the one described in Chapter 6, except for a few differences that we describe in this section. Recall that in the ScottyActivity model, *stages* are the periods of time between consecutive events in which the control variables vector, $\mathbf{c}$, takes a constant value. Since the control variables are constant during stages, the trajectory of the state variables vector, $\mathbf{x}$, during each stage is described by a straight line segment, and the change is linear in time. In the ScottyPath model, consecutive events are not separated by just one stage but, instead, by a sequence of stages. Since the vector of control variables, $\mathbf{c}$, takes a constant value during each *stage*, each *sequence* describes a piecewise linear trajectory, where each segment corresponds to each stage. Since the semantics of tQSP problems only

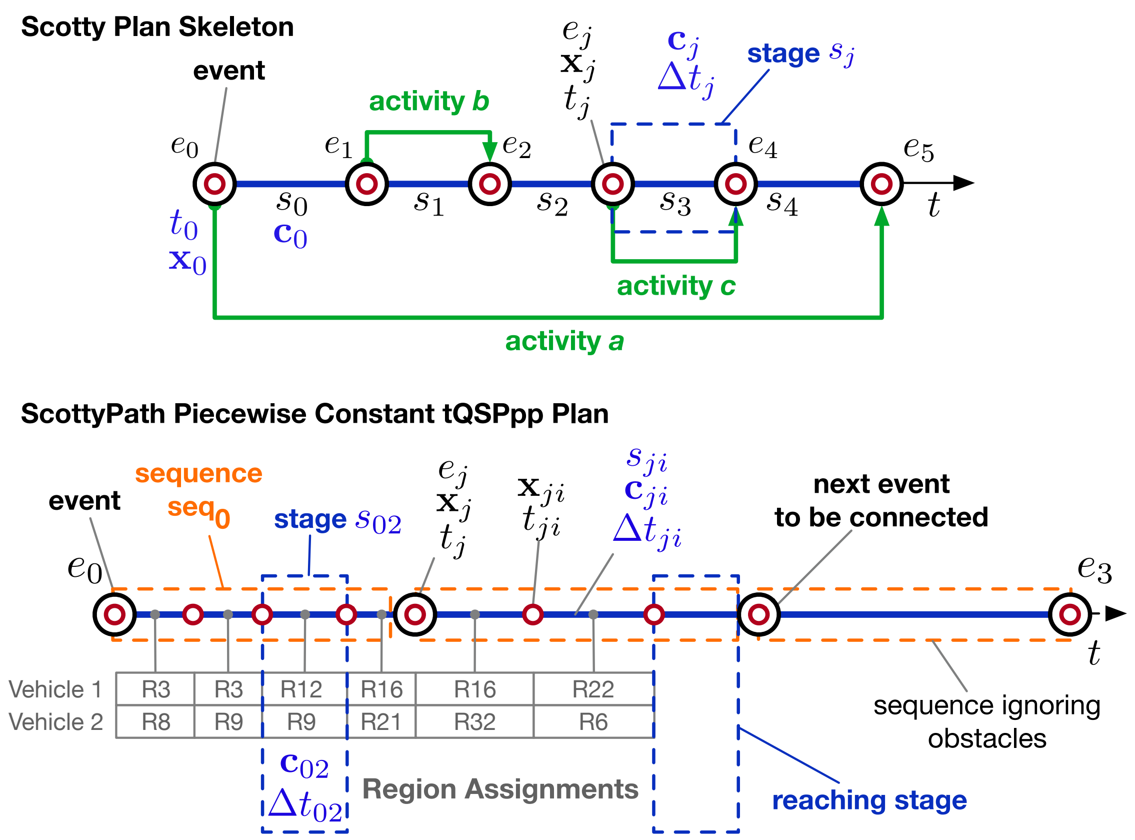


Figure 11-7: Events, sequences and stages in a ScottyPath plan and their relation to ScottyActivity plan skeletons.

allow state constraints, control variable constraints or continuous effect to become enabled or disabled at events, the same constraints apply to all the stages in the same sequence.

Figure 11-7 describes the relation between events, sequence and stages and the differences with the ScottyActivity plan skeletons. We can distinguish three types of sequences: 1) sequences leading to connected events, 2) the sequence leading to the event that has to be connected next, and 3) sequences between unconnected events.

Every stage in sequences leading to connected events, such as seq_0 in Figure 11-7, assign a safe region to each vehicle. Vehicles are required to remain in this region for the duration of the stage. Since each stage corresponds to a straight line segment and safe regions are convex, it suffices to constrain the beginning and the end of the stage to be inside the region in order to guarantee that the full segment is contained in the region, as explained in Chapter 5.

Next, consider the sequence leading to the event that has to be connected next, such as seq_1 in Figure 11-7. This sequence has one or more stages with assigned safe regions, and one stage at the end with no safe regions that ends at the event that needs to be connected next. We call this last stage the *reaching stage*. The start of the reaching stage marks the point from which no safe regions are assigned and, therefore, obstacles are ignored. If the reaching stage has length and duration 0, that is, its presence does not make a difference, then the event would be connected by a path safe regions for each vehicle.

All sequences after the sequence containing the reaching stage link consecutive pairs of events in which neither has been connected by a path of safe regions yet. This is the case for seq_2 in Figure 11-7. These sequences only have one stage, and this stage does not assign safe regions. The events in these sequences are connected by only one straight line segment that ignores obstacles and that uses the same constraints as in the ScottyActivity model.

The other difference in the ScottyPath model compared to ScottyActivity's that remains to explain is the computation of the cost (g) part and the heuristic (h) part of each objective term. Since the objective terms are the same as in ScottyActivity, the formulation for the objectives is the same as the one described in Chapter 6. However, instead of computing each objective term throughout the entire plan, as in ScottyActivity, we divide each objective term into the part that goes through the section of the tQSP connected by safe regions and the part that ignores obstacles after that. We do this because we need to compute the committed cost incurred in the region assignment of the node (g), and the best case remaining cost-to-go in the part of the tQSP not yet connected by safe regions, that is, the heuristic value (h). The part of each objective, obj, that goes through the safe region contributes to the committed cost term (g) whereas the part that ignores obstacles contributes to the heuristic term (h). Figure 11-8 shows, with an example, how this division between committed cost and remaining cost-to-go is done.

In order to compute both g and h we solve the model to minimize $f = g + h$. Note that minimizing only f is not sufficient to guarantee that the returned value for

```
Algorithm 11.3: COMPUTE-G-H-DIST
   Input: A search node, $N$.
   Output: Whether the node's region assignment is feasible, and the values of
           the cost, $g$, and heuristic, $h$, of $N$, as well as the connection distance,
           $d$, to the next unconnected event.
   Algorithm
 1 $\langle g, h\rangle \leftarrow \langle nil, nil\rangle$
 2 prog ← BUILD-PROGRAM($P$, $N$.region_sequences)
 3 feasible ← OPTIMIZE(prog, $\langle g + h, h\rangle$)
 4 if feasible
 5     $\langle g, h\rangle \leftarrow \langle$GET-VALUE($g$), GET-VALUE($h$)$\rangle$
 6     OPTIMIZE(prog, $\langle distance\rangle$)
 7     $N$.distance_to_next_event ← GET-VALUE($distance$)
 8 return ⟨feasible, $g$, $h$⟩
```

h is as small as possible. The solver has no incentive to ensure that the heuristic part of the optimization is the one that strictly does not go through safe regions. As a consequence, the solver could choose values for h larger than the real remaining cost, rendering the heuristic not admissible. In our case, this does not affect the optimality of the standard A* algorithm, since the order in which nodes are visited only depends on f, and our convex model guarantees an optimal value for f. This issue is important, however, when using weighted A*. Without an admissible heuristic, it is otherwise not possible to guarantee the optimality bounds that weighted A* provides. Therefore, in order to address this problem, we solve a slightly different problem that minimizes h for the smallest possible value of $f = g + h$. This is represented with the following optimization problem:

$$\begin{aligned} \underset{t_{ij}, \mathbf{x}_{ij}, \mathbf{c}_{ij}}{\text{minimize}} \quad & h \\ \text{subject to} \quad & \inf\left(g + h\right), \\ & g = \sum_{obj} g_{obj}, \\ & h = \sum_{obj} h_{obj}, \\ & \text{tQSP constraints} \end{aligned} \tag{11.5}$$

, where t_{ij} and $\mathbf{x}_{ij}$ are the times and values of the state variables at the start and end of each stage and $\mathbf{c}_{ij}$ are the constant values of the control variables vector during each stage. In practice, this is achieved by solving the model twice. The first time minimizing f and the second time minimizing h such that the minimum value of f is maintained. Most commercial solvers support multi-objective optimization. In this case, it suffices to set f as the primary and h as the secondary objective (Line 3 of Algorithm 11.3).

Figure 11-8 demonstrates how the computation for g and h is performed for a search node, for a scenario similar to the ship-ROV example scenario. In this example, the event that needs to be connected next is e_1. The objective for this problem is the distance traveled by the ship. As shown in the diagram, the cost part of the objective is the distance traveled by the ship while traversing through the safe regions (shown in a solid blue line). The heuristic part of the objective is the remaining distance traveled, while ignoring obstacles (shown in a dashed blue line).

If the model has no solution, the region assignment associated with the search node is not feasible and the node is discarded. If the model has a solution, we solve it again to determine whether the current region assignment can connect the next unconnected event in the plan (Line 6 of Algorithm 11.3). In this case, the objective for the model is the duration of the *reaching stage*. If the resulting duration of the reaching stage is 0, no change in the state variables happens during the reaching stage, and the end of the previous stage matches the next event in time and value of the state variables. Therefore, the current region assignment can connect to the next event if desired, and the search creates such child extension when the node is expanded. Figure 11-9 shows two examples of two different search nodes and the trajectories returned by the solver when minimizing f and when minimizing the duration of the reaching stage. In the first case, the connection is not possible. In the second case, the region assignment allows event e_1, the ROV deployment event, to be reached through the region assignment associated with the search node.

Finally, the same model is used to find the final tQSP plan with piecewise constant control once all events are connected by safe regions. The objective in this case is

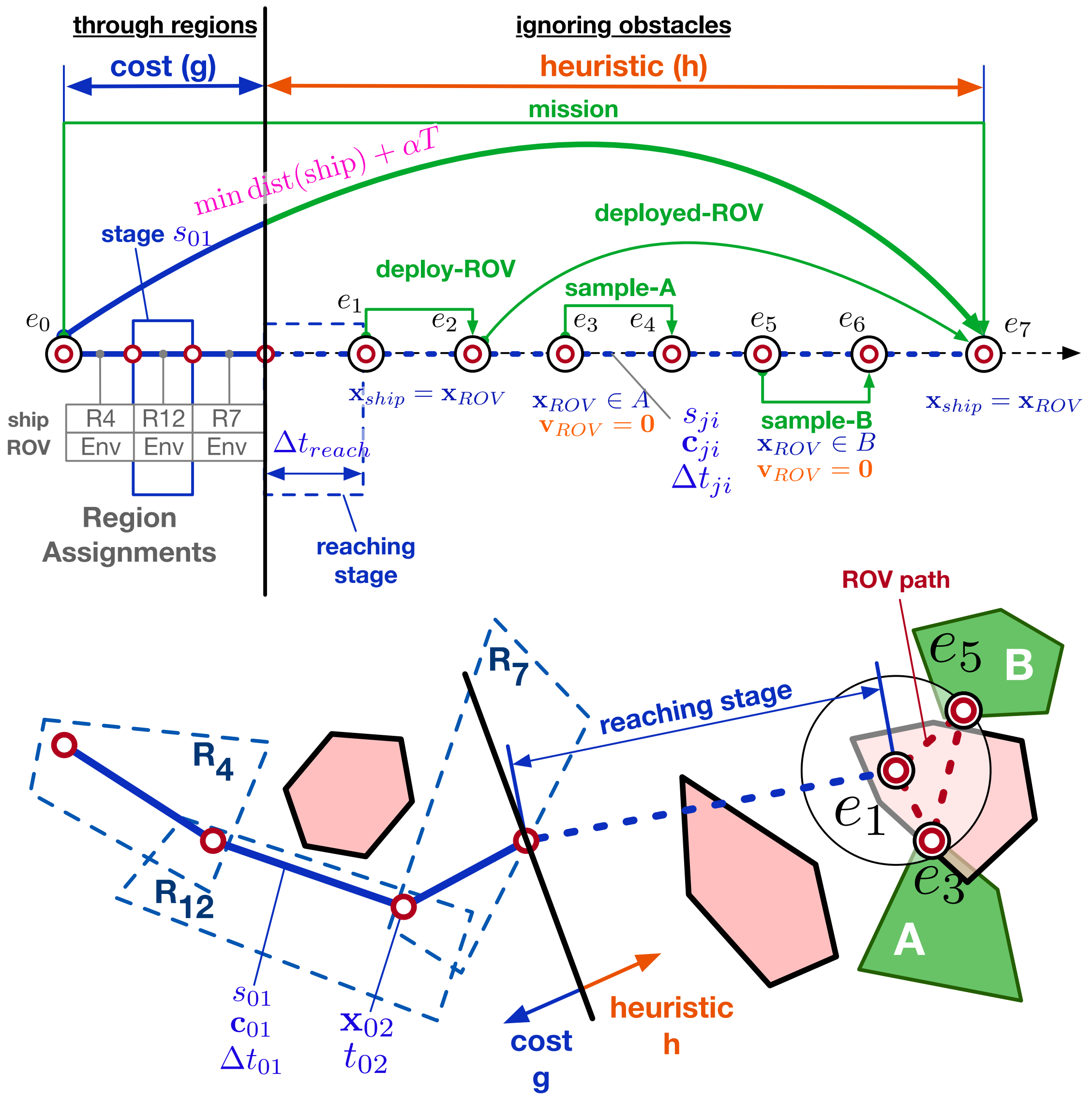
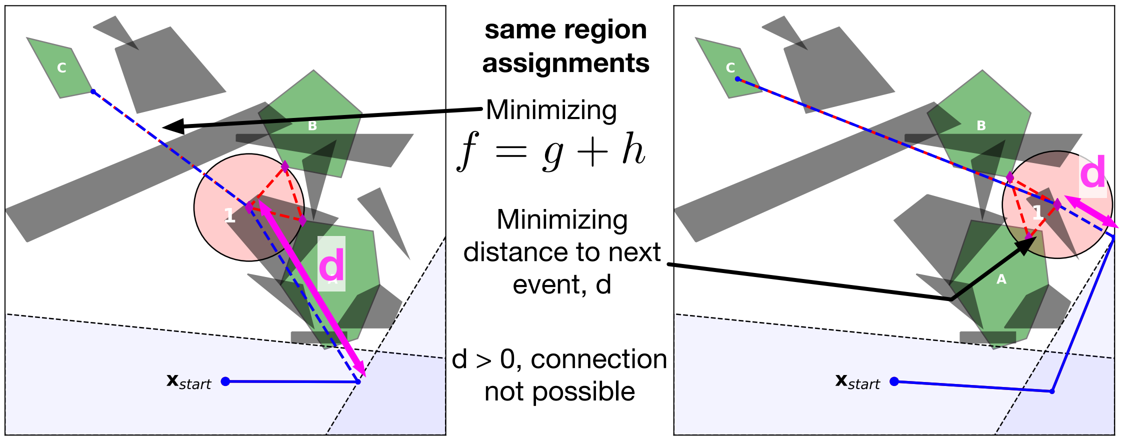
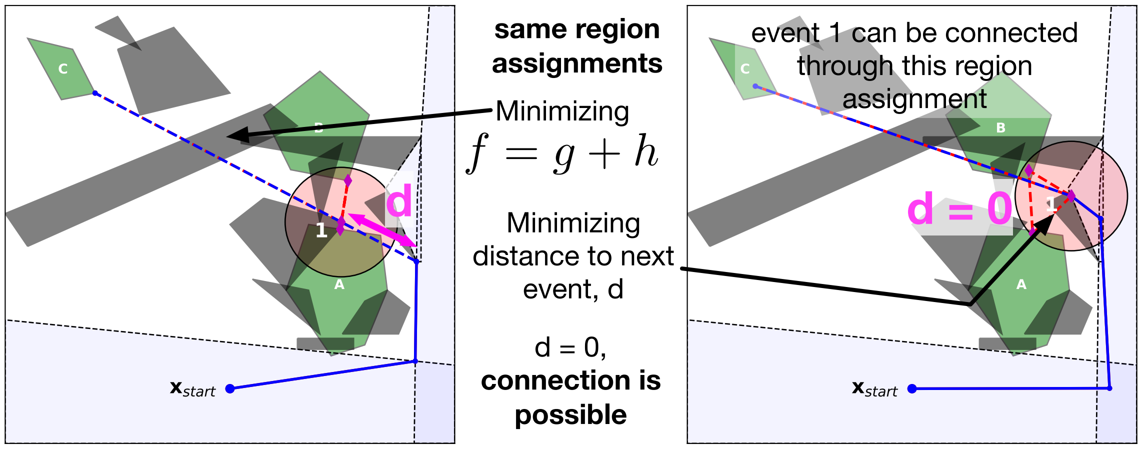


Figure 11-8: Example showing how g and h are computed for a search node in which the ROV deployment event, e_1, is being connected. The part of each objective term that takes place through the safe regions adds the committed cost (g) of the search node, while the part that takes place after the last region contributes to the heuristic value (h). A search node can connect the next event if it is possible to assign a duration of 0 to the *reaching stage*.


C
B
1
d
$\mathbf{x}_{start}$
same region assignments
Minimizing
$f = g + h$
Minimizing distance to next event, d
d > 0, connection not possible
C
B
d
1
A
$\mathbf{x}_{start}$

(a)


C
d
1
A
$\mathbf{x}_{start}$
same region assignments
Minimizing
$f = g + h$
Minimizing distance to next event, d
d = 0,
connection is possible
event 1 can be connected through this region assignment
C
B
d = 0
1
A
$\mathbf{x}_{start}$

(b)

Figure 11-9: Figure (a) and (b) show two different region assignments of nodes trying to connect event 1 through safe regions. Left of figure (a) shows the trajectories resulting from optimizing for $f = g + h$, while the right hand side figure shows the trajectories minimizing the distance between the last point going through safe regions and the point that satisfies event 1 constraints. Since the distance between those points is not zero, the region assignment in (a) cannot connect event 1. On the other hand, Figure (b) shows a region assignment that is able to connect event 1 (right figure).

the same as when determining the values of g and h. However, since all events are connected, h is 0, and the final cost of the plan is $f = g$.

11.5 Chapter Summary

In this chapter, we presented ScottyPath, the second and last main component of the Scotty Planning System. ScottyPath takes an input qualitative state plan that is generated either by hand or with ScottyActivity, and returns an obstacle-free plan that consists of a schedule and state and control trajectories for each vehicle. The state trajectories of each vehicle are obstacle free because they are guaranteed to be contained inside convex safe regions, which are computed in advance. The key insights in ScottyPath are the following. First, we use informed search to assign a sequence of safe regions to each vehicle. Second, we use the Scotty convex model to compute, jointly, the cost and heuristic of candidate assignments. Third, we rely on our convex optimization model to determine whether a candidate region assignment can satisfy the next goal constraints in the plan, even when those goal constraints do not explicitly define the state-space region that needs to be reached.

Chapter 12

ScottyPath Experimental Results

In this chapter, we evaluate the scalability of ScottyPath in terms of the size of the input tQSP, the number of vehicles and the number of safe regions. We use the same robotic domains from Chapter 9 to demonstrate that our approach scales well with the size of the tQSP. The performance of our approach degrades with the number of vehicles and safe regions. However, we demonstrate that by trading some solution quality, we can significantly improve the runtime of our planner.

12.1 Description of the Experiments

Recall that ScottyPath is designed to take tQSP problems generated from the output plans found by ScottyActivity, and return obstacle-free plans in which vehicles go through safe regions. We use the same domains that we used to evaluate the performance of ScottyActivity and that are explained in Section 9.2. These are the AUV, the ROV and the Air Refueling domains. For those domains, ScottyActivity solves the activity and trajectory planning problem. That is, from a set of initial conditions and a set of goal conditions at the end of the plan, ScottyActivity returns a set of ordered starts and ends of activities and a control and a state trajectory that achieves the problem goals. ScottyActivity solves this problem in the absence of obstacles. ScottyPath does not focus on activity planning, since it is already performed by ScottyActivity. Instead, the test problems are tQSP problems that are generated from the solutions found by ScottyActivity for each of the problems in Section 9.2. In particular, we use the solution found by ScottyActivity using the obj-EHC search algorithm presented in Section 8.4, since it often generates better quality plans, as shown in Section 9.2.4. The original problems in Section 9.2 do not have obstacles. To evaluate the performance of ScottyPath, we augment the tQSPs generated from the ScottyActivity solutions with random obstacles and safe regions for each vehicle.

When testing the scalability of ScottyPath, we focus on its performance with respect to several characteristics. First, we use the three same benchmarking domains from Section 9.2: the AUV, the ship-ROV and the Air Refueling domains. These domains are different in terms of the complexity of the constraints involved, the number of vehicles and the length of the plans.

Second, each of these domains has 20 instances. The first instances of each domain have associated short tQSPs, with few events and episodes. Later instances have significant longer tQSPs in terms of the number of events and episodes. In the case of the Air Refueling domain, problem instances 11-20 have one additional UAV. The size of the input tQSP directly impacts the depth of the ScottyPath search.

Third, we use, for each domain, three environments with varying number of obstacles and safe regions. These environments are generated randomly. The number of safe regions in a tQSP problem directly impacts the branching factor (breadth of the search), as well as the depth. That is, for an environment with many safe regions and many obstacles, it may be necessary to visit many of those safe regions in order to reach a goal region. Similarly, a large number of safe regions implies that the number of descendants of each search node is large.

Fourth, we use weighted A* to solve the benchmark problems. We solve each problem for three different values of the heuristic weight, in order to evaluate the trade off between the optimality of the solution found and the runtime of the algorithm.

These four factors result in 540 problems of varying difficulty that we use to evaluate the performance of ScottyPath. Throughout the rest of this section, we describe these factors that affect the difficulty of the benchmark problems.

12.1.1 Benchmark Domains

The AUV, the ROV and the Air Refueling domains represent robotic scenarios. These domains are described in detail in Section 9.2.

Recall that the AUV domain represents a simple problem in which an autonomous underwater vehicle (AUV) needs to visit one or more regions. There are up to 14 target regions in this domain, depending on the problem instance.

The ROV domain is similar to the example scenario that we use throughout Chapter 11, and that is described in Section 11.1. There are two vehicles in this problem, a ship and a remotely operated vehicle (ROV). In this domain, the ROV needs to take samples in up to 20 regions, depending on the problem instance. The solution tQSPs found by ScottyActivity specify when the ROV is deployed and recovered and the order in which the regions are sampled. As in the example scenario in Chapter 11, the obstacles that we consider are in the surface and only affect the ship, and not the ROV.

Finally, the Air Refueling domain describes a scenario in which an unmanned aerial vehicle (UAV) needs to take images at up to 10 regions, depending on the problem instance. The UAV has limited battery, and its decrease rate depends on the velocity of the UAV. To complete its mission, the UAV can refuel, while flying, from a tanker plane. While refueling, the UAV and the tanker plane need to stay within a close distance from each other. Instances 11-20 have an additional UAV. The solution tQSPs found by ScottyActivity specify when the UAVs refuel and the order in which the regions are imaged. In this domain, the obstacles need to be considered by the tanker plane and, also, the UAVs.

The three domains are different in terms of the number of vehicles and the complexity of the constraints used. For example, the AUV domain only has one vehicle. On the other hand, some instances of the Air Refueling have three vehicles, and all of them need to avoid obstacles. Moreover, the refueling constraint requires vehicles to stay close to each other. Moreover, the battery depletion effect requires complicated cone constraints. More complicated problems require more state and control variables and more complicated constraints. This affects the time that it takes to solve each of the optimization problems in the search.

12.1.2 Domain Instances

Each of the three domains has 20 problem instances. In general, each successive problem is harder than the previous one. The difficulty is increased by forcing the vehicle to visit an additional region. This translates in the tQSPs having more events and episodes. In the AUV domain, an additional target region simply requires one more episode in the tQSPs. This is not the case in the ROV and Air Refueling domains. For example, an additional target region may require the ship to recover the ROV, navigate towards the new sampling region and deploy it there one more time.

Since the ScottyPath algorithm finds sequences of safe region paths that connect, one by one, the events in the tQSP, the size of the tQSP directly affects the depth of the search.

12.1.3 Number of Safe Regions and Obstacles

Recall that ScottyPath finds obstacle free plans by ensuring that each vehicle is always within a safe region. Therefore, the number of available safe regions for each vehicle has a large impact in the runtime of the ScottyPath algorithm. In order to evaluate the scalability of ScottyPath in terms of the number of safe regions, we generate three random environments for each of the three domains. We call these environments the ‘easy’, ‘medium’ and ‘hard’ environments. These environments are shown in Figure 12-1. The ‘easy’ environment of each domain has around 10 regions, the ‘medium’ has 30 and the ‘hard’ environment has roughly 100 safe regions. In the case of the AUV and the ROV domains, only one vehicle has to go through safe regions. The Air Refueling domain is harder, since both the tanker and the UAV need to go through the safe regions. Furthermore, there is an additional UAV in instances 11-20 that also needs to go through the safe regions.

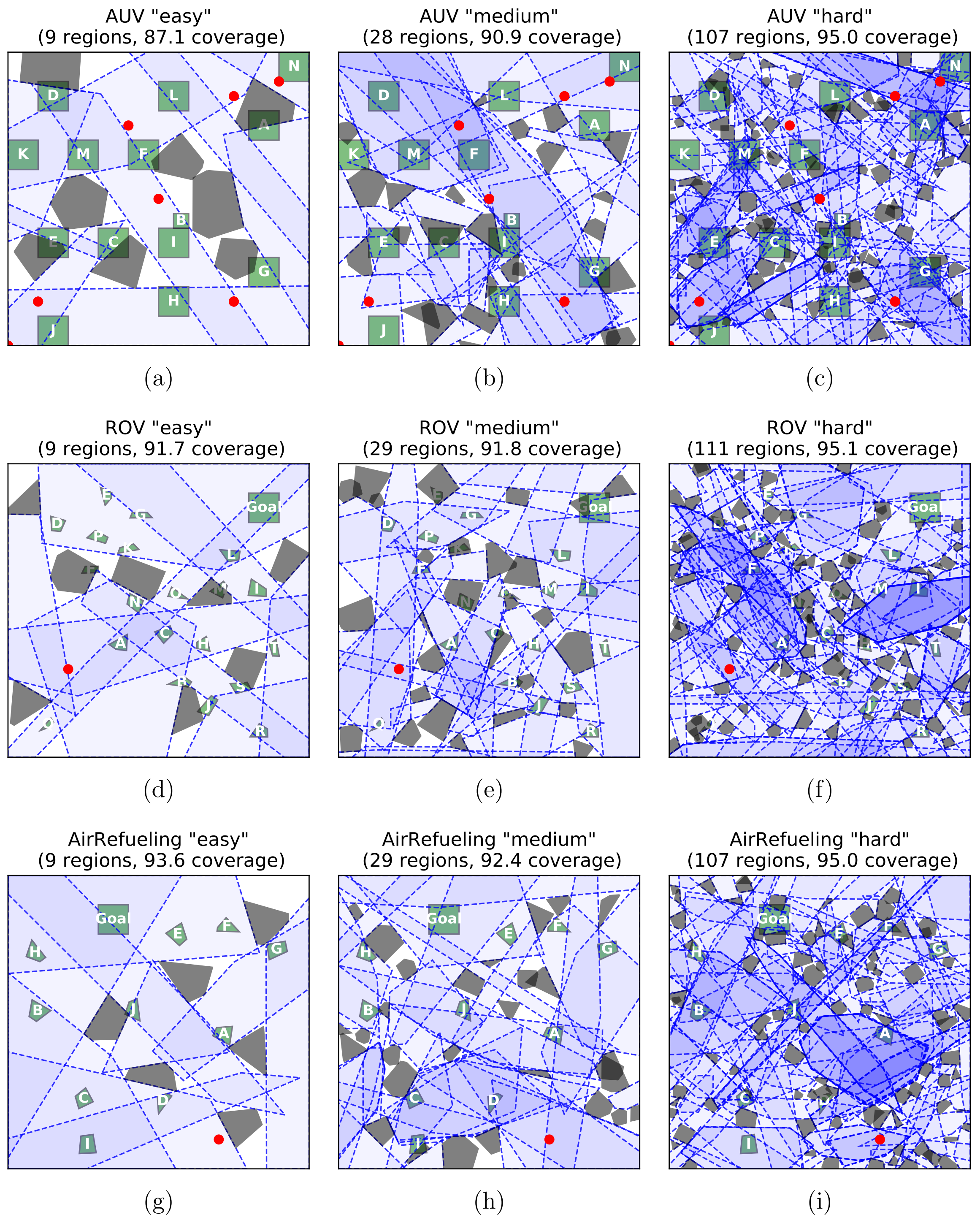

AUV "easy"
(9 regions, 87.1 coverage)
AUV "medium"
(28 regions, 90.9 coverage)
AUV "hard"
(107 regions, 95.0 coverage)
(a)
(b)
(c)
ROV "easy"
(9 regions, 91.7 coverage)
ROV "medium"
(29 regions, 91.8 coverage)
ROV "hard"
(111 regions, 95.1 coverage)
(d)
(e)
(f)
AirRefueling "easy"
(9 regions, 93.6 coverage)
AirRefueling "medium"
(29 regions, 92.4 coverage)
AirRefueling "hard"
(107 regions, 95.0 coverage)
(g)
(h)
(i)
Goal
N
D
L
A
K
M
F
B
E
C
I
G
H
J

Figure 12-1: This figure shows the 'easy', 'medium' and 'hard' environments for the AUV, ROV and Air Refueling domains. Target regions are shown in green, obstacles in gray and safe regions in blue. The red dots show the starting points of vehicles across all problem instances.

12.1.4 Heuristic Weight

The ScottyPath algorithm, as presented in Section 11.4 is optimal. It returns the plan that minimizes the problem objective while ensuring that all the vehicles always remain within safe regions. However, it is often desirable to trade some solution quality in order to improve the runtime of the algorithm. In order to do so, we use weighted A*. In the experiments reported in this chapter, we use three settings for the heuristic weight: 1.025, 1.05 and 2. In our results, we report the increase in the solution objective and the decrease in the planning runtime as the heuristic weight increases.

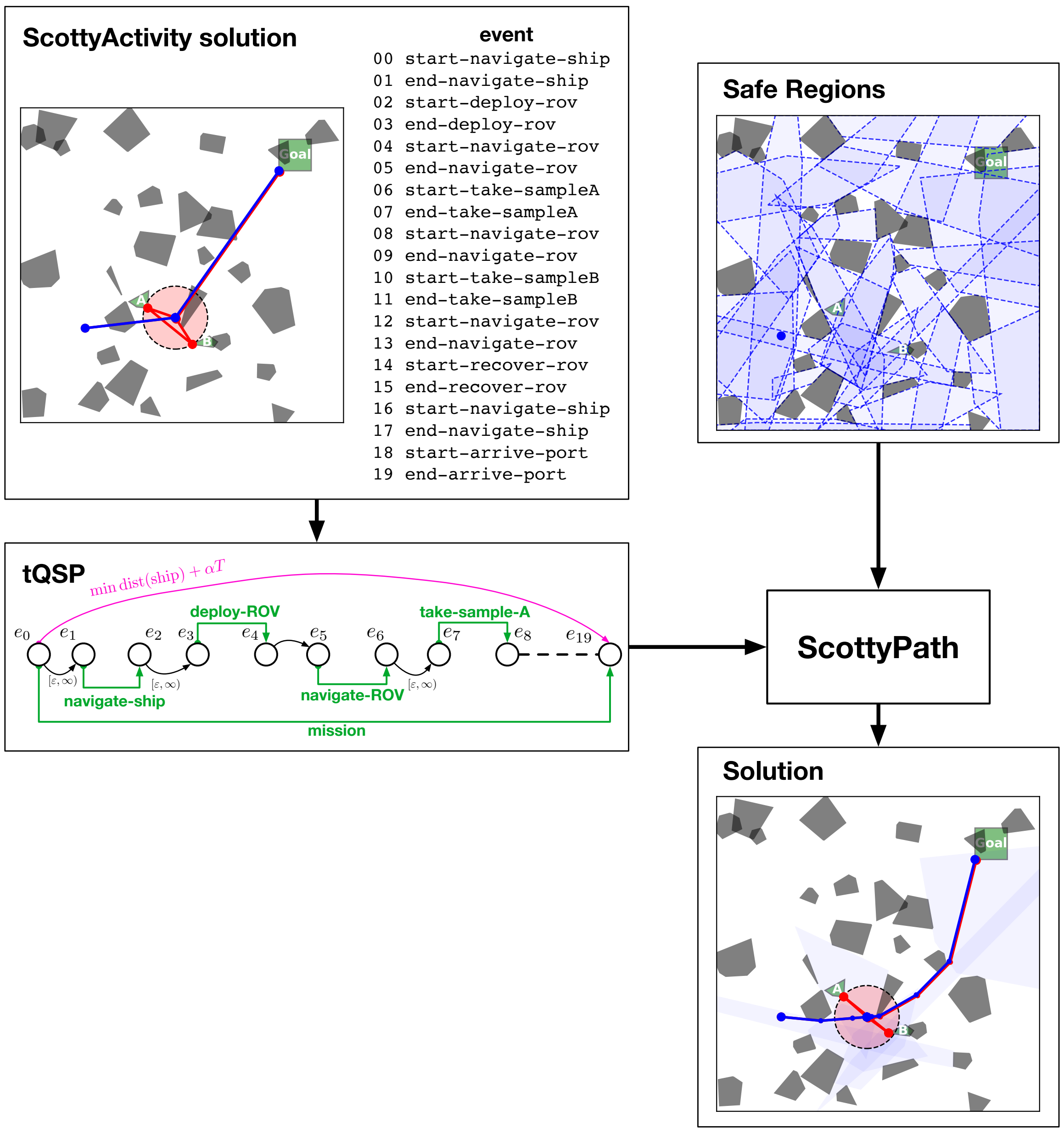
12.2 Generation of Problem Instances

In this section we describe, with an example, how we generate the benchmark problems from the solutions generated by ScottyActivity in Section 9.2.

Figure 12-2 shows how the tQSP problems that we use to benchmark ScottyPath are generated. For each instance of each domain, we retrieve the solution found by ScottyActivity. As shown in the figure, this solution ignores obstacles and consists, in part, of an ordered list of starts and ends of activity. The activities in the solution plan are the episodes, while their starts and ends are the events of the tQSP. The tQSP is generated as described in Section 11.3.2. Finally, the input to ScottyPath consists of this tQSP and a set of regions for each vehicle.

12.3 Results

In this section, we report the performance of ScottyPath in the benchmark problems described in this chapter. The benchmarks were performed on an Intel Core i7-5960X CPU 3.00GHz using Gurobi 7.5 as the SOCP solver. The results for the AUV, ROV and Air Refueling domains are shown in Tables 12.1 to 12.3. Each row in the table represents each of the twenty instances of the three domains (column labeled **P**). The column **Ne** shows the number of events in the tQSP associated with that problem



```
       event
00 start-navigate-ship
01 end-navigate-ship
02 start-deploy-rov
03 end-deploy-rov
04 start-navigate-rov
05 end-navigate-rov
06 start-take-sampleA
07 end-take-sampleA
08 start-navigate-rov
09 end-navigate-rov
10 start-take-sampleB
11 end-take-sampleB
12 start-navigate-rov
13 end-navigate-rov
14 start-recover-rov
15 end-recover-rov
16 start-navigate-ship
17 end-navigate-ship
18 start-arrive-port
19 end-arrive-port
```

Figure 12-2: This diagram shows how the tQSP problems used in this chapter are generated.

AUV domain

		Easy (9 regions)											Medium (28 regions)											Hard (107 regions)										
		$\alpha = 1.025$			$\alpha = 1.05$				$\alpha = 2$				$\alpha = 1.025$			$\alpha = 1.05$				$\alpha = 2$				$\alpha = 1.025$			$\alpha = 1.05$				$\alpha = 2$			
P	Ne	t	L	N	t	L	N	O	t	L	N	O	t	L	N	t	L	N	O	t	L	N	O	t	L	N	t	L	N	O	t	L	N	O
01	4	0.3	8	26	0.3	8	26	0.0	0.1	4	17	7.4	1.4	7	97	1.2	7	97	0.0	0.6	7	48	4.6	1.3	7	114	1.3	7	114	0.0	1.1	6	103	0.9
02	8	0.9	11	45	0.6	11	42	0.0	0.6	11	40	0.0	4.8	11	286	3.4	11	211	0.0	1.4	12	86	0.1	14.1	14	768	8.2	12	498	0.4	2.5	10	163	4.7
03	12	1.5	18	75	1.5	18	75	0.0	1.3	18	63	0.0	7.9	15	367	6.0	15	292	0.0	2.4	15	117	0.0	27.5	18	1356	13.9	18	569	0.0	5.2	16	246	3.3
04	16	2.7	24	95	2.5	24	95	0.0	2.2	24	83	0.0	11.7	22	470	12.0	22	469	0.0	4.1	21	158	5.5	35.0	26	1448	20.0	26	660	0.0	9.5	23	334	2.3
05	20	3.7	26	134	3.6	26	133	0.0	3.0	26	107	0.0	17.1	27	534	17.1	27	529	0.0	6.5	28	185	5.4	52.9	31	1607	28.7	31	800	0.0	14.9	29	426	1.9
06	20	3.2	26	109	3.2	26	108	0.0	2.6	26	85	0.0	22.4	29	773	22.5	29	768	0.0	19.5	29	623	3.1	44.1	31	1447	39.2	31	1343	0.0	17.3	32	482	1.6
07	24	4.6	30	133	4.4	30	132	0.0	4.1	30	119	0.0	24.8	33	747	24.2	33	746	0.0	20.5	37	589	3.5	83.8	40	2327	77.3	40	2127	0.0	21.5	37	533	4.9
08	24	5.9	32	158	5.4	32	147	0.0	4.2	30	118	0.7	44.1	34	1198	43.6	34	1193	-0.3	34.6	35	851	3.7	78.1	36	2171	68.8	36	1678	0.0	24.8	35	619	1.9
09	28	9.6	38	221	9.8	38	214	0.0	5.6	36	138	0.5	51.6	38	1255	50.7	38	1250	0.0	40.1	39	908	2.8	95.3	48	2313	80.0	48	1811	-0.0	33.2	43	719	2.7
10	28	7.0	34	185	6.8	34	184	0.0	4.9	34	134	0.0	32.9	35	824	32.0	35	813	0.0	17.2	37	390	0.6	81.0	46	2141	57.9	46	1374	0.1	31.0	41	652	0.5
11	32	8.2	38	202	8.1	38	200	0.0	5.3	38	125	0.0	41.9	41	910	41.2	41	906	0.0	21.6	41	421	0.4	111.6	52	2428	102.5	53	2210	0.1	41.9	48	757	0.8
12	32	10.3	38	242	10.2	38	240	0.0	7.3	38	171	0.0	50.0	45	1175	48.5	44	1144	0.2	31.2	43	800	2.1	129.0	52	2313	96.9	51	1738	0.2	39.3	49	687	3.7
13	36	11.9	42	258	11.6	42	256	0.0	8.8	42	187	0.0	73.6	49	1503	70.7	47	1469	0.2	37.6	46	881	2.1	159.6	56	2727	121.6	55	1996	0.2	46.3	51	785	4.2
14	36	13.3	51	248	13.3	51	246	0.0	9.3	49	186	0.4	114.3	51	2106	111.7	51	2079	0.0	49.2	53	786	3.2	241.0	54	4588	190.3	53	3459	0.5	54.1	54	931	0.9
15	40	15.6	57	259	14.4	57	248	0.0	11.6	53	203	8.2	113.1	55	1993	113.3	55	1948	0.0	36.4	55	584	2.8	252.2	61	4398	176.1	60	3113	0.0	58.4	57	932	0.6
16	40	19.8	55	336	19.0	55	331	0.0	11.0	50	211	1.0	121.6	54	2269	120.1	54	2236	0.0	84.7	56	1382	3.3	275.8	66	3842	139.1	65	1996	0.5	56.4	58	890	3.6
17	44	23.3	59	371	21.9	59	363	0.0	13.0	54	231	1.0	144.9	61	2499	139.6	62	2448	0.2	99.9	61	1605	3.5	341.8	72	4523	184.3	71	2598	0.5	70.7	63	963	3.9
18	48	23.6	61	361	20.2	61	320	0.0	16.0	61	257	0.0	172.1	66	2687	170.8	64	2644	-0.1	85.7	62	1294	3.9	283.2	71	4162	213.0	68	2735	0.3	76.0	65	1024	2.8
19	52	31.0	67	430	30.0	67	426	0.0	18.6	65	268	1.9	187.8	73	2789	187.5	72	2762	0.1	131.8	72	1871	2.6	341.1	80	4326	203.5	78	2602	-0.0	87.6	73	1108	2.3
20	56	35.0	74	464	34.1	74	464	0.0	21.4	72	286	1.7	203.6	77	2812	200.1	76	2787	0.1	144.6	77	1942	2.6	467.3	85	5246	247.5	82	2897	-0.1	103.3	77	1170	3.7

Table 12.1: Benchmarking results for simplified domains **t**: Planning time in seconds; **L**: Plan length; **S**: Number of nodes expanded; **N**: Number of optimization problems solved; **T**: Mean optimization time for each optimization problem in milliseconds.

ROV domain

		Easy (9 regions)											Medium (29 regions)											Hard (111 regions)										
		$\alpha = 1.025$			$\alpha = 1.05$				$\alpha = 2$				$\alpha = 1.025$			$\alpha = 1.05$				$\alpha = 2$				$\alpha = 1.025$			$\alpha = 1.05$				$\alpha = 2$			
P	Ne	t	L	N	t	L	N	O	t	L	N	O	t	L	N	t	L	N	O	t	L	N	O	t	L	N	t	L	N	O	t	L	N	O
01	16	6	17	120	6	17	120	0.0	6	17	120	0.0	60	21	1246	59	21	1247	0.0	10	18	184	7.0	227	22	4346	214	22	4182	0.0	36	22	620	19.9
02	20	7	20	141	6	20	114	0.1	8	21	133	0.5	61	25	1016	60	25	1001	0.0	16	23	254	6.5	23	27	330	22	27	299	0.3	19	26	287	17.3
03	24	11	24	169	8	24	127	0.1	11	25	152	0.5	88	30	1217	83	28	1217	0.1	22	27	295	6.4	31	31	374	27	31	331	0.3	25	30	311	17.1
04	36	43	37	472	37	37	413	0.0	22	37	220	0.0	52	40	460	51	40	460	0.0	40	40	366	0.0	438	43	3398	103	44	876	0.0	65	42	526	4.6
05	40	42	41	427	40	41	428	0.0	24	41	236	0.0	129	44	1321	69	45	592	0.0	45	45	381	1.5	183	48	1421	148	49	1167	0.0	81	45	623	0.6
06	52	59	53	476	60	53	475	0.0	42	55	302	1.0	92	56	616	92	56	600	0.0	74	56	479	0.6	580	60	3455	182	59	1161	0.0	124	60	748	1.8
07	56	65	57	489	66	57	489	0.0	45	57	304	0.0	191	60	1436	188	60	1423	0.0	78	62	476	2.0	171	66	1003	153	64	922	0.3	151	65	876	4.8
08	68	116	69	686	118	69	686	0.0	69	71	361	0.3	339	76	1940	338	76	1938	0.0	127	76	627	3.3	195	79	862	178	78	820	0.0	183	77	871	4.1
09	72	108	73	614	106	73	611	0.5	77	73	410	0.5	318	81	1797	316	81	1778	0.0	123	79	575	3.4	1138	89	5223	930	87	4148	0.1	237	83	1021	4.2
10	84	148	87	689	144	87	688	-0.0	106	87	463	1.4	417	92	2008	408	91	1999	0.1	171	93	694	0.1	TO	F	5390	TO	F	5308	-	332	99	1154	-
11	88	155	91	705	156	91	704	0.0	115	91	477	1.4	437	96	2032	428	95	2023	0.1	196	99	755	1.0	TO	F	5003	TO	F	5196	-	370	103	1225	-
12	100	304	105	1146	309	105	1146	0.0	153	104	548	0.3	759	111	2666	735	111	2565	0.0	316	115	985	10.8	722	119	2385	624	119	2041	0.0	499	115	1649	7.5
13	104	331	109	1202	337	109	1202	0.0	167	108	568	0.3	807	115	2730	792	115	2606	0.0	342	120	1008	10.7	766	122	2443	656	122	2098	0.0	552	119	1717	7.4
14	108	376	113	1290	375	113	1290	0.0	191	113	614	0.3	843	119	2785	840	119	2660	0.0	391	124	1082	10.7	857	127	2577	733	127	2220	0.0	570	123	1770	7.5
15	120	390	131	1156	382	131	1153	0.0	314	131	920	3.5	1153	140	3199	1049	140	2961	0.0	455	139	1188	11.0	TO	F	4196	1083	144	2870	-	691	139	1949	-
16	124	194	F	664	191	F	664	-	190	F	660	-	1179	144	3269	1121	144	3012	0.0	480	142	1225	11.0	TO	F	4187	1121	147	2924	-	729	142	1991	-
17	136	205	F	664	207	F	664	-	206	F	661	-	1152	156	2938	1076	156	2772	0.1	641	154	1463	8.7	TO	F	3892	TO	F	2943	-	1030	155	2539	-
18	140	210	F	664	214	F	664	-	209	F	662	-	1189	161	2988	1062	158	2730	0.4	638	156	1428	8.0	TO	F	3748	TO	F	2895	-	TO	F	2833	-
19	144	214	F	664	213	F	664	-	212	F	663	-	TO	F	2972	1118	162	2763	-	637	160	1445	-	TO	F	3628	TO	F	2844	-	TO	F	2763	-
20	156	229	F	664	224	F	664	-	227	F	662	-	TO	F	2888	1173	174	2811	-	708	174	1507	-	TO	F	3521	TO	F	2771	-	TO	F	2713	-

Table 12.2: ROV Benchmarking results for simplified domains **t**: Planning time in seconds; **L**: Plan length; **S**: Number of nodes expanded; **N**: Number of optimization problems solved; **T**: Mean optimization time for each optimization problem in milliseconds.

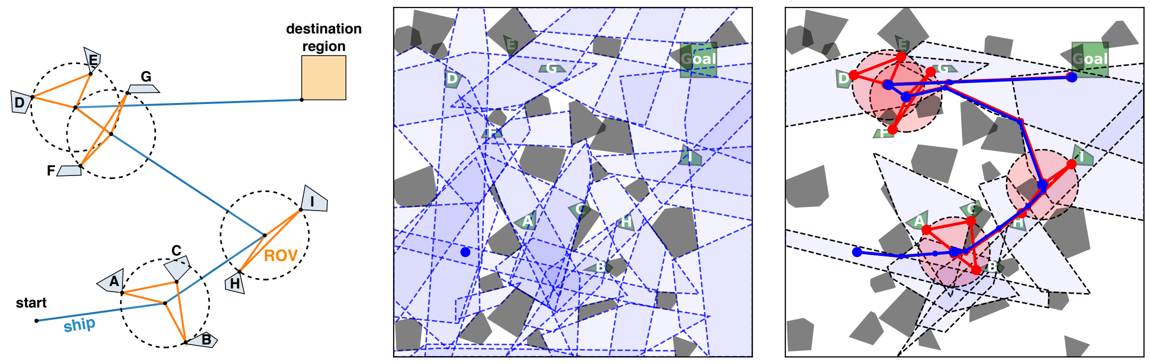
Air Refueling domain

		Easy (9 regions)											Medium (29 regions)											Hard (107 regions)										
		$\alpha = 1.025$			$\alpha = 1.05$				$\alpha = 2$				$\alpha = 1.025$			$\alpha = 1.05$				$\alpha = 2$				$\alpha = 1.025$			$\alpha = 1.05$				$\alpha = 2$			
P	Ne	t	L	N	t	L	N	O	t	L	N	O	t	L	N	t	L	N	O	t	L	N	O	t	L	N	t	L	N	O	t	L	N	O
01	8	6	13	133	6	13	133	0.0	6	13	126	4.2	394	22	5824	351	19	4981	0.6	26	17	396	3.2	TO	F	18444	TO	F	17971	0.0	144	20	2000	0.0
02	12	1	F	-	1	F	-	-	2	F	-	-	TO	F	8392	1166	F	8496	-	1180	F	8418	-	2	F	-	2	F	-	0.0	2	F	-	0.0
03	16	24	22	257	28	26	272	0.5	36	29	310	15.0	157	36	1081	103	33	817	0.2	110	37	810	11.3	TO	F	13899	TO	F	13698	0.0	TO	F	9197	0.0
04	18	40	26	380	29	27	264	0.0	35	26	306	9.6	TO	F	11230	TO	F	11830	-	305	45	1893	-	TO	F	9286	TO	F	8021	0.0	588	51	2867	0.0
05	20	252	33	2017	242	32	1955	0.4	60	28	478	7.3	TO	F	11143	TO	F	9843	-	304	45	1888	-	TO	F	7857	TO	F	7870	0.0	888	53	4375	0.0
06	22	434	F	2800	414	F	2779	-	365	F	2806	-	TO	F	8825	TO	F	8777	-	596	47	2745	-	TO	F	6899	TO	F	6179	0.0	TO	F	5338	0.0
07	28	121	41	730	99	40	644	0.0	72	42	404	7.5	TO	F	8191	TO	F	7228	-	283	47	1419	-	TO	F	8568	TO	F	8484	0.0	TO	F	4518	0.0
08	30	157	40	876	128	40	687	0.0	76	44	432	9.9	TO	F	8003	TO	F	8370	-	420	69	1618	-	TO	F	7004	TO	F	6238	0.0	780	71	2817	0.0
09	32	112	39	665	123	42	669	0.0	83	42	422	3.7	TO	F	5469	TO	F	4841	-	254	48	1055	-	TO	F	6969	TO	F	7196	0.0	TO	F	3543	0.0
10	34	319	46	1615	326	48	1522	0.0	116	48	510	1.5	TO	F	5426	TO	F	5140	-	371	57	1357	-	TO	F	7085	TO	F	6744	0.0	TO	80	4773	0.0
11	10	25	18	256	25	18	256	0.0	19	17	238	9.2	TO	F	7390	TO	F	6861	-	395	22	3025	-	TO	F	13163	TO	F	12166	0.0	TO	F	7097	0.0
12	14	184	20	1309	119	20	867	0.0	35	20	287	1.0	TO	F	12726	TO	F	12447	-	1047	35	6133	-	TO	F	9792	TO	F	8184	0.0	TO	F	6867	0.0
13	16	77	23	519	43	23	323	-0.0	45	23	336	4.1	TO	F	10898	TO	47	101543	-	TO	F	7260	-	TO	F	9969	TO	F	9178	0.0	TO	F	5697	0.0
14	18	79	25	497	51	25	356	-0.0	49	25	366	10.4	TO	F	10505	TO	F	9787	-	1039	43	6160	-	TO	F	7807	TO	F	6835	0.0	TO	F	7838	0.0
15	22	277	39	1310	177	36	928	0.0	111	33	573	2.1	TO	F	9225	TO	F	8231	-	TO	F	4649	-	TO	F	9937	TO	F	9668	0.0	TO	F	8056	0.0
16	24	680	43	2743	653	43	2638	-0.0	193	39	787	5.7	TO	F	7749	TO	F	7379	-	1113	61	3222	-	TO	F	7685	TO	F	8285	0.0	TO	F	7973	0.0
17	26	845	41	3797	842	42	3660	-0.0	195	40	743	0.5	TO	F	5765	TO	F	5508	-	TO	F	3463	-	TO	F	7597	TO	F	7567	0.0	TO	F	7225	0.0
18	32	TO	F	4461	TO	F	4335	-	657	63	1566	-	TO	F	5296	TO	F	4885	-	TO	F	2395	-	TO	F	4172	TO	F	4459	0.0	TO	F	4151	0.0
19	34	TO	F	4001	TO	F	3805	-	547	62	1232	-	TO	F	5925	TO	F	5776	-	TO	F	2741	-	TO	F	4518	TO	F	4196	0.0	TO	F	3590	0.0
20	36	64	F	274	65	F	274	-	67	F	276	-	TO	F	2996	TO	F	3106	-	TO	F	2659	-	TO	F	2792	TO	F	2624	0.0	TO	F	3356	0.0

Table 12.3: AirRefueling Benchmarking results for simplified domains **t**: Planning time in seconds; **L**: Plan length; **S**: Number of nodes expanded; **N**: Number of optimization problems solved; **T**: Mean optimization time for each optimization problem in milliseconds.

instance. Recall that the tQSP is generated from the ScottyActivity solution and that it does not depend on the obstacles or safe regions. Each table is divided into three sections, corresponding to the 'easy', 'medium' and 'hard' environments. Each of these is divided into three groups, which correspond to the three settings for the heuristic weight.

The experiments were run with a time limit of twenty minutes (1200 seconds). The column labeled **t** shows ScottyPath's runtime in seconds. Entries labeled 'TO' indicate that ScottyPath could not solve the problem within the time limit. The column labeled **L** shows the number of stages in the solution found by ScottyPath. Recall that each stage assigns a safe region to each vehicle and corresponds to each segment of the piecewise control trajectory. Entries labeled 'F' indicate that ScottyPath was not able to find a solution, either because the planner timed out or because the search space was exhausted and no feasible solution was found. The column labeled **N** shows the number of search nodes expanded. Finally, column **O** shows the relative increase in the objective value that the solution achieves compared to the objective achieved with the heuristic weight 1.025. Positive values indicate that the objective is larger and the solution is, therefore, worse. Entries labeled '-' indicate



(a) (b) (c)

Figure 12-3: Results for problem 9 of the ROV domain. (a) shows the solution found by ScottyActivity, in the absence of obstacles, using obj-EHC. (b) shows the arrangement of the surface obstacles and safe regions for the ship, for the 'medium' environment. Finally, (c) shows the obstacle free plan found by ScottyPath, with the ship path and the chosen safe regions in blue, and the ROV path in red.

that the relative objective increase could not be computed because ScottyPath could not find a solution using the heuristic weight of 1.025 within the time limit.

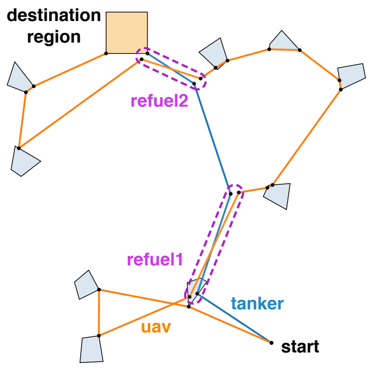
With a single vehicle, like in the AUV domain, the results scale fairly well. The hardest problem in the 'easy' environment, which consists of 9 safe regions, is solved in 35 seconds when the $h_w = 1.025$ setting us used. The 'medium' environment, which consists of about 30 safe regions, takes, in average, 6 to 7 times longer to solve for each of the heuristic weight settings. However, the 'hard' environment, which consists of 107 safe regions, takes only twice as much time, in the worst case, to solve as the 'medium' environment. All the problems in the AUV domain were solved within the time limit, regardless of the environment used or the heuristic weight setting. The hardest problem solved is instance 20 using the 'hard' environment and the $h_w = 1.025$ setting. The solution to this problem, found in 470 seconds, finds a path for the AUV visiting 14 regions. This path consists of 85 stages, where each has associated a safe region and a constant control value.

The ROV domain is significantly harder, and Table 12.2 illustrates this fact. In this domain there are two vehicles: the ship and the ROV. Only the ship needs to avoid obstacles and, therefore, it needs safe regions. However, both vehicles are considered jointly since they always need to observe the tether range constraint.

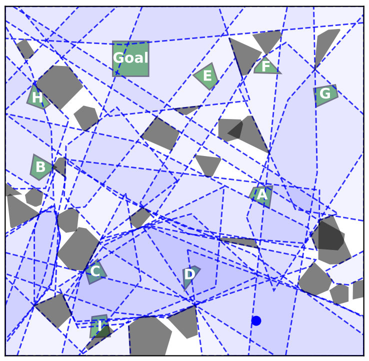
Moreover, typical tQSPs in this domain are much longer than in the AUV case, as there are episodes for deploying and recovering the ROV, multiple navigation episodes for each vehicle and sampling episodes. In fact, the tQSP for instance 20 of the ROV domain has 156 events, compared to the 56 events in the tQSP of the largest instance in the AUV domain. Figure 12-3 shows the solution to instance 9 when using the 'medium' environment and the $h_w = 1.05$ setting. The left figure shows the plan found by ScottyActivity in the absence of obstacles. This plan forms the tQSP that ScottyPath solves. The center figure shows the 'medium' environment, which consists of 29 regions for the ship.

The ROV domain takes considerably more time to solve than the AUV domain. The runtime penalty from environment to environment is roughly similar to that in the AUV domain. Runtime is about 3-4 times slower when switching from the 9 safe regions in the 'easy' environment to the 29 regions in the 'medium' environment. Runtime is about twice as long, in the worst case, in the 'hard' environment with 111 regions than in the 'medium' environment. In the ROV domain, ScottyPath is not able to solve, within the time limit, the last two instances using the 'medium' environment and the most demanding heuristic weight setting. Furthermore, ScottyPath also times out in many instances when using the 'hard' environment. Note also that ScottyPath is unable to solve instances 16 to 20 when using the 'easy' environment, and that these failures are not due to the time limit. In these cases, ScottyPath is unable to find a solution because the arrangement of obstacles and safe regions is not compatible with the tQSP found by ScottyActivity. For example, the tQSP in one of those instances may require the ROV to reach multiple target sampling regions without moving the ship. However, this constraint is impossible to satisfy with the safe regions in the environment. This is a limitation of our approach, that separates the activity planning problem from the motion planning problem.

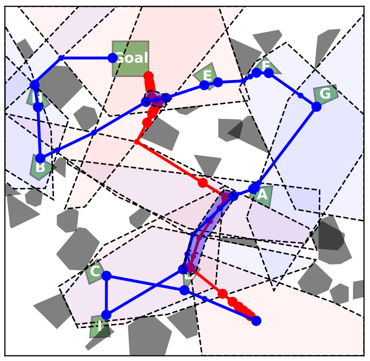
The AirRefueling domain is the hardest domain. In this domain, both the tanker plane and the UAV need to avoid obstacles. Moreover, the UAV has a limited battery whose capacity decreases as a function of its speed. Instances 11-20 have an additional UAV that needs to avoid obstacles as well. Figure 12-4 shows the solution found for



(a)



(b)



(c)

Figure 12-4: Results for problem 9 of the AirRefueling domain. (a) shows the solution found by ScottyActivity, in the absence of obstacles, using obj-EHC. (b) shows the arrangement of the no-fly zones and safe regions for both the tanker and the UAV, for the 'medium' environment. Finally, (c) shows the obstacle free plan found by ScottyPath, with the tanker path and its safe regions in red, and the UAV path and its safe regions in blue.

instance 9. The left figure shows the plan found by ScottyActivity, which is used to define the tQSP problem. The 'medium' environment for this domain is shown in the center figure. Finally, the right figure shows the solution found by ScottyPath when using the $h_w = 2.0$ setting.

The results for the AirRefueling domain are shown in Table 12.3. ScottyPath can solve most problems within the time limit when using the 'easy' environment, regardless of the heuristic weight setting. However, increasing the number of safe regions has a large performance penalty in this problem. Using the two most demanding heuristic weight settings, ScottyPath can only solve a few problems when using the 'medium' environment, and none when using the 'hard' environment. Using the $h_w = 2.0$ setting, ScottyPath can solve most problems when using the 'medium' environment, but it is unable to solve most within the time limit when using the 'hard' environment. These results illustrate the difficulty of assigning safe regions to multiple vehicles when the number of possible safe regions is large.

Finding optimal solutions is hard when the tQSPs are long or the number of safe regions is large. However, it is possible to use weighted A* in order to obtain solutions quicker by trading solution quality. Figures 12-5 to 12-7 show examples of

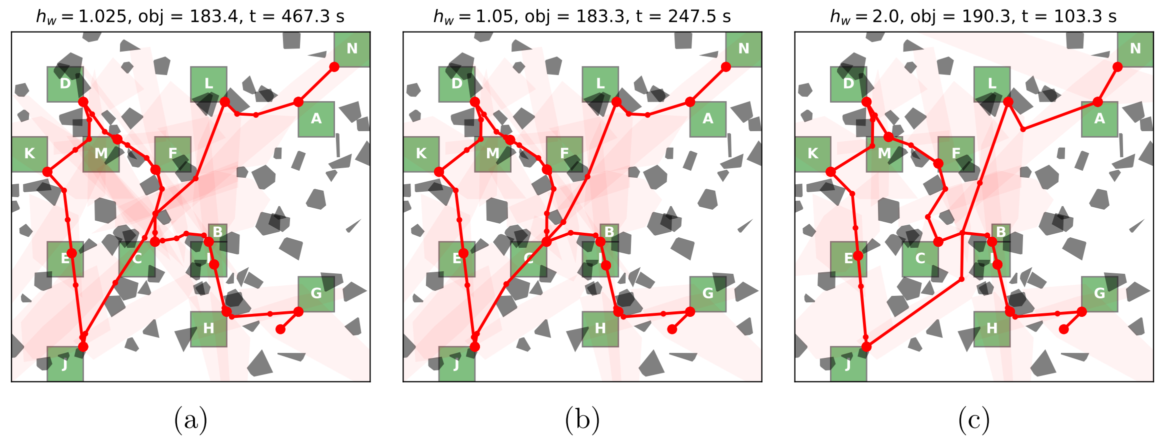


Figure 12-5: Results for instance 20 of the AUV domain using the ‘hard’ environment.

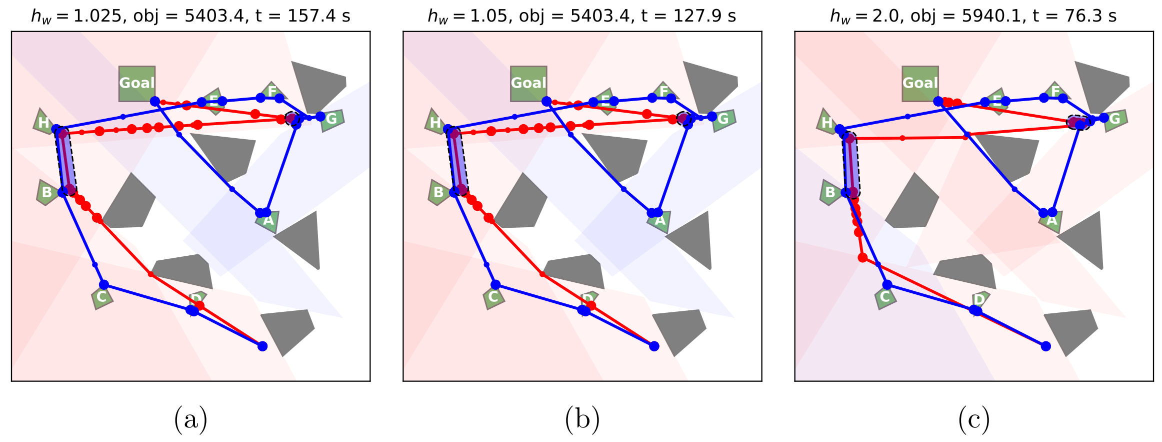


Figure 12-6: Results for instance 8 of the Air Refueling domain using the ‘easy’ environment.

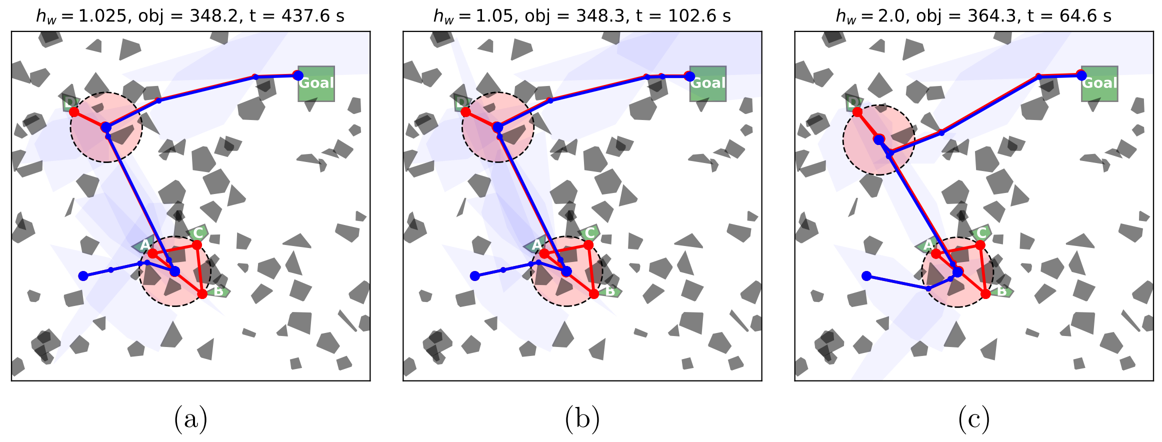


Figure 12-7: Results for instance 4 of the ROV domain using the ‘hard’ environment.

solutions found for the same problem when using the three heuristic weight settings: $h_w = 1.025$, $h_w = 1.05$, and $h_w = 2.0$. These heuristic weight settings ensure that the returned solution has an objective within 2.5%, 5% and 200% of the optimal objective. While the best solution quality should be found with the $h_w = 1.025$ setting, our results show that, in practice, the $h_w = 1.05$ setting produces almost always the same solution, while using considerably less time. For example, Figure 12-5 shows that the solution obtained using $h_w = 1.05$ is slightly better than the solution obtained with $h_w = 1.025$, while taking almost half as long to solve.

As expected, large heuristic weight values, such as $h_w = 2.0$, often produce significantly suboptimal plans. For example, the tanker trajectory, shown in red in Figure 12-6c, is significantly longer than it should be, when compared to the trajectory returned using the other heuristic weight settings (Figures 12-6a and 12-6b. However, we also found that, sometimes, using a large heuristic weight can reduce the planning time drastically without having an equally large loss in the solution quality. As an example, the solution found using $h_w = 2.0$, shown in Figure 12-7c, is very similar to the solution found using the $h_w = 1.025$ setting, shown in Figure 12-7a, but it was computed roughly an order of magnitude faster.

12.4 Chapter Summary

In this chapter, we evaluated the scalability of ScottyPath with respect to the size of the input tQSP, the number of vehicles and the number of safe regions. Our results showed that ScottyPath scales well with the size of the tQSP in environments with a moderate number of safe regions. Our results also show that the runtime of the planner increases significantly when multiple vehicles are used and when the number of safe regions is large. However, we show that, by trading some solution quality, it is possible to significantly improve the runtime of the planner.

Part IV

Conclusions

Chapter 13

Conclusions

13.1 Summary of Contributions

As robots become increasingly capable and accessible, there is an unfulfilled need for controlling them efficiently and autonomously. In order to narrow the gap between the needs of robotic missions and the capabilities of state of the art automated planning techniques, this thesis presented the Scotty Planning System. The Scotty Planning System is an activity and trajectory planning framework designed to efficiently plan for long term missions involving coordinated robotic vehicles.

In order to address this problem, I presented five contributions in this thesis.

1. **An Architecture for Planning and Execution of Missions With Multiple Coordinated Robotic Vehicles**. My first contribution is an architecture for solving hybrid activity and trajectory planning problems that involve coordinated robotic vehicles over long horizons. This problem is hard due to its highly combinatorial nature, which arises from the activity selection and the avoidance of obstacles. We solve this problem by handling the two combinatorial aspects separately with two planners. The first planner solves the activity planning problem, and generates a plan skeleton, in the form of a qualitative state plan (QSP), that specifies a sequence of behaviors for each robot that satisfy the mission goals. The second planner finds control and collision-free

trajectories for each robot while satisfying all the time, state and control constraints in the plan skeleton. We handle both combinatorial aspects separately in order to leverage heuristics that are better suited for each of those problems. Finally, we propose a receding horizon executive that combines a discrete-time model with more accurate dynamics and multi-vehicle obstacle avoidance for a limited horizon, and a continuous time model with first-order dynamics for the rest of the plan.

2. **An Architecture for Solving Combinatorial Planning Problems**. Both the activity planning problem and the obstacle avoidance problem involve discrete choices that are highly combinatorial. The two planners that we present in this thesis to solve this problem use the same master-slave architecture. The master uses heuristic forward search to make the discrete combinatorial choices. The slave is a subplanner that solves a relaxed problem in order to test the consistency of candidate solutions and to compute their heuristic value. Our method of making the discrete choices differs from other traditional approaches that use mixed-integer solvers. These often use a branch and bound search algorithm where problems where the integer constraints are relaxed are repeatedly solved in order to compute bounds. The results that we present in this thesis show that our heuristic forward search method performs orders of magnitude faster, since we leverage specialized heuristics that allow us to navigate the discrete combinatorial space more efficiently. Each of the two planners, ScottyActivity and ScottyPath, use the same subplanner, ScottyConvexPath, that solves a relaxed planning problem in which the activities and their order are fixed and the environment is obstacle-free.

3. **A Convex, Goal-directed Trajectory Planner for Coordinated Robotic Vehicles Over Long Horizons**. The third contribution in this thesis is ScottyConvexPath, a goal-directed trajectory planner. ScottyConvexPath solves the problem of simultaneously finding the schedule and the control and state robot trajectories for a given plan skeleton. ScottyConvexPath solves a relaxed

problem in which obstacles are not considered, and where the activities are specified by the plan skeleton. Since there are no obstacles and the order of the activities is specified by the plan skeleton, the relaxed problem that ScottyConvexPath solves does not involve combinatorial choices. A key contribution of this thesis is a method for solving this relaxed problem efficiently over long horizons while respecting the robot dynamics and constraints. This is achieved by reformulating the problem as a second order cone program (SOCP). Our formulation respects multi-vehicle coordination constraints and first order dynamics and does not require discretization of either time, control or state. Our encoding uses convex quadratic constraints to represent velocity controlled dynamics and to model common effects, such as velocity dependent battery consumption. In order to maintain a continuous time formulation, our encoding avoids non-convex terms that arise in straightforward formulations. Because our planner operates on continuous time, it is well suited for finding robot trajectories over long horizons. ScottyConvexPath constitutes a core element of this thesis. Both ScottyActivity and ScottyPath use ScottyConvexPath as their subplanner. The SOCP encoding that ScottyConvexPath uses is also essential for MPC-Scotty, the executive that is being developed as a future extension to our system.

4. **A Hybrid Activity and Trajectory Planner Based on Heuristic Forward Search**. The fourth contribution in this thesis is ScottyActivity, a state of the art hybrid activity and trajectory planner. ScottyActivity solves the hybrid problem of simultaneously selecting the activities along their schedule, and finding the robot control and state trajectories that satisfy the goal constraints in an obstacle-free environment. ScottyActivity implements hybrid activity and trajectory planning as heuristic forward search with delete relaxations. ScottyActivity uses ScottyConvexPath to test the consistency of candidate action plans and to compute the objective lower bound that complements the delete relaxation heuristic in the case of heuristic ties. ScottyActivity performs as fast as other state of the art HFS activity planners, but is much more expres-

sive. Since ScottyConvexPath operates on continuous-time, ScottyActivity can solve problems with longer horizons than other hybrid planners, at the expense of using simple, first-order dynamics. Contrary to most activity planners, our rich representation of activities allows ScottyActivity to generate plans that are flexible in time, state and control, which makes them suitable for online execution. Within the Scotty Planning System, ScottyActivity generates candidate activity plans, for which ScottyPath later finds collision-free trajectories.

5. **A Qualitative State Plan Path Planner for Coordinated Robots With Obstacles**. Finally, this thesis presented ScottyPath, an optimal QSP planner for multiple coordinated vehicles that avoids collisions with obstacles. The qualitative state plan that ScottyPath takes as its input is the plan skeleton that ScottyActivity generates and that specifies the activities that are executed and their order. These activities specify the constraints that the robots are subject to, as well as their dynamics. ScottyPath finds optimal control and collision-free trajectories for multiple coordinated robotic vehicles that satisfy the constraints in the plan skeleton. In order to guarantee collision-free trajectories, ScottyPath finds paths that ensure that each vehicle is always inside a set of overlapping convex safe regions, which are generated in advance. Similar to ScottyActivity, ScottyPath solves the problem by performing heuristic search over a combinatorial space. ScottyPath uses ScottyConvexPath to compute the cost and heuristic of candidate plans. The combinatorial space that ScottyPath explores represents the traversals through sequences of convex safe regions. Since ScottyConvexPath operates on continuous time, ScottyPath can efficiently find plans over long horizons. Our results show that our heuristic search method performs two orders of magnitude faster than mixed-integer approaches.

13.2 Future Work

There are many interesting avenues for future research related to the Scotty Planning System. We describe some of them in this section.

Execution of Scotty Plans with Higher Fidelity Dynamics and Constraints

The Scotty Planning System, as described in this thesis, computes grounded plans that contain a control trajectory for each vehicle. These plans cannot be executed directly for two reasons. First, these plans assume that there is no uncertainty and that no deviations occur during the execution of the plan. Second, the vehicle dynamics are represented with a simplified first order model that does not correspond to reality. However, the Scotty Planning System can generate flexible plans in terms of time, control and state. These flexible plans are QSPs and are given by the set of constraints that the solution plan satisfies. In order to address the two issues described earlier, we propose a receding horizon planner, that we refer to as *MPC-Scotty*, that executes the QSP obtained with the Scotty Planning System. MPC-Scotty combines a discrete-time detailed model for a limited horizon with a simpler continuous-time model that is used for the rest of the plan into a single mixed-integer second order cone program (MISOCP). By using a discrete-time formulation for a limited detailed horizon, MPC-Scotty can handle accurate dynamics and vehicle to vehicle collisions. By also considering a continuous-time model with the same first order approximations used by Scotty for the rest of the plan after the horizon, the solutions are guided beyond the detailed horizon. This allows MPC-Scotty to produce solutions with better quality than other MPC approaches that do not reason beyond the planning horizon. It also allows us to detect, during execution, early failures that may occur beyond the horizon. MPC-Scotty is being developed in the MIT MERS lab and an initial version is already solving problems similar to the ones we presented in this thesis.

Full Integration of Activity and Trajectory Planning with Obstacles

In this thesis, we presented an approach that performs planning for robotic missions by breaking the problem into the activity planning problem and the path planning problem of the resulting QSP solution. The activity problem that ScottyActivity solves considers robot trajectories with their dynamics and coordination constraints. However, the resulting QSP solution that ScottyActivity generates, and that is the input to ScottyPath, does not consider obstacles. Therefore this approach is not complete nor optimal. In effect, there could be solution QSPs generated by ScottyActivity that cannot be solved due to the presence of obstacles. As we describe in Section 12.3, this happened a few times in our benchmark problems. In order to solve this issue, the activity planning, the trajectory planning and the path planning problems need to be solved jointly. This could be done by using ScottyPath as a consistency checker for the partial plans defined by the search nodes in the ScottyActivity search. That is, instead of using the convex model to test the consistency of partial plans, ScottyActivity would use ScottyPath directly. The challenge is that this approach would require ScottyPath to be significantly faster than what it is at the moment. The consistency checks that ScottyActivity currently performs with the convex model take in the order of milliseconds. The same checks, with the addition of obstacles, performed by ScottyPath would probably take a few seconds. Given the large number of states that ScottyActivity expands while solving typical planning problem, this approach is not currently feasible. However, the performance of ScottyPath could be improved to lessen the impact of this issue.

Activity Planning for Qualitative State Plans

The PDDL-S activity planning problems that ScottyActivity solves only specify initial conditions and goal conditions at the end of the plan. However, many robotic missions require a more flexible specification for goal conditions given as temporally evolved goals in the form of a QSP. Such QSP describes goal constraints for the robots at different steps in the mission. Some planners are capable of performing activity planning for QSPs. For example, Kongming [63, 62] is capable of planning for QSPs by performing a clever encoding that converts each temporally extended goal to an activities that only become feasible when its prior goals are satisfied. This encoding reduces the problem to a traditional problem with an initial condition and a goal condition at the end of the plan. Perhaps more interestingly, the tBurton planner [94, 93] was designed to handle QSPs by using a novel approach based on causal graph decomposition. This is an interesting approach that could be used to extend

Scotty's capabilities. However, tBurton is a temporal planner that was not designed to handle continuous states or control variables, and a number of challenges need to be solved in order to incorporate the causal graph decomposition ideas into Scotty.

Optimal Activity Planning With an Objective Guided Heuristic

Similar to most heuristic forward search activity planners, the heuristic value that ScottyActivity computes is the remaining number of activities needed to reach the goal in the relaxed planning graph. This heuristic does not consider the problem objective in any way. Therefore, the heuristic does not guide the search towards plans with smaller objective values, but to plans with a fewer number of activities. The obj-EHC search algorithm that we presented in Section 8.4 tries to mitigate this issue by breaking heuristic ties with the objective function. However, it would be desirable that the heuristic estimated the cost to go according to the problem objective. This is not straightforward for two reasons. First, estimating the cost to go would involve, for each heuristic computation, solving an additional optimization problem on the relaxed planning graph which could have a significant performance penalty. Second, the relaxed planning graph ignores delete effects, which means, for example, that a vehicle is allowed to visit more than one region at a time. The optimization program used to estimate the remaining cost would have to use relaxed constraints to handle these inconsistencies. As an example, the program could consider a relaxation consisting of the convex hull of all the regions selected for a vehicle at a given layer, instead of having to choose a particular one. Other challenges, such as determining which continuous effects in activities in the relaxed planning graph are applied and when would have to be solved. However, such heuristic would allow ScottyActivity to generate significantly higher quality solution plans, and is, therefore, an interesting avenue for future research.

QSP Path Planning for unordered QSPs

While ScottyPath is designed to work in conjunction with ScottyActivity, it is a very expressive and capable planner on its own. In fact, the MIT MERS team will be using it to plan obstacle-free trajectories for multiple coordinated robots in an ocean exploration mission in Hawaii that will be led by WHOI in the beginning of 2018. However, ScottyPath is limited, in its current form, to planning for QSPs that are totally ordered (tQSPs). This is a strong limitation, as it requires the order of all the events in the QSP to be known in advance. This is not always the case. For some missions it is desirable that the planner makes the choice of which sampling region should be visited first. To remove this limitation, ScottyPath could use a similar approach like the one we presented in Chapter 10 that the geometric path planner uses to handle multiple unordered region goals. Recall that, in order to plan for unordered region goals, our approach solves, for each search node, a mixed-integer SOCP that jointly computes the cost, the heuristic value and the order of the remaining goals. A similar approach could be used to allow ScottyPath to handle general QSPs. However, it remains to be seen whether the performance penalty incurred in solving such mixed-integer program for each search node would be too severe.

An Improved Path Planning Strategy For Multiple Vehicles

Since the semantics of QSP problems allow arbitrary convex quadratic constraints between vehicles, ScottyPath needs to consider all vehicles jointly. Therefore, node expansions require selecting the next convex safe region for each vehicle. As a consequence, the search branching factor increases significantly with the number of vehicles. This results in long planning times for planning problems with few vehicles. However, in many robotic missions, vehicles are not jointly constrained, or are only jointly constrained during certain parts of the mission. A more efficient approach for handling multiple vehicles would separate the parts of the plan where a vehicle or a set of vehicles can be assigned safe region paths that are independent from the rest of the vehicles, which would significantly reduce the search branching factor.

Chance-constrained Planning for Robotic Missions

The Scotty Planning System currently assumes that the state of the world and the model of the robots are perfectly known. Since this is often not the case in the real world, we propose that, during execution, a detailed receding horizon planner is used to refine the plans produced by Scotty. However, over the last few years, a number of interesting techniques have arisen that can reason with models of uncertainty. These approaches are able to find plans that guarantee, with a certain degree of confidence, that constraints are not violated. These plans are often called chance-constrained plans. For example, the PSulu planner [74, 10, 73, 75] uses a method called iterative risk allocation (IRA) to generate chance-constrained motion plans for mobile robots that provide guarantees about the safety of the robot. Similarly, the RAO* [83] algorithm has recently been used in the CLARK architecture to perform generative planning for robots [84, 82]. CLARK only reasons with temporal activities with discrete conditions and effects. However, a future extension to Scotty would greatly benefit from the ideas demonstrated by PSulu and CLARK in order to generate chance-constrained plans that take into account models of uncertainty.

A More Expressive QSP Path Planner to Allow For More Realistic Missions

Inspired by our close collaboration with Woods Hole Oceanographic Institution, there are a number of things that the Scotty Planning system should support in order to become a more useful tool for planning oceanographic science missions in the future. We discuss some of them here:

- **Optimal Planning with Knowledge of Underwater Currents**. Nowadays, it is not uncommon to have access to current prediction models, such as the ones provided by NOAA, in the areas where scientific expeditions take place. Similarly, many ships used in these expeditions use an acoustic Doppler current profiler (ADCP) in order to measure the local currents in the area. Since some of the robotic platforms that are commonly used in long term missions are low powered and have a limited ability to fight strong currents, it would be desirable to find plans that optimize the battery usage by taking into account the current profile. ScottyPath could make use of this information by incorporating the current velocity into the safe regions.

- **Planning for Coordinated Robots with Different Dynamics**. The typical WHOI missions involve heavy and slow vehicles, such as the ship, small and agile vehicles, such as AUVs, and vehicles with limited maneuverability, such as gliders. In order to effectively coordinate them, it is important to consider their accurate dynamics. For example, some AUVs may be able to descend a steep underwater hill while maintaining a constant distance to the bottom, but unable to climb the same hill without colliding with it.

- **Planning in the Presence of Mobile Obstacles**. Some of the ocean exploration missions that WHOI is interested in take place to shipping lanes, which are used by large ships. While it is desirable to avoid the shipping lanes, it is often necessary to cross them at certain parts of the mission. There are nowadays information systems that provide the position of large ships traversing these lanes. ScottyPath would benefit from using this information in order to plan paths crossing these lanes that do not collide with other ships. In order to make use of this information, ScottyPath could either recompute the safe regions for the areas crossing the shipping lanes, or it could post process the resulting plans in order to ensure that no collisions take place.

Appendix A

Proofs of Completeness and Optimality of PDDL-S Plans with Piecewise Constant Control

In this section, we provide the proofs for the completeness and optimality theorems stated in Section 4.2.1.

A.1 Completeness of PDDL-S Plans with Piecewise Constant Control

Assume the PDDL-S Problem has a valid PDDL-S Plan solution, p, with an arbitrary control trajectory. This solution p is characterized by:

- The schedule of its N start or end events, given by their execution times t_i with $i = 0, \ldots, N-1$.
- The control trajectory $\mathbf{c}(t)$.

We prove the completeness of PDDL-S plans with piecewise constant control by construction. We use the original valid PDDL-S plan p to construct the PDDL-S plan with piecewise constant control p_c in the following way:

- The execution times of the events, t_{ci}, are the same as the times of the original solution, $t_{ci} = t_i$. Since they are the same, we use refer to them as t_i from this point.

- The control trajectory, $\mathbf{c_c}(t)$, is a piecewise constant trajectory. Recall that a *stage*, s_i, is the period of time between two consecutive events taking place at t_i and t_{i+1}. The value of the vector of control variables is constant during each *stage*, $\mathbf{c_c}(t) = \mathbf{c_c}_i \in \mathbb{R}^m, i \in [t_i, t_{i+1})$, for $i = 0, \ldots, N-2$. Each constant value $\mathbf{c_c}_i$ at each stage takes the value of the average of the original control trajectory $\mathbf{c}(t)$ during that stage:

$$\mathbf{c_c}_i = \frac{1}{\Delta t_i} \int_{t_i}^{t_{i+1}} \mathbf{c}(t) dt, \tag{A.1}$$

  where $\Delta t_i = t_{i+1} - t_i$ is the duration of the stage.

Lemma A.1. *The piecewise constant control trajectory of p_c, $\mathbf{c_c}(t)$ satisfies all the control variable constraints at all times.*

Proof. Recall that each control variable constraint is restricted to be a convex constraint on the control variable vector in the form of $g(\mathbf{c}(t)) \leq 0$. Using (A.1), we can write:

$$g(\mathbf{c_c}_i) = g\left(\frac{1}{\Delta t_i} \int_{t_i}^{t_{i+1}} \mathbf{c}(t) dt\right) \tag{A.2}$$

We make use of Jensen's Inequality to assist our proof [51]. Jensen's Inequality states the following:

$$g\left(\int_a^b h(t) f(t) dt\right) \leq \int_a^b g\left(h(t)\right) f(t) dt \tag{A.3}$$

, where:

- f is a non-negative function such that $\int_a^b f(t)dt = 1$.

- h is any real-valued measurable function.

- g is convex over the range $[a, b]$.

We use the vector version of (A.3) on (A.2) by identifying terms as follows: $a = t_i$, $b = t_{i+1}$, $g = g$, $f(t) = \frac{1}{\Delta t_i}$, $\mathbf{h}(t) = \mathbf{c}(t)$. Note that Jensen's conditions hold since g is convex. This allows us to write:

$$g(\mathbf{c}_{\mathbf{c}i}) = g\left(\frac{1}{\Delta t_i}\int_{t_i}^{t_{i+1}} \mathbf{c}(t)dt\right) \leq \int_{t_i}^{t_{i+1}} g(\mathbf{c}(t))\frac{1}{\Delta t_i}dt \tag{A.4}$$

Since p is a valid PDDL-S plan, $g(\mathbf{c}(t)) \leq 0$ at every stage and, therefore, (A.4) reduces to $g(\mathbf{c}_{\mathbf{c}i}) \leq 0$ during stage s_i. Since this is applicable for every stage and every control variable constraint, we conclude that the piecewise constant control trajectory of p_c, $\mathbf{c}_{\mathbf{c}}(t)$, satisfies the control variable constraints. □

Lemma A.2. *The state variables that are not resources take the same values in the PDDL-S plan with piecewise constant control, p_c, as in the original plan, p.*

Proof. Recall that state variables that are not resources can only change their value due to active CLTE effects. Recall, as well, that the change in one such state variable x_j during a stage s_i is given by:

$$\Delta x_j(j) = \sum \Delta x_{j_{eff}}, \tag{A.5}$$

where *eff* represents each of the active CLTE effects during stage s_i. Recall as well from Section 4.1.4.1 that the change due to a CLTE on a state variable x during stage s_i is given by:

$$\Delta x_{CLTE}(j) = \int_{t_i}^{t_{i+1}} \mathbf{k}^T \cdot \mathbf{c}_{\mathbf{c}}(\tau)d\tau \tag{A.6}$$

In the case of the plan with piecewise constant control, $\mathbf{c}_{\mathbf{c}}(t)$ takes the constant value $\mathbf{c}_{\mathbf{c}i}$ during stage s_i:

$$\Delta x_{CLTE}(j) = \int_{t_i}^{t_{i+1}} \mathbf{k}^T \cdot \mathbf{c}_{\mathbf{c}i}d\tau = \mathbf{k}^T \cdot \frac{1}{\Delta t_i}\int_{t_i}^{t_{i+1}} \mathbf{c}(t)dt \int_{t_i}^{t_{i+1}} d\tau = \int_{t_i}^{t_{i+1}} \mathbf{k}^T \cdot \mathbf{c}(\tau)d\tau \tag{A.7}$$

(A.7) proves that the cumulative change due to CLTE effects during a stage s_i is the same regardless of whether the original control trajectory $\mathbf{c}(t)$ or its averaged value during the stage, $\mathbf{c_c}_i$, is used. Since the initial variables are the same, and the cumulative change during each stage is the same, the values of the state variables that are not resources are the same ($\mathbf{x}_i = \mathbf{x}(t_i) = \mathbf{x_c}(t_i) = \mathbf{x_c}_i$) at each event in the original plan, p, and the plan with piecewise constant control, p_c. □

Corollary A.1. *The state trajectory of the PDDL-S plan with piecewise constant control, p_c, satisfies all the state constraints on the state variables that are not resources.*

Proof. Since the values are the same at the events, $\mathbf{x_c}_i = \mathbf{x}(t_i)$, and p is valid, p_c satisfies all the state constraints at the events. The overall maintenance conditions between events are also satisfied by $\mathbf{x_c}(t)$. This is the case because the overall conditions during stages between consecutive events are convex by definition and need to be satisfied also at those events. Chapter 5 proves by convexity that this is the case. □

Lemma A.3. *The resource variables take greater or equal values at the events in the PDDL-S plan with piecewise constant control, p_c, than in the original plan p.*

Proof. The change during stages on resources due to CLTE effects is the same as in the case of state variables that are not resources. The remaining change in resources is due to LNE or LSNE effects.

Recall that the change during stage s_i in resource r due to LNE effect is given by:

$$\Delta r_{LNE}(i) = -k \cdot \int_{t_i}^{t_{i+1}} \|\mathbf{c_c}(t)\| dt \tag{A.8}$$

Substituting for the constant value of $\mathbf{c_c}(t)$ during the stage:

$$\Delta r_{LNE}(i) = -k \cdot \left\| \int_{t_i}^{t_{i+1}} \mathbf{c}(t) \frac{1}{\Delta t_i} \right\| \int_{t_i}^{t_{i+1}} dt \tag{A.9}$$

We can use Jensen's inequality (A.3) again by identifying terms: $a = t_i$, $b = t_{i+1}$, $g(\mathbf{x}) = \|\mathbf{x}\|$, $f(t) = \frac{1}{\Delta t_i}$, $\mathbf{h}(t) = \mathbf{c}(t)$. The conditions for Jensen's inequality hold since norms are convex functions. Therefore, we obtain:

$$\left\| \int_{t_i}^{t_{i+1}} \mathbf{c}(t) \frac{1}{\Delta t_i} \right\| \cdot \Delta t_i \leq \int_{t_i}^{t_{i+1}} \|\mathbf{c}(t)\| \frac{1}{\Delta t_i} dt \cdot \Delta t_i \tag{A.10}$$

Note that (A.10) is the factor that multiplies $-k$ for the cumulative decrease of the LNE effect in the original plan with arbitrary control trajectory. Therefore, the decrease due to the LSE effect is smaller in the case of the piecewise constant control trajectory than in the plan with the arbitrary control trajectory.

Similarly, for LSNE effects, the change in a resource during stage s_i is given by:

$$\Delta r_{LSNE}(i) = -k \cdot \int_{t_i}^{t_{i+1}} \|\mathbf{c}_{\mathbf{c}}(t)\|^2 dt \tag{A.11}$$

Substituting for the constant value of $\mathbf{c}_{\mathbf{c}}(t)$ during he stage:

$$\Delta r_{LSNE}(i) = -k \cdot \left\| \int_{t_i}^{t_{i+1}} \mathbf{c}(t) \frac{1}{\Delta t_i} \right\|^2 \int_{t_i}^{t_{i+1}} dt \tag{A.12}$$

We can use Jensen's inequality (A.3) one more time with the correspondence: $a = t_i$, $b = t_{i+1}$, $g(\mathbf{x}) = \|\mathbf{x}\|^2$, $f(t) = \frac{1}{\Delta t_i}$, $\mathbf{h}(t) = \mathbf{c}(t)$. Jensen's conditions hold again since g is a convex function. It is a convex function because it is the composition of a convex non-decreasing function (x^2 is non-decreasing over the positive reals) and a convex function (the norm function). We then obtain:

$$\left\| \int_{t_i}^{t_{i+1}} \mathbf{c}(t) \frac{1}{\Delta t_i} \right\|^2 \cdot \Delta t_i \leq \int_{t_i}^{t_{i+1}} \|\mathbf{c}(t)\|^2 \frac{1}{\Delta t_i} dt \cdot \Delta t_i \tag{A.13}$$

Again, (A.13) is the factor that multiplies $-k$ for the cumulative decrease of the LSNE effect in the original plan with arbitrary control trajectory. Therefore, the decrease due to the LSNE effect is also smaller in the case of the piecewise constant control trajectory.

From Lemma A.2, the change produced on resources due to CLTE effects is the same in p_c as in p. However, the resources in p_c experience a smaller decrease in p_c during stages due to LSE or LSNE effects than in p. As a consequence, resources have always larger or equal values at the events in p_c with piecewise constant control trajectories than in p with its arbitrary control trajectory. $\square$

Corollary A.2. *The resource variables in the PDDL-S plan with piecewise constant control satisfy all the constraints they are subject to.*

Proof. Recall that resources can only be subject to greater than some value constraints. Since p is a valid plan, the resources in p satisfy the constraints they are subject to. Since the resources take values greater or equal at the events in p_c than in p, the resources satisfy all their constraints at the events. Due to convexity and as proved in Chapter 5, the resources also satisfy the constraints at all intermediate points between events. $\square$

Lemma A.4. *p_c is a valid PDDL-S plan.*

Proof. p_c is a valid PDDL-S plan since:

1. All temporal constraints are satisfied since the event execution times of p_c are the same of p, and p satisfies all temporal constraints.

2. All the propositional constraints (discrete conditions and effects of activities) are satisfied since p satisfies all propositional constraints and the ordering of the starts and ends of activities is the same.

3. The piecewise constant control trajectory satisfies all the control variable constraints (Lemma A.1)

4. The state variables satisfy all the state constraints (Corollaries A.1 and A.2).

$\square$

Theorem 1 (Completeness of PDDL-S Plans with Piecewise Constant Control)**.** *If a PDDL-S problem has a solution, there always exists a solution that is a PDDL-S Plan with piecewise constant control.*

Proof. For every valid PDDL-S plan with an arbitrary control trajectory, it is also possible to construct a valid PDDL-S plan with a piecewise constant control trajectory (Lemma A.4). Therefore, there are no solvable PDDL-S problems that cannot be solved with a PDDL-S plan with piecewise constant control. $\square$

A.2 Optimality of PDDL-S Plans with Piecewise Constant Control

Recall that the minimization objective of a PDDL-S problem is given as a linear combination of the following types of terms:

- The total duration of the plan (plan makespan)
- The value of a state variable at the end of the plan. In the case of a resource, its coefficient can only be negative (i.e. resources can only be maximized).
- $\int_0^T \|\mathbf{c}_e(\tau)\| d\tau$, where $\mathbf{c}_e$ is a vector of control variables and T is the plan makespan. Its coefficient can only be nonnegative (i.e. this term can be minimized but not maximized).
- A similar term with the same conditions with the square of the norm, $\int_0^T \|\mathbf{c}_e(\tau)\|^2 d\tau$.

Corollary A.3. *The makespan is the same in p_c than in p.*

Proof. All the events take place at the same time in p_c than in p. Therefore, the last event takes place at the same time in both plans. □

Corollary A.4. *Any state variable that is not a resource takes the same value at the end in p_c than in p.*

Proof. Since the values of the state variables that are not resources take the same value at all events in p_c than in p (Lemma A.2), they also take the same value at the end of the plan. □

Corollary A.5. *Any resource variable takes a larger value at the end of the plan in p_c than in p.*

Proof. Since the values of the resources take larger value at all events in p_c than in p (Lemma A.3), they also take larger values at the end of the plan. □

Lemma A.5. *$\int_0^T \|\mathbf{c}_e(\tau)\| d\tau$ takes a smaller or equal value in p_c than in p.*

Proof. The proof is similar to that of Lemma A.3. We can write the term as a sum of integrals over each stage in the plan:

$$\int_0^T \|\mathbf{c}_e(\tau)\| d\tau = \sum_{i=0}^{N-2} \int_{t_i}^{t_{i+1}} \|\mathbf{c}_e(\tau)\| d\tau \tag{A.14}$$

In a PDDL-S plan with piecewise constant control, $\mathbf{c}_e(t) = \mathbf{c}_{\mathbf{c}e}(t)$ takes a constant value $\mathbf{c}_{\mathbf{c}ei}$ during each stage s_i. Therefore, each stage term can be written as:

$$\int_{t_i}^{t_{i+1}} \|\mathbf{c}_{\mathbf{c}ei}\| d\tau = \left\| \int_{t_i}^{t_{i+1}} \mathbf{c}_e(t) \frac{1}{\Delta t_i} \right\| \cdot \Delta t_i \leq \int_{t_i}^{t_{i+1}} \|\mathbf{c}_e(t)\| \frac{1}{\Delta t_i} dt \cdot \Delta t_i \tag{A.15}$$

, where the equality follows from the definition of $\mathbf{c}_\mathbf{c}$ (A.1), and the inequality is, again, Jensen's inequality (A.3), and it is applied in the same way as in (A.10). We can then write:

$$\int_0^T \|\mathbf{c}_{\mathbf{c}e}(\tau)\| d\tau = \sum_{i=0}^{N-2} \int_{t_i}^{t_{i+1}} \|\mathbf{c}_{\mathbf{c}e}(\tau)\| d\tau \leq \sum_{i=0}^{N-2} \int_{t_i}^{t_{i+1}} \|\mathbf{c}_e(\tau)\| d\tau = \int_0^T \|\mathbf{c}_e(\tau)\| d\tau, \tag{A.16}$$

which proves that the term is smaller for p_c than for p. □

Lemma A.6. *$\int_0^T \|\mathbf{c}_e(\tau)\|^2 d\tau$ takes a smaller or equal value in p_c than in p.*

Proof. The proof is analogous to the proof for Lemma A.5. The difference is that Jensen's inequality is applied as in (A.13). That leads us to

$$\int_0^T \|\mathbf{c}_{\mathbf{c}e}(\tau)\|^2 d\tau = \sum_{i=0}^{N-2} \int_{t_i}^{t_{i+1}} \|\mathbf{c}_{\mathbf{c}e}(\tau)\|^2 d\tau \leq \sum_{i=0}^{N-2} \int_{t_i}^{t_{i+1}} \|\mathbf{c}_e(\tau)\|^2 d\tau = \int_0^T \|\mathbf{c}_e(\tau)\|^2 d\tau, \tag{A.17}$$

, which proves that the term is smaller for p_c than for p. □

Lemma A.7. *A PDDL-S plan with piecewise constant control, p_c, has better objective value than the plan with arbitrary control trajectory that it was formed from, p.*

Proof. The objective of a PDDL-S Plan is a linear combination of the terms presented above. Due to Corollaries A.3 and A.4, all the terms that could be negative or positive take the same value in p_c than in p. Due to Corollary A.5, all the terms that can only be negative are larger in p_c than in p. Finally, due to Lemmas A.5 and A.6, all the terms that can only be positive are smaller in p_c than in p. Therefore, p_c has a smaller or equal objective value than p. $\square$

Theorem 2 (Optimality of PDDL-S Plans with Piecewise Constant Control)**.** *The optimal solution to a PDDL-S problem, if one exists, is a PDDL-S Plan with piecewise constant control.*

Proof. Since there always exists a PDDL-S plan with piecewise constant control for any valid PDDL-S plan for a PDDL-S problem (Theorem 4.1) and this plan always has a better or equal objective value (Lemma A.7), the optimal plans for PDDL-S problems are PDDL-S plan with piecewise constant control. $\square$

Appendix B

An Example Scenario in PDDL-S Syntax

We now proceed to represent our motivation example from Section 3.1 with the syntax we have defined in Chapter 7. We start by defining the state variables of the problem, which are the x, y positions of the ship, (x_s, y_s), the AUV, (x_a, y_a) and the ROV, (x_r, y_r). Furthermore, the battery of the AUV is also a state variable in this problem. State variables are defined using the same syntax from PDDL2.1:

```
(:functions
 (xs) (ys) (xr) (yr)
 (xa) (ya) (ba))
```

There are six control variables in our problem, the x, y velocities of each vehicle. The maximum velocity of each vehicle is upper-bound constrained. As an example, the velocity of the ship (`vel-ship`) is defined and constrained with the following declaration:

```
(:control-variable vx-s
 :bounds (and (>= ?value -2.0) (<= ?value 2.0)))
(:control-variable vy-s
 :bounds (and (>= ?value -2.0) (<= ?value 2.0)))
(:control-variable-vector vel-ship
 :control-variables ((vx-s) (vy-s))
 :max-norm 2)
```

We define the rectangular region `mission-region` to limit the area that any of the vehicles can visit. Other regions, such as the sampling regions are polygons and are defined with their vertices according to the following syntax:

```
(:region mission-region
 :parameters (?x ?y)
 :condition  (and (in-rect (?x ?y) :corner (0 -100) :width 500 :height
    500)))
(:region regionA
 :parameters (?x ?y)
 :condition
 (and (in-poly (?x ?y) :vertices ((125 0) (130 -15) (115 -30) (100 0))))
    )
```

Since the ROV is tethered to the ship, it always needs to remain within a distance that is the length of the tether. Moreover, the AUV and the ROV can only be recovered by the ship when they are close enough. These two conditions are represented with the regions `rov-range` and `recover-range` defined next:

```
(:region rov-range
 :parameters (?x1 ?y1 ?x2 ?y2)
 :condition  (and
               (max-distance ((?x1 ?y1) (?x2 ?y2)) :d 60)))

(:region recover-range
 :parameters (?x1 ?y1 ?x2 ?y2)
 :condition  (and
               (max-distance ((?x1 ?y1) (?x2 ?y2)) :d 2)))
```

In order to enforce the tether constraint, the ROV and the ship are forced to remain in the `rov-range` region while they move. We present the `navigate-ROV` activity next to show how this is expressed:

```
(:durative-action navigate-ROV
 :duration  (and (>= ?duration 0.1) (<= ?duration 200))
 :condition (and
              (at start (rov-still))
              (over all (rov-deployed))
```

```
            (over all (inside (mission-region (xr) (yr))))
            (over all (inside (rov-range (xr) (yr) (xs) (ys)))))
:effect    (and
            (at start (not (rov-still)))
            (at end (rov-still))
            (at start (not (rov-positioned)))
            (at end (rov-positioned))

            (increase (xr) (* (vx-r) #t))
            (increase (yr) (* (vy-r) #t))))
```

As explained in the beginning of this section, the continuous change in the position of the ROV is represented with a CLTE effect that indicates, for example, that the rate of change of the position (x_r) is proportional to the control variable that represents the v_x velocity of the ROV. RNE effects, such as the one that decreases the AUV battery are expressed in a similar way. The continuous effects of the `navigate-AUV` activity are given by:

```
(increase (xa) (* (vx-a) #t))
(increase (ya) (* (vy-a) #t))
(decrease (ba) (* 1 (norm (vel-auv)) #t ))
```

, which indicates that the rate of change of the battery is proportional to the space traveled by the AUV ($\dot{b}_a = -\|\mathbf{v}_a\|$). The `navigate-AUV` also has an invariant condition (**over all** (>= (b_a) 0)) that ensures that the battery constraint is respected.

Finally, the minimization objective in this problem is the sum of the total plan length and the distance traveled by the ship. This objective is defined as:

```
(:metric minimize (+ (* 1 (total-time) )
                     (* 1 (norm (vel-ship)))))
```

The other discrete conditions and effects, as well as the discrete objectives are defined as in PDDL2.1 and are omitted here for the sake of brevity.

Appendix C

Benchmark Domains

In this appendix we provide the PDDL sources of some of the instances of the benchmark domains described in Section 9.2.

C.1 The AUV Domain

The PDDL of instance 3 of the AUV domain is provided next. In this domain the AUV needs to visit regions A, B and C. The other instances (1-20) are analogous. The only difference between the original and the simplified, linearized version is that the later does not include the maximum norm constraint (indicated with the **:control−variable−vector** statement).
The domain file (`auv03-domain.pddl`) is displayed next:

```
(define (domain auv−2D−3)
(:predicates
  (sample−takenA)
  (sample−takenB)
  (sample−takenC)
  (can−move))
(:functions (x) (y))

;; Control Variables
(:control−variable vel−x
 :bounds (and (>= ?value −2.0) (<= ?value 2.0)))
(:control−variable vel−y
 :bounds (and (>= ?value −2.0) (<= ?value 2.0)))
(:control−variable−vector vel−auv
 :control−variables ((vel−x) (vel−y))
 :max−norm 2)

;; Regions
(:region mission−region
  :parameters (?x ?y)
  :condition (and (in−rect (?x ?y) :corner (0 0) :width 100 :height 100)
    ))
(:region regionA
  :parameters (?x ?y)
  :condition (and (in-rect (?x ?y) :corner (80 70) :width 10 :height 10)
    ))
(:region regionB
  :parameters (?x ?y)
  :condition (and (in-rect (?x ?y) :corner (55 40) :width 5 :height 5)))
(:region regionC
  :parameters (?x ?y)
  :condition (and (in-rect (?x ?y) :corner (30 30) :width 10 :height 10)
    ))

;; Activities
(:durative-action glide
 :duration (and (>= ?duration 0.1) (<= ?duration 200))
 :condition (and (at start (can-move))
                 (over all (inside (mission-region (x) (y)))))
 :effect    (and (at start (not (can-move)))
                                 (at end (can-move))
                 (increase (x) (* (vel-x) #t))
                                 (increase (y) (* (vel-y) #t))))

(:durative-action take-sampleA
 :duration (and (>= ?duration 2) (<= ?duration 8))
 :condition (and  (at start (can-move))
                  (over all (inside (regionA (x) (y))))
                  (at end (inside (regionA (x) (y)))))
:effect (and      (at start (not (can-move)))
                  (at end (can-move))
                  (at end (sample-takenA))))

(:durative-action take-sampleB
 :duration (and (>= ?duration 2) (<= ?duration 8))
 :condition (and  (at start (can-move))
                  (over all (inside (regionB (x) (y))))
                  (at end (inside (regionB (x) (y)))))
:effect (and      (at start (not (can-move)))
                  (at end (can-move))
                  (at end (sample-takenB))))

(:durative-action take-sampleC
 :duration (and (>= ?duration 2) (<= ?duration 8))
 :condition (and  (at start (can-move))
                  (over all (inside (regionC (x) (y))))
                  (at end (inside (regionC (x) (y)))))
:effect (and      (at start (not (can-move)))
                  (at end (can-move))
                  (at end (sample-takenC)))))
```

The problem file (`auv03-problem.pddl`) is displayed next:

```
(define (problem auv-3D-problem-3)
  (:domain auv-2D-1)
  (:init
    (can-move)
    (= (x) 0) (= (y) 0))
  (:goal (and
            (sample-takenA)
            (sample-takenB)
            (sample-takenC))))
(:metric minimize (+ (* 1 (total-time))))
```

C.2 The ROV Domain

We provide the PDDL sources for instance 6 of the ROV domain. In this problem, the ROV needs to visit regions A, B, C, D, E and F (Figure 9-3).

C.2.1 Original (quadratic) version

The domain file (`rov06-domain.pddl`) is displayed below:

```
(define (domain rov-6)
(:predicates
  (rov-deployed)(rov-onboard)
  (rov-navigating)(rov-still)
  (rov-positioned)(ship-arrived)
  (mission-ongoing)
  (sample-takenA)(sample-takenB)
        (sample-takenC)(sample-takenD)
        (sample-takenE)(sample-takenF))

(:functions (xs) (ys) (xr) (yr))

;; Control Variables
(:control-variable vx-s
 :bounds (and (>= ?value -2.0) (<= ?value 2.0)))
(:control-variable vy-s
 :bounds (and (>= ?value -2.0) (<= ?value 2.0)))
(:control-variable-vector vel-ship
 :control-variables ((vx-s) (vy-s))
 :max-norm 2)
(:control-variable vx-r
 :bounds (and (>= ?value -2.0) (<= ?value 2.0)))
(:control-variable vy-r
 :bounds (and (>= ?value -2.0) (<= ?value 2.0)))
(:control-variable-vector vel-rov
 :control-variables ((vx-r) (vy-r))
 :max-norm 2)

;; Regions
(:region mission-region
  :parameters (?x ?y)
  :condition (and (in-rect (?x ?y) :corner (0 0) :width 100 :height 100)
    ))
(:region region-port
  :parameters (?x ?y)
  :condition (and (in-poly (?x ?y) :vertices ((80.00000 80.00000)
    (80.00000 90.00000)(90.00000 90.00000)(90.00000 80.00000)(80.00000
    80.00000)))))
```

```
(:region regionA
  :parameters (?x ?y)
  :condition (and (in-poly (?x ?y) :vertices ((39.37217 36.35934)
    (39.62838 41.83741)(33.58334 38.41339)(35.90700 36.75789)(39.37217
    36.35934)))))
(:region regionB
  :parameters (?x ?y)
  :condition (and (in-poly (?x ?y) :vertices ((53.20386 24.86533)
    (59.77362 23.94972)(60.97144 25.64728)(58.46709 27.47164)(53.20386
    24.86533)))))
(:region regionC
  :parameters (?x ?y)
  :condition (and (in-poly (?x ?y) :vertices ((54.84244 42.09887)
    (53.85109 44.74345)(48.76991 42.71553)(51.70078 38.83075)(54.84244
    42.09887)))))
(:region regionD
  :parameters (?x ?y)
  :condition (and (in-poly (?x ?y) :vertices ((14.22096 82.10052)
    (14.54697 77.25059)(17.45469 76.93250)(19.08229 80.64577)(14.22096
    82.10052)))))
(:region regionE
  :parameters (?x ?y)
  :condition (and (in-poly (?x ?y) :vertices ((32.26246 85.87668)
    (34.33392 88.33325)(34.46927 90.12637)(30.73706 91.88235)(32.26246
    85.87668)))))
(:region regionF
  :parameters (?x ?y)
  :condition (and (in-poly (?x ?y) :vertices ((30.13904 62.94699)
    (29.93304 65.07422)(25.68036 65.04391)(24.56301 62.75958)(30.13904
    62.94699)))))

(:region rov-range
 :parameters (?x1 ?y1 ?x2 ?y2)
 :condition (and (max-distance ((?x1 ?y1) (?x2 ?y2)) :d 10)))
(:region recover-range
  :parameters (?x1 ?y1 ?x2 ?y2)
  :condition (and (max-distance ((?x1 ?y1) (?x2 ?y2)) :d 0.5)))

;; Activities
(:durative-action navigate-ship
 :duration (and (>= ?duration 0.1) (<= ?duration 200))
 :condition (and (over all (mission-ongoing))
                 (over all (rov-onboard))
                 (over all (inside (mission-region (xs) (ys))))
                 (over all (inside (mission-region (xr) (yr)))))
 :effect (and
          (increase (xs) (* (vx-s) #t))
          (increase (ys) (* (vy-s) #t))
          ;; Also move ROV state variables
          (increase (xr) (* (vx-s) #t))
          (increase (yr) (* (vy-s) #t))))

(:durative-action navigate-ROV
 :duration (and (>= ?duration 0.1) (<= ?duration 200))
 :condition (and (over all (mission-ongoing))
                 (at start (rov-still))
                 (over all (rov-deployed))
                 (over all (inside (mission-region (xr) (yr))))
                 (over all (inside (rov-range (xr) (yr) (xs) (ys)))))
 :effect    (and (at start (not (rov-still)))
                 (at end (rov-still))
                 (at start (not (rov-positioned)))
                 (at end (rov-positioned))
                 (increase (xr) (* (vx-r) #t))
                 (increase (yr) (* (vy-r) #t))))

(:durative-action deploy-ROV
 :duration  (and (>= ?duration 10) (<= ?duration 10))
 :condition (and (over all (mission-ongoing))
                 (at start (rov-onboard)))
 :effect (and (at start (not (rov-onboard)))
              (at end (rov-still))
              (at end (rov-deployed))))

(:durative-action recover-ROV
 :duration  (and (>= ?duration 40) (<= ?duration 40))
 :condition (and (over all (mission-ongoing))
                 (at start (rov-deployed))
                 (over all (rov-still))
                 (over all (rov-positioned))
                 (over all (inside (recover-range (xr) (yr) (xs) (ys))))
   )
 :effect (and (at start (not (rov-deployed)))
              (at end (rov-onboard))
              (at start (not (rov-positioned)))))

(:durative-action arrive-port
:duration (and (>= ?duration 2) (<= ?duration 2))
:condition (and (over all (rov-onboard))
                (over all (inside (region-port (xs) (ys)))))
:effect (and (at end (ship-arrived))
             (at start (not (mission-ongoing)))))


(:durative-action take-sampleA
:duration  (and (>= ?duration 20) (<= ?duration 20))
:condition (and (over all (mission-ongoing))
                (over all (rov-still))
                (over all (rov-positioned))
                (over all (rov-deployed))
                (over all (inside (regionA (xr) (yr))))
                (at end (inside (regionA (xr) (yr)))))
:effect (and (at end (sample-takenA))
             (at end (not (rov-positioned)))))

(:durative-action take-sampleB
:duration  (and (>= ?duration 20) (<= ?duration 20))
:condition (and (over all (mission-ongoing))
                (over all (rov-still))
                (over all (rov-positioned))
                (over all (rov-deployed))
                (over all (inside (regionB (xr) (yr))))
                (at end (inside (regionB (xr) (yr)))))
:effect (and (at end (sample-takenB))
             (at end (not (rov-positioned)))))

(:durative-action take-sampleC
:duration  (and (>= ?duration 20) (<= ?duration 20))
:condition (and (over all (mission-ongoing))
                (over all (rov-still))
                (over all (rov-positioned))
                (over all (rov-deployed))
                (over all (inside (regionC (xr) (yr))))
                (at end (inside (regionC (xr) (yr)))))
:effect (and (at end (sample-takenC))
             (at end (not (rov-positioned)))))

(:durative-action take-sampleD
:duration  (and (>= ?duration 20) (<= ?duration 20))
:condition (and (over all (mission-ongoing))
                (over all (rov-still))
                (over all (rov-positioned))
                (over all (rov-deployed))
                (over all (inside (regionD (xr) (yr))))
                (at end (inside (regionD (xr) (yr)))))
:effect (and (at end (sample-takenD))
             (at end (not (rov-positioned)))))

(:durative-action take-sampleE
:duration  (and (>= ?duration 20) (<= ?duration 20))
:condition (and (over all (mission-ongoing))
                (over all (rov-still))
                (over all (rov-positioned))
                (over all (rov-deployed))
                (over all (inside (regionE (xr) (yr))))
                (at end (inside (regionE (xr) (yr)))))
:effect (and (at end (sample-takenE))
             (at end (not (rov-positioned)))))

(:durative-action take-sampleF
:duration  (and (>= ?duration 20) (<= ?duration 20))
:condition (and (over all (mission-ongoing))
                (over all (rov-still))
                (over all (rov-positioned))
                (over all (rov-deployed))
                (over all (inside (regionF (xr) (yr))))
                (at end (inside (regionF (xr) (yr)))))
:effect (and (at end (sample-takenF))
             (at end (not (rov-positioned))))))
```

The ROV instance 6 problem file:

```
(define (problem rov-problem-6)
(:domain rov-6)
(:init (rov-onboard)
       (rov-still)
       (mission-ongoing)
       (= (xs) 20.0)(= (ys) 30.0)
       (= (xr) 20.0)(= (yr) 30.0))
(:goal (and (sample-takenA)(sample-takenB)
            (sample-takenC)(sample-takenD)
            (sample-takenE)(sample-takenF)
            (rov-onboard)(ship-arrived))))
(:metric minimize (+ (* 0.1 (total-time) )
                     (* 2.5 (norm-sq (vel-ship)))))
```

C.2.2 Simplified, linearized version

In the linearized version the velocities of the AUV and the ship are not norm constrained and therefore, the **:control−variable−vector** statements do not appear in the domain file. Moreover, the original convex quadratic constraints that represent the maximum distance between the ROV and the ship (represented with the `rov-range` and `recover-range` regions) are replaced by linear approximations (in particular, octagon over-approximations). Therefore the `rov-range` and `recover-range` regions are described in the following way:

```
(:region rov-range
 :parameters (?x1 ?y1 ?x2 ?y2)
 :condition (and
             (<= (+ (* 1.0 (- ?x1 ?x2)) (* 2.414 (- ?y1 ?y2))) 24.142)
             (<= (+ (* -1.0 (- ?x1 ?x2)) (* 2.414 (- ?y1 ?y2))) 24.142)
             (<= (+ (* -2.414 (- ?x1 ?x2)) (* -1.0 (- ?y1 ?y2))) 24.142)
             (<= (+ (* -2.414 (- ?x1 ?x2)) (* 1.0 (- ?y1 ?y2))) 24.142)
             (<= (+ (* -1.0 (- ?x1 ?x2)) (* -2.414 (- ?y1 ?y2))) 24.142)
             (<= (+ (* 1.0 (- ?x1 ?x2)) (* -2.414 (- ?y1 ?y2))) 24.142)
             (<= (+ (* 2.414 (- ?x1 ?x2)) (* -1.0 (- ?y1 ?y2))) 24.142)
             (<= (+ (* 2.414 (- ?x1 ?x2)) (* 1.0 (- ?y1 ?y2))) 24.142)))

(:region recover-range
 :parameters (?x1 ?y1 ?x2 ?y2)
 :condition (and
             (<= (+ (* 1.0 (- ?x1 ?x2)) (* 2.414 (- ?y1 ?y2))) 1.207)
             (<= (+ (* -1.0 (- ?x1 ?x2)) (* 2.414 (- ?y1 ?y2))) 1.207)
             (<= (+ (* -2.414 (- ?x1 ?x2)) (* -1.0 (- ?y1 ?y2))) 1.207)
             (<= (+ (* -2.414 (- ?x1 ?x2)) (* 1.0 (- ?y1 ?y2))) 1.207)
             (<= (+ (* -1.0 (- ?x1 ?x2)) (* -2.414 (- ?y1 ?y2))) 1.207)
             (<= (+ (* 1.0 (- ?x1 ?x2)) (* -2.414 (- ?y1 ?y2))) 1.207)
             (<= (+ (* 2.414 (- ?x1 ?x2)) (* -1.0 (- ?y1 ?y2))) 1.207)
             (<= (+ (* 2.414 (- ?x1 ?x2)) (* 1.0 (- ?y1 ?y2))) 1.207)))
```

C.3 The Air Refueling Domain

As an example, we provide the PDDL sources for instance 15 of the Air Refueling domain. In this problem there are two UAVs that need to take images in six regions, while refueling from a single tanker airplane. For the reasons explained in Section 9.2.3, no linearized version is provided for this domain.

Domain (`onair15-domain.pddl`):

```
(define (domain onair-refuel-15)
(:predicates
  (can-start)
  (uav-canfly)(uav-flying)(uav-available)
  (uav2-canfly)(uav2-flying)(uav2-available)
  (tanker-flying)
  (mission-ongoing)
  (arrived)
  (photo-takenA)(photo-takenB)
  (photo-takenC)(photo-takenD)(photo-takenE))
(:functions (xt) (yt) (xb) (yb) (bb) (xb2) (yb2) (bb2))

;; Control Variables
(:control-variable vx-t
 :bounds (and (>= ?value -2.0) (<= ?value 2.0)))
(:control-variable vy-t
 :bounds (and (>= ?value -2.0) (<= ?value 2.0)))
(:control-variable-vector vel-tanker
 :control-variables ((vx-t) (vy-t))
 :max-norm 2)
(:control-variable vx-b
 :bounds (and (>= ?value -3.0) (<= ?value 3.0)))
(:control-variable vy-b
 :bounds (and (>= ?value -3.0) (<= ?value 3.0)))
(:control-variable-vector vel-uav
 :control-variables ((vx-b) (vy-b))
 :max-norm 3)
(:control-variable vx-b2
 :bounds (and (>= ?value -3.0) (<= ?value 3.0)))
(:control-variable vy-b2
 :bounds (and (>= ?value -3.0) (<= ?value 3.0)))
(:control-variable-vector vel-uav2
 :control-variables ((vx-b2) (vy-b2))
 :max-norm 3)
(:control-variable vx-b-ref
 :bounds (and (>= ?value -0.5) (<= ?value 0.5)))
(:control-variable vy-b-ref
 :bounds (and (>= ?value -0.5) (<= ?value 0.5)))
(:control-variable-vector vel-uav-ref
 :control-variables ((vx-b-ref) (vy-b-ref))
 :max-norm 0.5)
(:control-variable bat-recharge-rt
 :bounds (and (>= ?value 0.5) (<= ?value 10)))

;; Regions
```

```
(:region mission-region
 :parameters (?x ?y)
 :condition (and (in-rect (?x ?y) :corner (0 0) :width 100 :height 100))
   )
(:region end-region
 :parameters (?x ?y)
 :condition (and (in-poly (?x ?y) :vertices ((30.00000 80.00000)
   (30.00000 90.00000)(40.00000 90.00000)(40.00000 80.00000)(30.00000
   80.00000)))))
(:region regionA
 :parameters (?x ?y)
 :condition (and (in-poly (?x ?y) :vertices ((69.28348 48.10923)
   (68.00933 45.38239)(73.61835 42.22267)(74.51618 48.55133)(69.28348
   48.10923)))))
(:region regionB
 :parameters (?x ?y)
 :condition (and (in-poly (?x ?y) :vertices ((8.00984 57.59487)(7.01760
   51.92697)(9.45458 50.25484)(14.20403 53.92992)(8.00984 57.59487)))))
(:region regionC
 :parameters (?x ?y)
 :condition (and (in-poly (?x ?y) :vertices ((23.52966 20.52394)
   (28.28920 22.87291)(25.77673 27.59659)(22.34778 24.69332)(23.52966
   20.52394)))))
(:region regionD
 :parameters (?x ?y)
 :condition (and (in-poly (?x ?y) :vertices ((49.99606 18.74888)
   (54.37759 24.80803)(52.85137 25.66706)(49.60897 24.57518)(49.99606
   18.74888)))))
(:region regionE
 :parameters (?x ?y)
 :condition (and (in-poly (?x ?y) :vertices ((59.35168 78.26495)
   (57.61885 83.77747)(52.45846 80.30299)(56.94561 76.10759)(59.35168
   78.26495)))))

(:region refuel-range
 :parameters (?x1 ?y1 ?x2 ?y2)
 :condition (and (max-distance ((?x1 ?y1) (?x2 ?y2)) :d 2)))

;; Activities
(:durative-action fly-tanker
 :duration (and (>= ?duration 0.1) (<= ?duration 2000))
 :condition (and (at start (can-start))
             (over all (inside (mission-region (xt) (yt)))))
:effect (and (at start (not (can-start)))
              (at start (mission-ongoing))
              (at end (not (mission-ongoing)))
              (at start (tanker-flying))
              (at end (not (tanker-flying)))
              (increase (xt) (* (vx-t) #t))
              (increase (yt) (* (vy-t) #t))))

(:durative-action fly-uav
:duration (and (>= ?duration 0.1) (<= ?duration 2000))
:condition (and (at start (mission-ongoing))
                (at start (uav-canfly))
                (over all (inside (mission-region (xb) (yb))))
                (over all (>= (bb) 0)))
:effect (and (at start (not (uav-canfly)))
             (at start (uav-flying))
             (at end (not (uav-flying)))
             (increase (xb) (* (vx-b) #t))
             (increase (yb) (* (vy-b) #t))
             (decrease (bb) (* 0.1 (norm-sq (vel-uav)) #t))
             (decrease (bb) (* 1.1 (norm (vel-uav)) #t))))

(:durative-action fly-uav2
:duration (and (>= ?duration 0.1) (<= ?duration 2000))
:condition (and (at start (mission-ongoing))
                (at start (uav2-canfly))
                (over all (inside (mission-region (xb2) (yb2))))
                (over all (>= (bb2) 0)))
:effect (and (at start (not (uav2-canfly)))
             (at start (uav2-flying))
             (at end (not (uav2-flying)))
             (increase (xb2) (* (vx-b2) #t))
             (increase (yb2) (* (vy-b2) #t))
             (decrease (bb2) (* 0.1 (norm-sq (vel-uav2)) #t))
             (decrease (bb2) (* 1.1 (norm (vel-uav2)) #t))))

(:durative-action refuel-uav
:duration (and (>= ?duration 0.5) (<= ?duration 20))
:condition (and (over all (tanker-flying))
                (over all (uav-flying))
                (over all (<= (bb) 100))
                (at start (uav-available))
                (over all (inside (refuel-range (xt) (yt) (xb) (yb)))))
:effect (and (at start (not (uav-available)))
             (at end (uav-available))
             (increase (bb) (* (bat-recharge-rt) #t))))

(:durative-action refuel-uav2
:duration (and (>= ?duration 0.5) (<= ?duration 20))
:condition (and
            (over all (tanker-flying))
            (over all (uav-flying))
            (over all (<= (bb2) 100))
            (at start (uav2-available))
            (over all (inside (refuel-range (xt) (yt) (xb2) (yb2)))))
:effect (and
          (at start (not (uav2-available)))
          (at end (uav2-available))
          (increase (bb2) (* (bat-recharge-rt) #t))))

(:durative-action arrive-airport
 :duration (and (>= ?duration 2) (<= ?duration 2))
 :condition (and (at start (mission-ongoing))
                 (at start (uav-flying))
                 (at start (uav2-flying))
                 (at start (uav-available))
                 (at start (uav2-available))
                 (over all (inside (end-region (xt) (yt))))
                 (over all (inside (end-region (xb) (yb))))
                 (over all (inside (end-region (xb2) (yb2)))))
:effect (and (at end (arrived))
             (at start (not (mission-ongoing)))))

(:durative-action take-photoA
:duration (and (>= ?duration 15) (<= ?duration 15))
:condition (and (over all (mission-ongoing))
                (over all (uav-flying))
                (over all (inside (regionA (xb) (yb))))
                (at end (inside (regionA (xb) (yb))))
                (at start (uav-available)))
:effect (and (at start (not (uav-available)))
             (at end (uav-available))
             (at end (photo-takenA))))

(:durative-action take-photoA2
:duration (and (>= ?duration 15) (<= ?duration 15))
:condition (and (over all (mission-ongoing))
                (over all (uav2-flying))
                (over all (inside (regionA (xb2) (yb2))))
                (at end (inside (regionA (xb2) (yb2))))
                (at start (uav2-available)))
:effect (and (at start (not (uav2-available)))
             (at end (uav2-available))
             (at end (photo-takenA))))

(:durative-action take-photoB
:duration (and (>= ?duration 15) (<= ?duration 15))
:condition (and (over all (mission-ongoing))
                (over all (uav-flying))
                (over all (inside (regionB (xb) (yb))))
                (at end (inside (regionB (xb) (yb))))
                (at start (uav-available)))
:effect (and (at start (not (uav-available)))
             (at end (uav-available))
             (at end (photo-takenB))))

(:durative-action take-photoB2
:duration (and (>= ?duration 15) (<= ?duration 15))
:condition (and (over all (mission-ongoing))
                (over all (uav2-flying))
                (over all (inside (regionB (xb2) (yb2))))
                (at end (inside (regionB (xb2) (yb2))))
                (at start (uav2-available)))
:effect (and (at start (not (uav2-available)))
             (at end (uav2-available))
             (at end (photo-takenB))))

(:durative-action take-photoC
:duration (and (>= ?duration 15) (<= ?duration 15))
```

```
:condition (and (over all (mission-ongoing))
                (over all (uav-flying))
                (over all (inside (regionC (xb) (yb))))
                (at end (inside (regionC (xb) (yb))))
                (at start (uav-available)))
:effect (and (at start (not (uav-available)))
             (at end (uav-available))
             (at end (photo-takenC))))

(:durative-action take-photoC2
:duration (and (>= ?duration 15) (<= ?duration 15))
:condition (and (over all (mission-ongoing))
                (over all (uav2-flying))
                (over all (inside (regionC (xb2) (yb2))))
                (at end (inside (regionC (xb2) (yb2))))
                (at start (uav2-available)))
:effect (and (at start (not (uav2-available)))
             (at end (uav2-available))
             (at end (photo-takenC))))

(:durative-action take-photoD
:duration (and (>= ?duration 15) (<= ?duration 15))
:condition (and (over all (mission-ongoing))
                (over all (uav-flying))
                (over all (inside (regionD (xb) (yb))))
                (at end (inside (regionD (xb) (yb))))
                (at start (uav-available)))
:effect (and (at start (not (uav-available)))
             (at end (uav-available))
             (at end (photo-takenD))))

(:durative-action take-photoD2
:duration (and (>= ?duration 15) (<= ?duration 15))
:condition (and (over all (mission-ongoing))
                (over all (uav2-flying))
                (over all (inside (regionD (xb2) (yb2))))
                (at end (inside (regionD (xb2) (yb2))))
                (at start (uav2-available)))
:effect (and (at start (not (uav2-available)))
             (at end (uav2-available))
             (at end (photo-takenD))))

(:durative-action take-photoE
:duration (and (>= ?duration 15) (<= ?duration 15))
:condition (and (over all (mission-ongoing))
                (over all (uav-flying))
                (over all (inside (regionE (xb) (yb))))
                (at end (inside (regionE (xb) (yb))))
                (at start (uav-available)))
:effect (and (at start (not (uav-available)))
             (at end (uav-available))
             (at end (photo-takenE))))

(:durative-action take-photoE2
:duration (and (>= ?duration 15) (<= ?duration 15))
:condition (and (over all (mission-ongoing))
                (over all (uav2-flying))
                (over all (inside (regionE (xb2) (yb2))))
                (at end (inside (regionE (xb2) (yb2))))
                (at start (uav2-available)))
:effect (and (at start (not (uav2-available)))
             (at end (uav2-available))
             (at end (photo-takenE)))))
```

Problem (`onair15-problem.pddl`):

```
(define (problem onair-problem-1)
 (:domain onair-refuel-1)
 (:init (can-start)
        (uav-canfly)(uav-available)
        (uav2-canfly)(uav2-available)
        (= (xt) 70.0)(= (yt) 10.0)
        (= (xb) 70.0)(= (yb) 10.0)
        (= (xb2) 70.0)(= (yb2) 10.0)
        (= (bb) 100)(= (bb2) 100))
 (:goal (and
         (photo-takenA)(photo-takenB)
         (photo-takenC)(photo-takenD)
         (photo-takenE)
          (arrived))))
(:metric minimize (+ (* 5 (total-time) )
                     (* 20 (norm (vel-tanker)))))
```

Bibliography

[1] Johannes Aldinger and Johannes Löhr. The Jumpbot Domain for Numeric Planning. Technical report, University of Freiburg, April 2016.

[2] Johannes Aldinger and Bernhard Nebel. Interval Based Relaxation Heuristics for Numeric Planning with Action Costs. In *KI 2017: Advances in Artificial Intelligence*, pages 15–28. Springer, Cham, Cham, September 2017.

[3] C G Atkeson, B P W Babu, N Banerjee, D Berenson, C P Bove, X Cui, M DeDonato, R Du, S Feng, P Franklin, M Gennert, J P Graff, P He, A Jaeger, J Kim, K Knoedler, L Li, C Liu, X Long, T Padir, F Polido, G G Tighe, and X Xinjilefu. No falls, no resets: Reliable humanoid behavior in the DARPA robotics challenge. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 623–630. IEEE, November 2015.

[4] Josef Bajada, Maria Fox, and Derek Long. Temporal Planning with Semantic Attachment of Non-Linear Monotonic Continuous Behaviours. In *IJCAI 2015, Proceedings of the 24th International Joint Conference on Artificial Intelligence, Buenos Aires, Argentina, July 25-31, 2015*, 2015.

[5] T Bedrax-Weiss, J Frank, A Jónsson, and C McGann. EUROPA2: Plan database services for planning and scheduling applications. 2004.

[6] J Benton, Amanda Jane Coles, and Andrew Coles. Temporal Planning with Preferences and Time-Dependent Continuous Costs. *ICAPS 2012*, 2012.

[7] Sara Bernardini, Maria Fox, and Derek Long. Planning the Behaviour of Low-Cost Quadcopters for Surveillance Missions. *Twenty-Fourth International Conference on Automated Planning and Scheduling*, November 2014.

[8] Sara Bernardini, Maria Fox, Derek Long, and Chiara Piacentini. Boosting Search Guidance in Problems with Semantic Attachments. In *Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, ICAPS 2017, Pittsburgh, Pennsylvania, USA, June 18-23, 2017.*, pages 29–37, 2017.

[9] Brian Bingham, Brendan Foley, Hanumant Singh, Richard Camilli, Katerina Delaporta, Ryan Eustice, Angelos Mallios, David Mindell, Christopher Roman, and Dimitris Sakellariou. Robotic tools for deep water archaeology: Surveying

an ancient shipwreck with an autonomous underwater vehicle. *Journal of Field Robotics*, 27(6):702–717, 2010.

[10] Lars Blackmore, Masahiro Ono, and Brian C Williams. Chance-Constrained Optimal Path Planning With Obstacles. *IEEE Transactions on Robotics*, 27(6):1080–1094, December 2011.

[11] Avrim L Blum and Merrick L Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1):281–300, 1997.

[12] Sergiy Bogomolov, Daniele Magazzeni, Stefano Minopoli, and Martin Wehrle. PDDL+ Planning with Hybrid Automata: Foundations of Translating Must Behavior. In *Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, ICAPS 2015, Jerusalem, Israel, June 7-11, 2015.*, pages 42–46, 2015.

[13] Sergiy Bogomolov, Daniele Magazzeni, Andreas Podelski, and Martin Wehrle. Planning as Model Checking in Hybrid Domains. In *AAAI-17*, pages 2228–2234, 2014.

[14] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[15] Daniel Bryce, Sicun Gao, David J Musliner, and Robert P Goldman. SMT-Based Nonlinear PDDL+ Planning. *AAAI-17*, pages 3247–3253, 2015.

[16] S Cambon, R Alami, and F Gravot. A Hybrid Approach to Intricate Motion, Manipulation and Task Planning. *The International Journal of Robotics Research*, 28(1):104–126, January 2009.

[17] Richard Camilli, Paraskevi Nomikou, Javier Escartin, Pere Ridao, Angelos Mallios, Stephanos P Kilias, Ariadne Argyraki, Muriel Andreani, Valerie Ballu, Ricard Campos, Christine Deplus, Taoufic Gabsi, Rafael Garcia, Nuno Gracias, Natalia Hurtos, Lluis Magi, Catherine Mevel, Manuel Moreira, Narcis Palomeras, Olivier Pot, David Ribas, Lorraine Ruzie, and Dimitris Sakellariou. The Kallisti Limnes, carbon dioxide-accumulating subsea pools. *Scientific Reports*, 5(1):srep12152, July 2015.

[18] Richard Camilli, Christopher M Reddy, Dana R Yoerger, Benjamin A S Van Mooy, Michael V Jakuba, James C Kinsey, Cameron P McIntyre, Sean P Sylva, and James V Maloney. Tracking hydrocarbon plume transport and biodegradation at Deepwater Horizon. *Science*, 330(6001):201–204, October 2010.

[19] M Cashmore, M Fox, T Larkworthy, D Long, and D Magazzeni. AUV mission control via temporal planning. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 6535–6541. IEEE, 2014.

[20] Michael Cashmore, Maria Fox, Derek Long, and Daniele Magazzeni. A Compilation of the Full PDDL+ Language into SMT. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016.*, pages 79–87, 2016.

[21] Amanda Jane Coles, Andrew Coles, Maria Fox, and Derek Long. Temporal Planning in Domains with Linear Processes. In *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pages 1671–1676, 2009.

[22] Amanda Jane Coles, Andrew Coles, Maria Fox, and Derek Long. Forward-Chaining Partial-Order Planning. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada, May 12-16, 2010*, pages 42–49, 2010.

[23] Amanda Jane Coles, Andrew Coles, Maria Fox, and Derek Long. COLIN: Planning with continuous linear numeric change. *Journal of Artificial Intelligence Research (JAIR)*, 44:1–96, 2012.

[24] Andrew Coles, Maria Fox, Derek Long, and Amanda Smith. Planning with Problems Requiring Temporal Coordination. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pages 892–897, 2008.

[25] William Cushing, Subbarao Kambhampati, Mausam, and Daniel S Weld. When is Temporal Planning Really Temporal? In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, pages 1852–1859, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[26] Hongkai Dai, A Valenzuela, and Russ Tedrake. Whole-body motion planning with centroidal dynamics and full kinematics. In *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*, pages 295–302. IEEE, 2014.

[27] R Dechter, I Meiri, and J Pearl. Temporal constraint networks. *Artificial Intelligence*, 1991.

[28] Robin Deits and Russ Tedrake. Efficient Mixed-Integer Planning for UAVs in Cluttered Environments. *groups.csail.mit.edu*, December 2014.

[29] Robin Deits and Russ Tedrake. Footstep Planning on Uneven Terrain with Mixed-Integer Convex Optimization. In *Proceedings of the 2014 IEEE/RAS International Conference on Humanoid Robots (Humanoids 2014), Madrid, Spain, 2014*, August 2014.

[30] Robin Deits and Russ Tedrake. Computing large convex regions of obstacle-free space through semidefinite programming. *Algorithmic Foundations of Robotics XI*, 2015.

[31] Giuseppe Della Penna, Daniele Magazzeni, Fabio Mercorio, and Benedetto Intrigila. UPMurphi: A Tool for Universal Planning on PDDL+ Problems. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*, 2009.

[32] Minh Binh Do and Subbarao Kambhampati. Sapa: A Multi-objective Metric Temporal Planner. *J Artif Intell Res(JAIR)*, 20:155–194, 2003.

[33] Christian Dornhege, Patrick Eyerich, Thomas Keller, Sebastian Tr u g, Michael Brenner, and Bernhard Nebel. Semantic Attachments for Domain-independent Planning Systems. In *Proceedings of the Nineteenth International Conference on International Conference on Automated Planning and Scheduling*, pages 114–121. AAAI Press, 2009.

[34] Patrick Eyerich, Robert Matmüller, and Gabriele Röger. Using the Context-enhanced Additive Heuristic for Temporal and Numeric Planning. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling, ICAPS 2009, Thessaloniki, Greece, September 19-23, 2009*, 2009.

[35] M Fallon, S Kuindersma, S Karumanchi, and Russ Tedrake. An Architecture for Online Affordance-based Perception and Whole-body Planning. *Journal of Field ...*, 2015.

[36] Enrique Fernandez-Gonzalez, Erez Karpas, and Brian C Williams. Mixed Discrete-Continuous Heuristic Generative Planning Based on Flow Tubes. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 1565–1572, 2015.

[37] Enrique Fernandez-Gonzalez, Erez Karpas, and Brian C Williams. Mixed Discrete-Continuous Heuristic Generative Planning based on Flow Tubes (extended version). In *PlanRob Workshop, ICAPS 2015*, pages 1–10, May 2015.

[38] Enrique Fernandez-Gonzalez, Erez Karpas, and Brian C Williams. Mixed Discrete-Continuous Planning with Convex Optimization. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 1–7, 2017.

[39] M Fox and D Long. PDDL2. 1: An Extension to PDDL for Expressing Temporal Planning Domains. *J Artif Intell Res(JAIR)*, 2003.

[40] M Fox and D Long. Modelling Mixed Discrete-Continuous Domains for Planning. *J Artif Intell Res(JAIR)*, 2006.

[41] Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. FFRob: An efficient heuristic for task and motion planning. *lis.csail.mit.edu*, 2014.

[42] Alfonso E Gerevini, Patrik Haslum, Derek Long, Alessandro Saetti, and Yannis Dimopoulos. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*, 173(5-6):619–668, April 2009.

[43] Sean Gillies and others. Shapely: manipulation and analysis of geometric objects. Technical report, toblerity.org, 2007.

[44] Dylan Hadfield-Menell, Christopher Lin, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. Sequential quadratic programming for task plan optimization. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS*, pages 5040–5047. IEEE, 2016.

[45] Jörg Hoffmann. The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. *J Artif Intell Res(JAIR)*, 20:291–341, 2003.

[46] Jörg Hoffmann and Stefan Edelkamp. The deterministic part of IPC-4: An overview. *Journal of Artificial Intelligence Research*, 24:519–579, 2005.

[47] Jörg Hoffmann and Bernhard Nebel. The FF Planning System: Fast Plan Generation Through Heuristic Search. *J Artif Intell Res(JAIR)*, 14:253–302, 2001.

[48] Andreas Hofmann. *Robust Execution of Bipedal Walking Tasks From Biomechanical Principles.* PhD thesis, Massachusetts Institute of Technology, January 2006.

[49] Andreas Hofmann and Brian C Williams. Exploiting spatial and temporal flexibility for plan execution of hybrid, under-actuated systems. *AAAI 2006*, 2006.

[50] Andreas Hofmann and Brian C Williams. Temporally and spatially flexible plan execution for dynamic hybrid systems. *Artificial Intelligence*, (0 SP - EP - PY - T2 -):–, 2015.

[51] J L W V Jensen. Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta Mathematica*, 30(0):175–193, 1906.

[52] Sertac Karaman and Emilio Frazzoli. Incremental Sampling-based Algorithms for Optimal Motion Planning. *arXiv.org*, May 2010.

[53] Henry A Kautz and Bart Selman. Planning as Satisfiability. In *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI'92)*, pages 359–363, 1992.

[54] Henry A Kautz and Bart Selman. Unifying SAT-based and Graph-based Planning. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*, pages 318–325, 1999.

[55] Eric Krotkov, Douglas Hackett, Larry Jackel, Michael Perschbacher, James Pippine, Jesse Strauss, Gill Pratt, and Christopher Orlowski. The DARPA Robotics Challenge Finals: Results and Perspectives. *Journal of Field Robotics*, 34(2):229–240, 2017.

[56] Scott Kuindersma, Robin Deits, Maurice Fallon, Andrés Valenzuela, Hongkai Dai, Frank Permenter, Twan Koolen, Pat Marion, and Russ Tedrake. Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot. *Autonomous Robots*, 40(3):429–455, 2015.

[57] C Kunz, C Murphy, R Camilli, H Singh, J Bailey, R Eustice, M Jakuba, K i Nakamura, C Roman, T Sato, R A Sohn, and C Willis. Deep sea underwater robotic exploration in the ice-covered Arctic ocean with AUVs. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3654–3660. IEEE, September 2008.

[58] Benoit Landry, Robin Deits, Peter R Florence, and Russ Tedrake. Aggressive quadrotor flight through cluttered environments using mixed integer programming. *ICRA*, pages 1469–1475, 2016.

[59] S M LaValle. Randomized Kinodynamic Planning. *The International Journal of Robotics Research*, 20(5):378–400, May 2001.

[60] Thomas Léauté. Coordinating Agile Systems through the Model-based Execution of Temporal Plans. Master's thesis, July 2005.

[61] Thomas Léauté and Brian C Williams. Coordinating Agile Systems through the Model-based Execution of Temporal Plans. In *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, pages 114–120, 2005.

[62] Hui Li and Brian C Williams. Hybrid Planning with Temporally Extended Goals for Sustainable Ocean Observing. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*, 2011.

[63] Hui X Li. *Kongming: a generative planner for hybrid systems with temporally extended goals*. PhD thesis, Massachusetts Institute of Technology, 2010.

[64] Hui X Li and Brian C Williams. Generative Planning for Hybrid Systems Based on Flow Tubes. In *Proceedings of the Eighteenth International Conference on Automated Planning and Scheduling, ICAPS 2008, Sydney, Australia, September 14-18, 2008*, pages 206–213, 2008.

[65] Jyh-Ming Lien and Nancy M Amato. Approximate convex decomposition of polygons. *Computational Geometry*, 35(1-2):100–123, August 2006.

[66] Nir Lipovetzky and Hector Geffner. A Polynomial Planning Algorithm that Beats LAMA and FF. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS'17)*, 2017.

[67] Nir Lipovetzky and Hector Geffner. Best-first Width Search: Exploration and Exploitation in Classical Planning. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI'17)*, 2017.

[68] J Löhr, P Eyerich, T Keller, and B Nebel. A Planning Based Framework for Controlling Hybrid Systems. *ICAPS*, 2012.

[69] Derek Long and Maria Fox. Exploiting a Graphplan Framework in Temporal Planning. In *Proceedings of the Thirteenth International Conference on Automated Planning and Scheduling (ICAPS 2003), June 9-13, 2003, Trento, Italy*, pages 52–61, 2003.

[70] Tomás Lozano-Pérez and Leslie Pack Kaelbling. A constraint-based method for solving sequential manipulation planning problems. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 3684–3691. IEEE, 2014.

[71] Drew Mcdermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL - The Planning Domain Definition Language. Technical report, 1998.

[72] Nicola Muscettola, P Pandurang Nayak, Barney Pell, and Brian C Williams. Remote Agent: to boldly go where no AI system has gone before. *Artificial Intelligence*, 103(1-2):5–47, August 1998.

[73] Masahiro Ono. *Robust, Goal-directed Plan Execution with Bounded Risk.* PhD thesis, Massachusetts Institute of Technology, 2012.

[74] Masahiro Ono and Brian C Williams. Iterative Risk Allocation: A new approach to robust Model Predictive Control with a joint chance constraint. In *2008 47th IEEE Conference on Decision and Control*, pages 3427–3432. IEEE, 2008.

[75] Masahiro Ono, Brian C Williams, and Lars Blackmore. Probabilistic Planning for Continuous Dynamic Systems under Bounded Risk. *Journal of Artificial Intelligence Research*, 46:511–577, 2013.

[76] Florian Pantke, Stefan Edelkamp, and Otthein Herzog. Symbolic discrete-time planning with continuous numeric action parameters for agent-controlled processes. *Mechatronics*, 34:38–62, March 2016.

[77] Hae-Won Park, Patrick M Wensing, and Sangbae Kim. Online Planning for Autonomous Running Jumps Over Obstacles in High-Speed Quadrupeds. In *Robotics: Science and Systems*, 2015.

[78] Chiara Piacentini, V Alimisis, M Fox, and D Long. *Combining a temporal planner with an external solver for the power balancing problem in an electricity network.* Twenty-Third International ..., 2013.

[79] Wiktor Mateusz Piotrowski, Maria Fox, Derek Long, Daniele Magazzeni, and Fabio Mercorio. Heuristic Planning for PDDL+ Domains. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 3213–3219, 2016.

[80] N Ratliff, M Zucker, J A Bagnell, and S Srinivasa. CHOMP: Gradient optimization techniques for efficient motion planning. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 489–494. IEEE, 2009.

[81] S Richter and M Westphal. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 2010.

[82] Pedro Santana. *Dynamic execution of temporal plans with sensing actions and bounded risk.* PhD thesis, Massachusetts Institute of Technology, 2016.

[83] Pedro Santana, S Thiebaux, and Brian C Williams. *RAO*: An Algorithm for Chance-Constrained POMDP's.* Thirtieth AAAI Conference on ..., 2016.

[84] Pedro Santana, Tiago Vaquero, Eric Timmons, Brian C Williams, Catharine McGhan, Richard M Murray, and Claudio Toledo. Risk-aware Planning in Hybrid Domains: An Application to Autonomous Planetary Rovers. In *The AIAA Space and Astronautics Forum and Exposition (AIAA SPACE)*, 2016.

[85] Emre Savas, Maria Fox, Derek Long, and Daniele Magazzeni. Planning Using Actions with Control Parameters. In *ECAI 2016 - 22nd European Conference on Artificial Intelligence, 29 August-2 September 2016, The Hague, The Netherlands - Including Prestigious Applications of Artificial Intelligence (PAIS 2016)*, pages 1185–1193, 2016.

[86] Enrico Scala, Patrik Haslum, Sylvie Thiébaux, and Miquel Ramírez. Interval-Based Relaxation for General Numeric Planning. *ECAI*, 2016.

[87] J Schulman, J Ho, A Lee, I Awwal, H Bradlow, and Pieter Abbeel. Finding Locally Optimal, Collision-Free Trajectories with Sequential Convex Optimization. *Robotics: Science and ...*, 2013.

[88] Ji-Ae Shin and Ernest Davis. Processes and continuous change in a SAT-based planner. *Artificial Intelligence*, 166(1):194–253, 2005.

[89] David E Smith and Daniel S Weld. Temporal Planning with Mutual Exclusion Reasoning. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31 - August 6, 1999. 2 Volumes, 1450 pages*, pages 326–337, 1999.

[90] Siddharth Srivastava, Eugene Fang, Lorenzo Riano, Rohan Chitnis, Stuart J Russell, and Pieter Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *2014 IEEE International Conference on Robotics and Automation, ICRA 2014, Hong Kong, China, May 31 - June 7, 2014*, pages 639–646. IEEE, 2014.

[91] Mark Stefik. Planning with constraints (MOLGEN: Part 1). *Artificial Intelligence*, 16(2):111–139, 1981.

[92] Marc Toussaint. Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 1930–1936, 2015.

[93] David Wang. *A Factored Planner for the Temporal Coordination of Autonomous Systems*. PhD thesis, Massachusetts Institute of Technology, May 2015.

[94] David Wang and Brian C Williams. tBurton: A Divide and Conquer Temporal Planner. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA.*, pages 3409–3417, 2015.

[95] Brian C Williams and P Pandurang Nayak. A reactive planner for a model-based executive. 15:1178–1185, 1997.

[96] Steven A Wolfman and Daniel S Weld. The LPSAT Engine & Its Application to Resource Planning. *IJCAI 2016*, 1999.