2016-08-01

From Qualitative to Quantitative: Supporting Robot Understanding in Human-Interactive Path Planning

Daqing Yi
Brigham Young University

From Qualitative to Quantitative: Supporting Robot Understanding in Human-Interactive Path Planning

Daqing Yi

A dissertation submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Michael A. Goodrich, Chair
Kevin D. Seppi
Randal W. Beard
Mark J. Clement
Parris K. Egbert

Department of Computer Science

Brigham Young University

ABSTRACT

From Qualitative to Quantitative: Supporting Robot Understanding in Human-Interactive Path Planning

Daqing Yi
Department of Computer Science, BYU
Doctor of Philosophy

Improvements in robot autonomy are changing human-robot interaction from low-level manipulation to high-level task-based collaboration. When a robot can independently and autonomously executes tasks, a human in a human-robot team acts as a collaborator or task supervisor instead of a tele-operator. When applying this to planning paths for a robot's motion, it is very important that the supervisor's qualitative intent is translated into a quantitative model so that the robot can produce a desirable consequence.

In robotic path planning, algorithms can transform a human's qualitative requirement into a robot's quantitative model so that the robot behavior satisfies the human's intent. In particular, algorithms can be created that allow a human to express multi-objective and topological preferences, and can be built to use verbal communication.

This dissertation presents a series of robot motion-planning algorithms, each of which is designed to support some aspect of a human's intent. Specifically, we present algorithms for the following problems: planning with a human-motion constraint, planning with a topological requirement, planning with multiple objectives, and creating models of constraints, requirements, and objectives from verbal instructions. These algorithms create a set of robot behaviors that support flexible decision-making over a range of complex path-planning tasks.

Keywords: Path Planning, Human-Robot Interaction, Language Understanding

ACKNOWLEDGMENTS

First and foremost I would like to express my deepest gratitude to my advisor, Dr. Michael A. Goodrich. I am very grateful for his continuous guidance and support throughout the research, for his encouragement. Particularly, Dr. Goodrich is very patient in instructing me in research and language, and also trusts and supports to me in taking challenges. I would also like to express special thanks to Dr. Kevin D. Seppi. I am very fortunate to have worked with him and have his advice and support in my research and life. I want to extend my appreciation to the rest of my dissertation committee: Dr. Randal W. Beard, Dr. Mark J. Clement, and Dr. Parris K. Egbert for their support over the years.

I would like to thank Dr. Thomas Howard at University of Rochester for being a mentor, a collaborator, and a friend. My sincere thanks also go to all my previous and current lab mates for the support in project and research, the help in study and life and the happy time we have spent together. I also want to express acknowledgement to all my course instructors and the BYU Computer Science Department for helping me to grow from a graduate student to a confident researcher.

I want to dedicate this dissertation to Dr. Jiang, who was my previous advisor and recently passed away. He gave my first lesson in conducting research I owe special thank to Dr. Yuan for mentoring in life and career. I also thank Dr. Lin and Dr. Han for their help and support in my difficult time. I would like to thank all my friends at BYU for making this a wonderful journey and a precious memory.

Last but not least, I will not finish this dissertation without the love and support from my family. Thank you to Dr. Ding and his family for a home in Utah, where we always spent important days together. Thank you to my wife Weiwei, who always provided support and encouragement, especially when I was pressed for time before deadlines. Thank you to my parents, who always supported me throughout my education, even though I decided to move across the ocean from them.

Table of Contents

List of Figures

List of Tables

Chapter 1

Background, Motivation, and Overview

1.1 Introduction

This chapter presents a logical view of the chapters of the dissertation. It gives an overview of the problem that translates from qualitative information to quantitative solutions. The problem-solving consists of language understanding, multi-objective path-planning and topology-based path-planning. This chapter explains how these components in the dissertation are connected and how they relate to the chapters of this dissertation.

1.1.1 Robotic Path Planning in Human-Robot Collaboration

The ways robots and humans "think" are very different. Humans are experts at making qualitative decisions, from low-level movement to high-level reasoning, and robots show great strength in solving quantitative problems like high-speed data processing, high-precision repeated motion, etc. Robots can

- deliver constant and stable performance without fatigue;
- collect versatile formats of observation data by different sensors;
- access some particular spaces that humans cannot reach; and
- do repetitive tasks and detailed computations without error, etc.

Humans can

- adapt to environment change in task execution;

- respond to unexpected occurrences that happen accidentally; and
- be robust to vague or conflicting pieces of information.

In organizing humans and robots in a team, the differences between humans and robots cannot be ignored. Due to the differences, humans and robots are often assigned different roles when collaborating in a team. In teamwide collaboration, there is often a team task, which is defined as the goal of the team. A team "coach" splits the team task into subtasks for team players. In the spirit of human-centered intelligence, we assume that the team "coach" is a human who executes high-level planning, splits tasks, and schedules teammates. The team players include both humans and robots. For example, in a search-and-rescue task, robots perform dull, dirty and dangerous tasks, and alert a human only when an abnormality is detected [87]. The human then deals with abnormal occurrences. The robot players explore regions that are inaccessible and dangerous to the humans, and assist the human players when needed. Particularly when situations change, the human players are needed to organize the robot players.

Because task requirements from a human coach are usually qualitative [18], how they can be precisely translated by robots into quantitative optimization problems is a big challenge. When a few requirements are ambiguous or difficult to describe, an interactive processes can be used to identify the human intent. Moreover, we have to consider the collaboration between the human players and the robot players in task execution. For example, there might exist constraints between the human's motion and the robot's motion in evaluating the team performance [124]. Generally, the robot is expected to translate qualitative information from the human into quantitative path planning.

1.1.2 Translating qualitative information into quantitative path planning

Allowing a human to use qualitative instructions to guide a robot simplifies the human's task. In modeling a path-planning problem, qualitative information needs to be translated into quantitative information, including geographic information, objectives to be optimized,

and constraints. From the qualitative perspective, we are interested in several phenomena associated with how humans express intent.

- *Humans tend to describe task instructions by natural language.* Language is an efficient way for humans to communicate, and robots may be expected to communicate with humans by language [51, 63]. Although progress is being made on the problem of human-robot communication via natural language, much work needs to be done to solve the problem in general and to apply natural language to path-planning in particular. In path-planning, robots need to be able to both interpret the qualitative semantic structures associated with a human's intent as well as to construct quantitative models that can be computed by processors. A key problem is establishing a shared spatial common ground between the humans and the robot teammates.

- *There often exist multiple objectives in a human's instructions.* In planning a path, there is often more than one factor to consider [4]. For example, in motion-planning for a rescue task, a human coach might hope a robot's search not only covers the most likely regions but also focuses the robot's effort in risky regions that humans should avoid [123]. In this example, there are two objectives: maximizing information and minimizing the risk to the robot. Extending the example, if the robot is also expected to reach some goal quickly, minimizing the path length should be added as a new objective. Human preferences on different objectives are often qualitative, meaning that they may be difficult to represent by the kinds of quantitative weights used in an optimization algorithm. Moreover, the objectives can conflict or be incomparable. Therefore, the robot needs to model multi-objective path planning problems and find solutions by prior or posterior information [82] that the human coach gives.

- *Humans often have topological preferences for planned paths.* Spatial constraints, such as those created by finding a path through obstacles, divide paths into different topologies [12]. It is natural that a human coach has a preference for some classes of paths over others [123]. These preferences may come from task requirements, team

scheduling constraints, or the properties of an agent. These topological differences can be represented by either a soft path shape constraint or a preference ranking. For example, the human coach hopes that a robot goes to some goal location as quickly as possible. In this case, there is no hard constraint of visiting some regions while avoiding some other regions. But visiting several regions, which means the paths belong to specific homotopic classes, is implicitly helpful to the performance, i.e. safety. This type of qualitative information can be important to the task performance and should not be ignored in planning paths.

This dissertation presents a series of robot motion-planning algorithms, each of which is designed to support some aspect of a human's intent. Specifically, we present algorithms for the following problems: planning with a human-motion constraint, planning with a topological requirement, planning with multiple objectives, and creating models of constraints, requirements, and objectives from verbal instructions. These algorithms create a set of robot behaviors that support flexible decision-making over a range of complex path-planning tasks.

1.2 Related Work

This section begins by reviewing research on human mental models relevant for robot motion planning. The section then reviews related work that are specific to the algorithms presented later in the dissertation. Since the dissertation is organized as a series of papers, some of the literature review is performed in individual papers.

1.2.1 Mental Models in Path Planning

In the literature on human-machine interaction, mental models play key roles. Operationally, a mental model is "a representation of how current states are turned into consequences through the actions of an agent" [43]. When we have humans and robots in a team, a shared mental model is used as a "hypothetical construct" [122] that models and explains certain coordinated behaviors of teams. It provides a framework of mutual awareness, which serves

as the means by which "an agent selects actions that are consistent and coordinated with those of its teammates" [1, 57, 75, 84, 85, 121]. Figure 1.1 illustrates the information flow in the shared mental model of the coach-based human-robot team described above.

A human's intent can be operationally defined as a preference over consequences produced by a robot's actions and possibly as a preference over the actions used by the robot to produce a desirable consequence. Because we assume that a robot's task is given by a human coach in a human-robot team, information flows from the human coach's mental model to the team players' mental models. Correctly modeling the human's intent determines the performance of task execution. However, a human's intent is often only qualitatively expressed, making it difficult to translate intent into quantitative models.

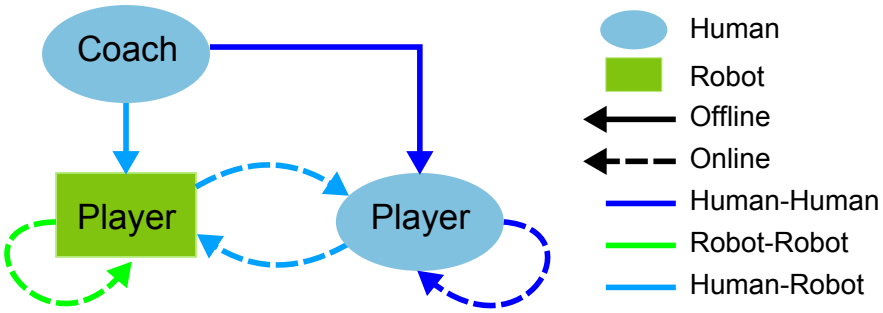


Figure 1.1: The information flow in the team.

In path-planning problems, the mental model provides goals and preferences for evaluating the performance of tasks. These goals and preferences are used to find optimal paths for task execution [26]. Current research work focus on introducing new features to the robotic mental models for creating new affordances, especially in interpreting humans' mental models. In human-machine interaction, the term "affordance" refers to "the perceived and actual properties of the thing, primarily those fundamental properties that determine just how the thing could possibly be used" [86].

This dissertation presents work that explores how to translate a human's qualitative intent into problems with quantitative models, and then how to solve the resulting quantitative problems. A qualitative map description is one common type of qualitative information

associated with a human's mental model [66]. Based on a human-sketched map, a probabilistic topological structure can be created to model a map with uncertainty [104]. Observations in navigation are used to correct the probabilistic topological structure, which is used as an estimated map. Planned paths are then adjusted online by the changes in the estimated maps. Qualitative information can also be represented by geometrical relations [77, 78] with landmarks. Within the constraints of feasibility, the estimated map structure can be updated by noisy sensory measurements.

Another important type of qualitative source is human language [112, 120]. Natural language instructions can be used as observations in the robotic SLAM problem. In [120], three layers of maps, which are semantic, topological and metric, are maintained in a graphical model structure. Both sensor information and human instructions can be merged into an environmental learning process [5–7]. By using the Bayesian filter framework, human information is loaded into a softmax likelihood model so that it can be merged into a Bayesian update.

1.2.2 Algorithm-Specific Work

In order to support the phenomena we mentioned in Section 1.1.2, we now review several fields that are related. In how to model problems, we review literature in modeling problems from verbal instructions. In how to support problem solving, we review studies in path-planning problems with submodularity, homotopy and multiple objectives.

Model Problems from Verbal Instructions

Current human-robot communication relies heavily on training human operators, which means that the operators should customize information for robots. This prevents human-robot interaction being applied in more problems

In planning paths, spatial information in the language is a very important for communicating a mental model. It has been proposed that a spatial semantic hierarchy is

maintained in the mental model, which represents the information relationship from the sensory to the topological and the metrical [66]. There have been a few grammars introduced to parse sentences into semantic elements by task specifications, e.g. SDC (Spatial Description Clause) [112] and TBS (Tactical Behavior Specification) [15]. Grounding semantic elements is an important area of current work [63]. In path planning, problems are often framed as optimization problems that seek to find paths best match given a set of semantic meanings [15, 112]. This approach can be strengthened by integrating multiple information sources, such as visual sensors and odometers [45].

Planners have been proposed to infer a human's intent from a human's verbal instruction [38, 51]. In such planners, groundings are extracted from semantics in a working environment. The verbal instruction is parsed into phrases by the grammar of the Spatial Descriptive Clause. A factor graph is then created to model correspondences between groundings and phrases. Some labeled training dataset can be used to train the factor graph model. After training, the factor graph model can be used to infer the groundings of verbal instructions.

Multiple Objectives in Path Planning

Tasks assigned to robots are complex which means that they can be performed in several different ways. This complexity leads to the requirement that the robot must make tradeoffs among several different objectives. For example, a robot in a search task may be expected to maximize the coverage area while minimizing energy consumption and avoiding risk; see, for example the applications in [81, 123]. As another example, a robot manipulator may need to satisfy performance criteria related to movement, joint velocities, joint accelerations, etc. [89].

A common method of finding a solution of a multi-objective optimization problem is optimizing the single objective created by a weighted sum of the multiple objectives. In path-planning with multiple objectives, the properties of the path produced by this method are determined by how the objectives are weighted. This means that a programmer, designer,

or human teammate must decide how to assign the weights so that the quantitative behavior matches the intent. In addition to the burden placed on the human operator, optimizing a weighted sum does not work when the multiple objectives are very difficult to compare or are expressed in incommensurate units.

In solving a multi-objective optimization problem, the output is often a set of non-dominated solutions, which are also often called Pareto-optimal solutions [82]. "Non-dominated" means that each solution is at least better in one objective or equally the same in all the objectives when compared with any other solution. In this proposed work, we assume that there can be an interactive process where a human selects one solution from the set of Pareto-optimal solutions [73]. In multi-objective path-planning, a set of Pareto-optimal paths could be found as well. The Pareto-optimal path that best matches the human's intent can be selected as the solution that best balances tradeoffs between the objectives.

Most popular methods in multi-objective optimization are not naturally applicable to path-planning [34, 130]. One approach to addressing this issue is to change the representation for a path by coding a path as a sequence of fixed-length line segments represented by directions [4, 52] or waypoints [89, 111] so that an evolutionary algorithm can be applied. Unfortunately, these approaches do not scale well for large problems, and estimating the required number of segments is very challenging. Another approach is to represent the path as a point in a very high-dimensional vector space. In this approach a path is represented as a point in an $n * d$ dimensional space formed by n different d-dimensional way-points [3, 4]. However the search can be very difficult if we allow the number of way-points and, therefore, the dimensionality of the optimization problem to vary. The algorithm does not work well when the obstacles in the path-planning space introduce discontinuities in these high-dimensional spaces, which limits the applicability of heuristic-based search approaches [111, 130]. There is still a need for an algorithm that can efficiently and effectively find a set of Pareto-optimal paths for a given set of objectives.

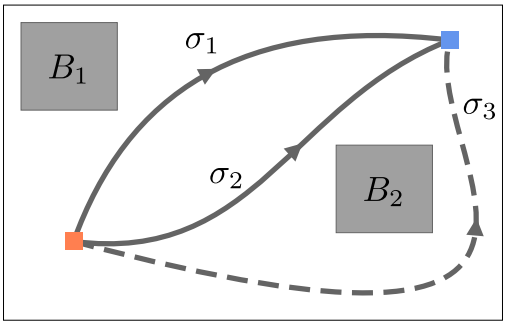


Figure 1.2: Path homotopy.

Homotopy in Path Planning

Unlike a point solution in common optimization problems, a path is not only evaluated by its fitness/cost but also its shape. Humans often represent the world using a topological-based rather than metric-based path planning [2, 66]. Various methods have been used to create topological-representations that can be used by path-planners for robots [40, 74, 104, 114]. Obstacles in a map divide paths into topological groups by the similarities in the shapes of the paths. The topological notion of *homotopy* presents a formal definition of the similarity between two paths. This definition can be used both to determine the similarity between two paths as well as to partition paths into different classes.

The topological concept of *homotopy* is a mathematical formalism of the inherent similarity or dissimilarity of two paths and allows us to precisely quantify such topological constraints. Given two paths, if one can be deformed into the other without encroaching any obstacle, they are said to be *homotopic* [48]. In Figure 1.2, the solid paths are homotopic. But they are not homotopic to the dashed path.

Homotopy-based path planning requires an algorithm to determine the homotopic equivalence of two paths, which is usually computationally expensive. There are a few research studies that focus on effectively and efficiently identifying the homotopy class to which a path belongs or determining the homotopic equivalence of two paths. The Voronoi diagram is used to identify a path from any homotopy class in [8]. By converting any path into a simple path from the Voronoi diagram, the homotopy class of the path can be determined.

However there exist limitations on finding some paths in a map with complex obstacles by this approach. By applying the funnel algorithm in the universal covering space, the minimum-length path is efficiently optimized in a given homotopy class in [49]. Similar to the Voronoi approach, the complexity of the problem is increased when the shapes of the obstacles are not smooth and convex. Semi-algebraic cuts are used to convert paths into "words" so that the homotopic equivalence can be recognized in [44]. Also, the Cauchy Integral theorem has been introduced to determine the homotopic equivalence of any two paths by marking some positions in obstacles as undefined [12]. Given two paths sharing the same start and goal, we can determine whether there is an obstacle inside a region that is enclosed by two paths by the value of the complex integral. Because the map is discretized, the computation cost expands greatly if some complex obstacles are reasonably approximated by the cells. How to efficiently and effectively find the optimal paths in different homotopy classes is still an open question.

Submodularity in Path Planing

Search is a very important task in path planning. In planning motion for a search task, the objective is usually to maximize the information collected with a limited motion resource. Because observations cover a region around the search agent, using a coverage model for the robot makes the path planning a *maximum coverage problem*. Maximum coverage is known to be a classic NP-hard combinatorial optimization problem [80] because coverage overlap cannot be ignored. Mutual information is introduced to measure the total information of a set of observation coverages, which implies a property of "nondecreasing submodularity" [108]. Submodularity is defined as a function property [108], which is:

$$\forall A \subseteq B \subseteq V \text{ and } s \in V \setminus B; f(A \cup s) - f(A) \leq f(B \cup s) - f(B). \tag{1.1}$$

A greedy approximation with a known performance bound can efficiently exploit the submodularity property of mutual information [108]. A branch and bound approach can also be used in informative path planning [13].

Maximizing the reward collected from a limited-length graph walk is usually known as an orienteering problem [118], in which the total reward is a sum of the rewards of visited vertices. If the reward function of a vertex has the submodularity property as in a maximum coverage problem, the problem is defined as a submodular orienteering problem [24]. Unfortunately, in the submodular orienteering case the location of the robot at time t constrains the reachable locations at time $t + 1$. Thus, naively applying a greedy algorithm to the submodular orienteering case, that is, with a "teleport" assumption, yields poor results [64]. For a generalization of the submodular orienteering problem in which the neighboring constraint can be converted into a time budget, a recursive greedy algorithm can be applied [24]. If the planning is considered for a human-robot team as we are proposing, the collaboration and the constraints between team players cannot be ingored. When we import the constraints from a human teammate's behavior to submodular path planning, this solution will not work any more.

1.3 Project Description

The papers in this dissertation model the following aspects of information flow from a human to a robot:

- How to support multiple objectives in tasks,
- How to support human's topological preference, and
- How to model problems from a language instruction.

Figure 1.3 illustrates the components and the relationship. Table 1.1 shows how the list of model requirements above can be interpreted as a general type of problem from the literature,

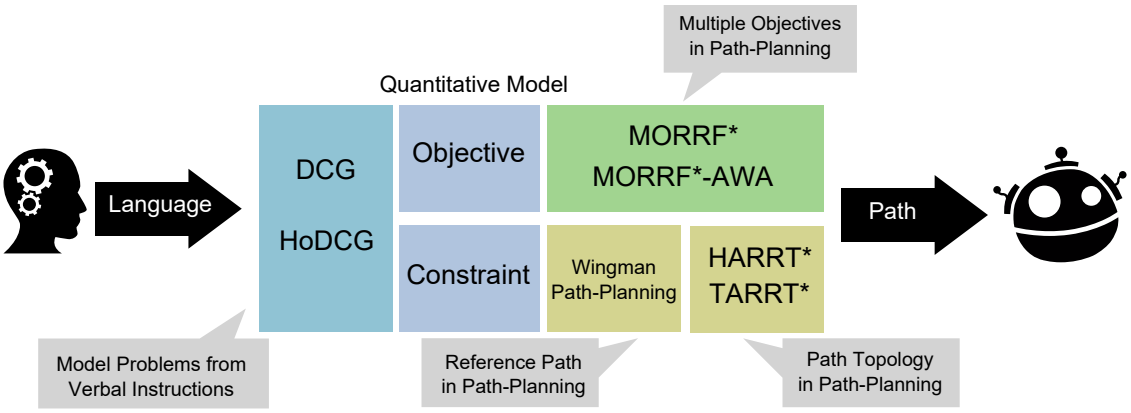


Figure 1.3: A framework from qualitative to quantitative.

and identifies a specific algorithm developed to solve the problem for path-planning in a human-robot team.

1.3.1 Model Path-Planning Problems from Natural Language Instructions

A human-natural way of human-robot interaction is to use human language. Instead of focusing on generic natural language processing problems, we only look at how path-planning problems can be modeled from semantic structures. The semantic structures are generated from verbal instructions by existing grammar parsers [63]. We need to provide methods to infer the task definition from the semantic structures, focusing on the requirements with multiple objectives and topological preferences.

The Tactical Behavior Specification (TBS) language has been introduced as a grammar to support instructions with spatial relations [15]. We extend this grammar so that it can be applied in our context in two ways: grounding adverbs and allowing topological preferences [123]. For example, if a human coach tells a robot to "go around building A quickly and carefully and then between the two trees while avoiding region C", it implies that there potentially exist two objectives from the adverbs, "quickly" and "carefully". Practically, this might mean to minimize the path length and minimize the risk. It also indicates the topology of the path that the human wants: "around building A", "between the two trees" and "avoiding region C". The semantic structures can be grounded into multi-objective path

Requirement	**Problem**	**Algorithm**
Model problems from verbal instructions	Grounding multiple objectives and topological preference in natural language	HoDCG
Support multiple objectives in tasks	Multi-objective path planning	MORRF* MORRF*-AWA
Support humans' topological preference	Submodular path planning with reference path constraint	Wingman path-planning
	Homotopy-based path planning	HARRT* TARRT*

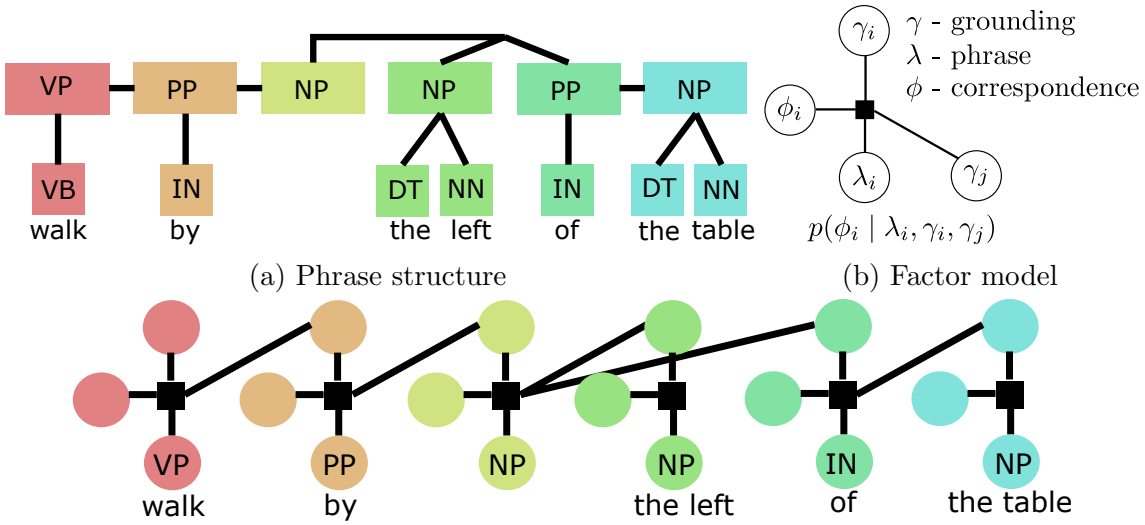
Table 1.1: Project description.

planning, homotopy-based path-planning or submodular path planning with constraints. The details are provided in Chapter 4.

We use a graphical model to infer the relations between the semantics and the groundings. The graphical model has been a popular tool to model ambiguous relationships between utterances and semantics. After training by language samples, the graphical model could infer the meaning of verbal language. Similarly, we can use it to understand a human's verbal commands to a robot. In the application of path-planning, the human's verbal command is associated with spatial elements. Therefore, the Spatial Descriptive Clause (SDC) has been introduced to parse the verbal command into a sequence of phrases [63].

The process of assigning physical meanings to phrases is called *grounding*. A verbal sentence is decomposed into phrases λ_i^r, which carry the information from the human. Grounding variables γ_j^r are extracted from the map, which include

- *verb fields* : the change in orientation between the viewpoints from verb/action semantic,
- *figure and landmark fields* : the viewpoint transition and the detected objects, and
- *spatial relations* : the functions of the geometries of the paths and the landmarks.



(a) Phrase structure

(b) Factor model

(c) Factor model

Figure 1.4: How a factor graph is constructed.

By assuming that the groundings are conditionally independent, a conditional random field (CRF) can be used as an undirected graphical model for the relationship between the phrases and the groundings. Correspondences ϕ^k are introduced to indicate whether a grounding element is linked with a phrase from the SDC [112], indirectly, it measures how the grounding variables match the verbal sentence. The corresponding node ϕ^k connects the grounding node $gamma_j^r$ and the phrase node λ_i^r. A labeled dataset can be used to train the graphical model. The objective of the training is maximizing the likelihood of the correspondences.

$$\arg\max_{\phi_{i,j} \in \Phi} \prod p(\phi_{i,j} \mid \gamma_j^r, \lambda_i^r) \tag{1.2}$$

Equation (1.2) can be converted into a graphical model. Firstly, a verbal command is parsed into a phrase structure. Figure 1.4a shows a phrase structure of a sentence "walk by the left of the table". Each phrase corresponds to a factor model as illustrated in Figure 1.4b. Each factor model defines the relationship among a phrase λ_i, a correspondence variable ϕ_i, a grounding γ_i and groundings in other models γ_j. By the structure of the phrase, a factor graph can be created. Figure 1.4c gives a factor graph that is created from the phrase

structure in Figure 1.4a. The trained graphical model can then be used to infer the most likely grounding nodes from a given verbal command.

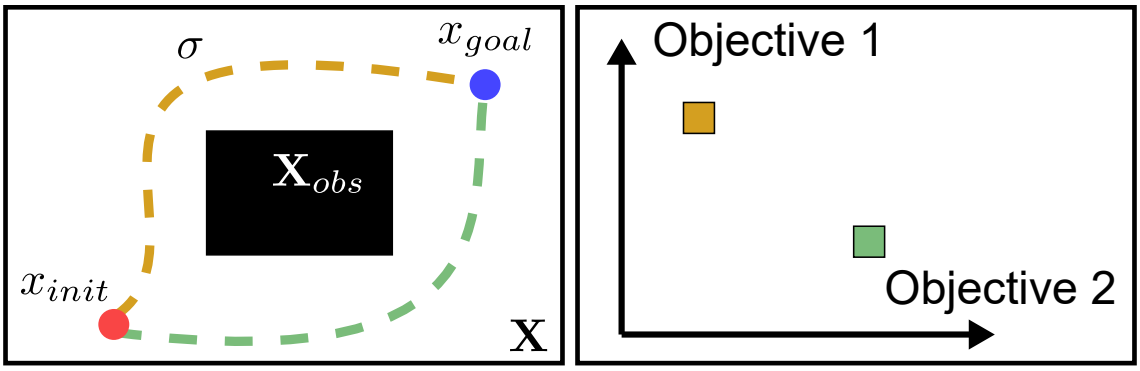
It is noticeable that some adverbs might mean different things in different contexts. For example, in a sentence of "put the pallet carefully on the truck", "carefully" requires a smooth motion. But in a sentence of "walk carefully in the crowd", "carefully" means avoiding collisions with someone else. It can also define multiple objectives as well, e.g. smoothly moving and collision avoiding simultaneously. A human language understanding process helps identify the human's intent behind an utterance. We can create a factor graph to model the correspondence between the adverbs and the intended objectives in different sentences and scenarios [112]. After training, it can be used to infer what kind of objectives that the human intended in the verbal command.

Topological preferences can be extracted from human language. In [51], the topological preference is inferred from human language and is used as a hard constraint in path planning. Sometimes, the topological preference can be a soft constraint. For example, if a human coach says "better avoiding region C", he/she prefers avoiding region C but not necessarily. The topological preference can also be the order of different homotopy classes. "Going through between A and B is better than between A and C" implies a homotopy class via the region between A and B is better than a class via the region between A and C. We present an algorithm that constructs a factor graph to represent the preference relationship between the groundings and the verbal phrases.

1.3.2 Multi-Objective Path Planning

To support a complex decision with multiple objectives, we define a multi-objective path planning problem and present an algorithm to find solutions.

When there are multiple objectives in a task, the goal is to find a solution that trades off between the objectives. A common method for finding a solution to a multi-objective optimization problem is to optimize a single objective created by a weighted sum of the



(a) Solution space. (b) Objective space.

Figure 1.5: Multi-objective path-planning.

multiple objectives. This means that either a programmer, designer , or a human teammate must decide how to assign the weights so that the qualitative behavior matches what is intended. Optimizing a weighted sum does not work when the multiple objectives are very difficult to compare or are expressed in incommensurate units. Thus, it will be better if a set of Pareto-optimal paths is firstly found and the human teammate can select one of several options.

Given this context, we now define the multi-objective path-planning problem to find a set of Pareto-optimal paths. Consider a multi-objective path-planning problem defined on a bounded, connected open set of possible solutions (in Figure 1.5a), and K different objectives (in Figure 1.5b). Without loss of generality, assume that each objective is to minimize these functions. Since the Pareto-optimal set of paths in $\mathbb{R}^d$ is not enumerable, the *goal is to find a representative, finite (M-element) subset of the Pareto-optimal set.* By "representative" we mean a diverse set of solutions that span the Pareto set rather than, for example, several points clustered in a single region of the Pareto set.

In contrast to searching through and comparing solutions in order to find a Pareto-optimal set, we use a decomposition-based method similar to MOEA-D [130]. MOEA-D is an algorithm that decomposes a multi-objective optimization problem into a set of subproblems. Each subproblem uses a weighed combination of the objectives to find specific points in the

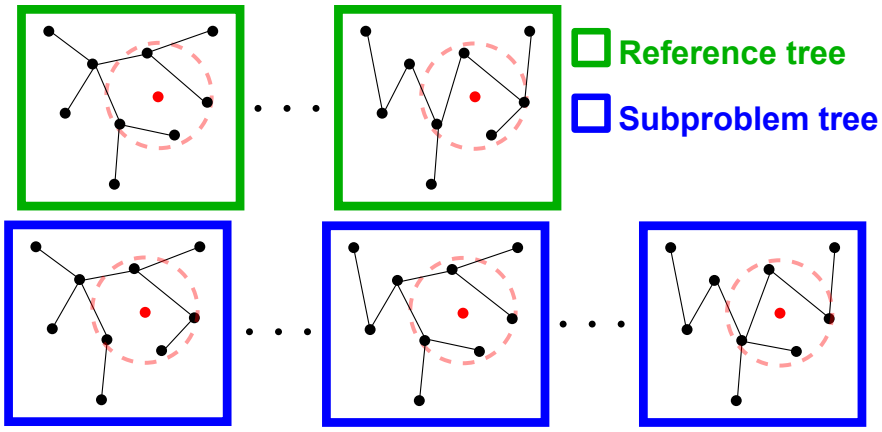


Figure 1.6: Rapidly exploring process

Pareto set or to guide the search for such points. The solutions generated by each method are a subset of the Pareto-optimal set.

Chapter 5 presents an algorithm that explores the solution space using RRT*-based tree structures but uses multiple trees in the spirit of decomposition-based multi-objective optimization. Because a set of trees are constructed in the exploration process, we call the algorithm MORRF* (Multi-Objective Rapidly exploring Random Forest*). Adopting the idea from the MOEA-D algorithm [130], the M elements in the solution set Σ^* will be obtained from M subproblems decomposed from the multi-objective problem.

Thus K reference trees are used, one each to explore the minimum of each objective, and M subproblem trees are used, one for each weighting vector, λ_m. The K reference trees and M subproblem trees constitute the exploration forest, as in Figure 1.6. The solutions obtained from M subproblem trees constitute a set of Pareto-optimal paths, which are the solutions to the multi-objective path-planning problem.

Weights for defining subproblems are uniformly sampled from a high-dimensional simplex for better diversity of solutions. However, the mapping from weight space to solution space is not linear. It means that the diversity of solutions still cannot be guaranteed. In order to maximize the diversity, Chapter X presents an algorithm called MORRF*-AWA, which is an enhanced MORRF* with "adaptive weight adjustment", which is discussed in Chapter 6. The notion of *sparsity level* is introduced to measure the how well the set of

possible solutions is being explored. Subproblem trees with lower sparsity levels will be removed while exploring the space. New subproblem trees will be created by weights that are likely to lead to solutions with high sparsity level. As a result, the diversity of the solutions can be increased.

1.3.3 Involving Topological Preferences in Path-Planning

We categorize topological preferences into two types by considering the level of collaboration. If a human and a robot are constrained in a collaborative movement in shared space, the motion of the robot has to be constrained in a region near the position of the human. A topological preference is defined by a reference path from the human. By considering the sensor coverage overlap in collaboration, we can model the problem as submodular path-planning problem with a reference-path constraint, which is described in Section 1.3.3. If we ignore the collaboration, the topological preference can then be presented by the homotopy classes in the map. Therefore, in this case, we can model the problem as homotopy-based path-planning problem in Section 1.3.3

Submodular Path-Planning with Reference Path Constraint

Before task execution, a coach can assign a subtask for a robot player to explore the world while staying near a human teammate. For this type of topological preference, the information about the human is given to the robot. Thus, assuming that the human player's motion can be predicted or is known, the robot player's motion is constrained by the human player's motion, yielding a path-planning problem with a reference-path constraint. We call this situation the *robot wingman* problem, which is addressed in Chapter 2

The coverage model is chosen to be the observation model of a search agent. Information entropy is used to model the information gain in the cooperative search in a human-robot team. We discretize a workspace into a set of cells so that the movement is constrained in

(a) A coverage model of observation. (b) A role of robot wingman.

Figure 1.7: A Robot Wingman Framework.

transiting from one cell to any of its neighbors. Also the observation of a search agent covers not only the cell he/she occupies but also neighboring cells within a given range.

We select the wingman as a role for the robot player, though there are a few roles that define how the robot's path is constrained by a human. The wingman role requires that the robot player remains within a tolerance range near around the human player while the human player is moving, as in Figure 1.7b.

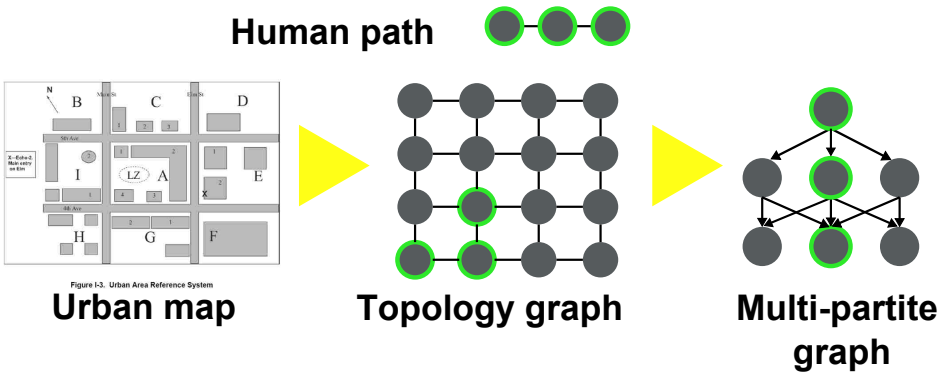
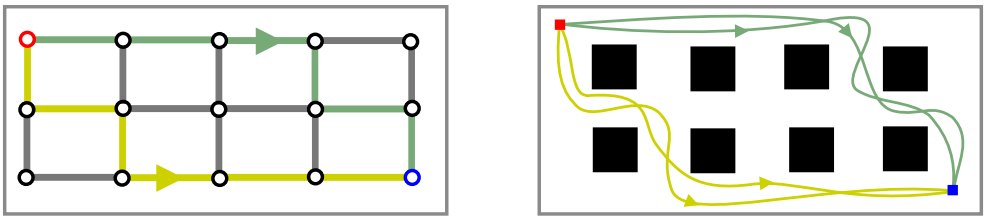


Figure 1.8: A Robot Wingman Framework.

Chapter 3 presents a solution to the plannning problem of a robot in a human-path constraint. Assume the robot has a model that can predict the human's path. At each time step, the wingman relationship induces the set of *visitable cells* for the robot. We can construct a multi-partite graph from the human-constrained path. Each partition is consist of the set of visitable cells at a corresponding time step. The edges are determined by the



(a) Topology space. (b) Path space.

Figure 1.9: Map with obstacles.

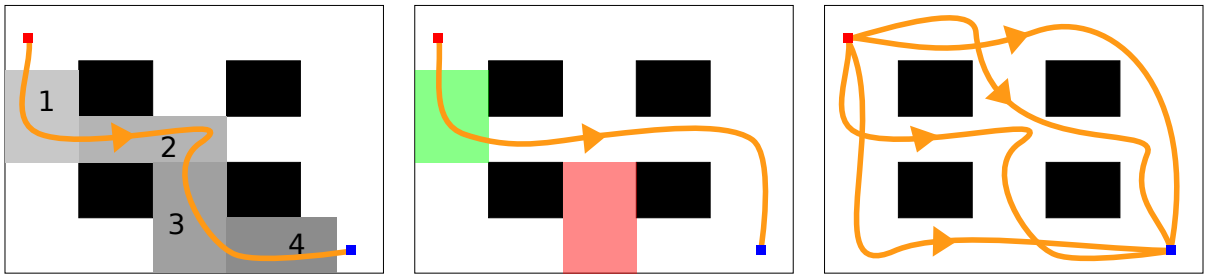
neighborhood of each cell from the discretized map. Figure 1.8 illustrates how a multi-partite graph is extracted from a workspace for planning.

In order to guarantee that the search process on a multi-partite graph always ends with a feasible path, we use a pruning process to ensure that each vertex can be reached from the previous partition and is connected to a vertex in the next partition. We propose to use backtracking to estimate the maximum total reward and use this estimation as our search heuristic. We then propose to use an expanding tree to create an anytime algorithm that approximates solution to the submodular orienteering problem on the multi-partite graph.

Homotopy-based Optimal Path Planning

Observe that, in planning paths for a robot player, a coach may want to express topological constraints like "go to the left of the obstacle". A set of paths that are homotopic to each other forms a *homotopy class*, and the set of homotopy classes partition the set of all possible paths between any two points A and B. In an environment containing obstacles, the homotopy partition can provide a useful way of grouping paths together based on the similarities of their "shapes", where the term "shape" is interpreted by using the formal topological notion. A human can reason in topology space, as in Figure 1.9a. While, a robot can utilize the topological information from the human to find paths, as in Figure 1.9b.

Chapter 7 presents an algorithm for homotopy-based path-planning problems. Reference frames are randomly generated that separate a map into subregions. They are used to create a homotopic DFA that converts any path into a string. The strings of paths can be



(a) Specifying a topology. (b) Visiting or avoiding regions. (c) No topology preference.

Figure 1.10: Different levels of topology information.

used to identify the path homotopy. We firstly proposed a homotopy-aware RRT* (HARRT*) algorithm. This algorithm uses an RRT* algorithm to explore possible paths, but each branch of a random tree is aware of the homotopy class to which it belongs. HARRT* use a bidirectional RRT* to obtain the optimal cost-to-come and cost-to-go of different positions. We can eventually have the optimal path with a via-position constraint by combining (a) the optimal subpath from the start to the via-position with (b) the optimal subpath from the goal to the via-position. In order to explore any possible homotopy class (including winding homotopy class), we later proposed a topology-aware RRT* (TARRT*), which is introduced in Chapter 8 The exploration of a homotopy class is enforcing branching by a sequence of subregions that is determined by the class.

Theoretical analyses of both algorithms are provided in Chap 9. The homotopy-based optimal path-planning supports different levels of topology information from human. Three levels are illustrated in Figure 1.10. Figure 1.10a means a defined topology from a human, which requires that a planner returns the optimal path of the topology. Figure 1.10b shows that a visiting region (green) and an avoiding region (red) specified by a human. This determines one or several eligible homotopy classes for planning paths. Another scenario is that a human has no topology preference. A planner should explore several possible homotopy classes, as in Figure 1.10c. It returns the optimal paths of them to the human to select.

1.4 Dissertation Chapters

This dissertation consists of nine papers, two of which are being prepared for submission. The papers are generally included in chronological order. The order is also determined by relations and dependencies. We also include one extra paper in the Appendix. This section gives a brief description of each chapter.

Chapter 2, which was published in [42], presents a Bayesian approach of modeling task-based mental models of a human-robot team. By using a robot wingman in a search task as an example, the shared mental model of a human-robot team is modeled as an information-update process.

Chapter 3, which was published in [124], presents algorithmic solutions to the cooperative team search with a robot wingman, which is defined in Chapter 2. A human-path constraint converts a search space into a multi-partite graph. A backtracking heuristic is integrated into an anytime framework to find solutions that maximize the robot's performance in a human-robot team.

Chapter 4, which was published in [123], presents a semantic-based path-planning algorithm to support collaboration in human-robot teams. The high-level human information is expressed by language command. The semantic information defines path-planning problems that leads to low-level manipulation of robots, which reveals multiple objectives and path-topology requirements.

Chapter 5, which was published in [125], presents an algorithm for multi-objective path-planning problems. The algorithm, MORRF*(Multi-Objective Rapildly-exploring Random Forrest*), uses a set of trees parallel exploring the solution space. The solutions that are returned by the trees asymptotically converge to Pareto-optimal paths.

Chapter 6, which is in preparation to be submitted to the *Journal of Artificial Intelligence Research*, extends MORRF* in Chapter 5. It introduces a new approach of defining subproblems and presents more complete performance analyses. We also introduce *adaptive weight adjustment* to MORRF*, which leads to a new algorithm MORRF*-AWA.

The new algorithm adjusts the weights in exploration that provides better diversity in the Pareto-optimal solutions.

Chapter 7, which was published in [128], presents a framework of homotopy identification. It is used to support homotopy-aware RRT* that uses bidirectional RRT* to find the optimal paths of simple homotopy classes.

Chapter 8, which was published in [127], presents a new algorithm, topology-aware RRT*, that extends the homotopy-class exploration from simple homotopy classes to all homotopy classes.

Chapter 9, which is in preparation to be submitted to *IEEE Transactions on Robotics*, integrates HARRT* in Chapter 7 and TARRT* in Chapter 8. It also provides theoretical analyses on both algorithms.

Chapter 10, which was published in [129], presents a language understanding algorithm, HoDCG, which is a factor graph created by the relationship of semantic elements. Path-topology information from human-language instructions is grounded to create path-planning problems with topological constraints. Path-planners can then be used to generate paths that satisfy the topological constraints. This process converts language instructions into paths for robotic navigation.

Chapter 11 concludes with findings of the dissertation work and summarizes the contribution of the dissertation. It also describes possible future work.

Appendix A, which was published in [126], introduces input-to-state stability to the convergence analysis of PSO algorithm, which models it as a consensus reaching problem in a swarm of particles. The results can be used to understand the dynamics of the optimal search in PSO and guide the parameter selection. This paepr in the appendix is not central to the theme of the dissertation, but is included because it is interesting work done while performing the rest of the dissertation research.

Chapter 2

Toward Task-Based Mental Models of Human-Robot Teaming: A Bayesian Approach [1]

Abstract

We consider a set of team-based information tasks, meaning that the team's goals are to choose behaviors that provide or enhance information available to the team. These information tasks occur across a region of space and must be performed for a period of time. We present a Bayesian model for (a) how information flows in the world and (b) how information is altered in the world by the location and perceptions of both humans and robots. Building from this model, we specify the requirements for a robot's computational mental model of the task and the human teammate, including the need to understand where and how the human processes information in the world. The robot can use this mental model to select its behaviors to support the team objective, subject to a set of mission constraints.

[1] Published in 5th International Conference, VAMR 2013, Held as Part of HCI International 2013. Authors are Michael A. Goodrich and Daqing Yi.

2.1 Introduction

In complex, rapidly evolving team settings in which a robot fulfills a role, the robot needs sufficient autonomy to allow its human teammates to be free to direct their attention to a wider range of mission-relevant tasks that may or may not involve the robot. In contrast to many prior applications in which the robot was either teleoperated or managed under strictly supervisory control [88], recent advances in robot technologies and autonomy algorithms are making it feasible to consider creating teams in which a robot acts as a teammate rather than a tool [22].

In this team-centered approach, both humans and robots can take on roles that match their strengths. Properly designed, this can facilitate the performance of the entire team. This idea has already been applied to reform human-robot interaction in many areas, like object identification, collaborative tasks performance, etc. [50]. In this paper, we adopt the notion of collaboration, operationally defined as the process of utilizing shared resources (communication, space, time) in the presence of asymmetric goals, asymmetric information, and asymmetric abilities as illustrated in Fig. 2.1. The word collaboration suggests that there are both overlaps and differences between the goals, information, and abilities of the agents involved. Colloquially, collaboration can happen when everyone has something unique to offer and something unique to gain, but there is some benefit to each individual if activity is correlated.

In a human-robot team, the asymmetries on abilties and information mostly come from the natural difference on agents' sensors and actuators. Additionally, an agent may exhibit ability and information asymmetry in different states of interacting with the environment, like location, lighting condition etc. Often, a team goal will be decomposed into subgoals in execution. The subgoals are usually assigned to agents in the team by organizing agents into specific roles with specific responsibilities, and this leads to goal asymmetries. In a collaboration framework, the interaction between agents not only focuses on common goals, but may also require providing support for others' goals. In a team search tasks, for example,

Figure 2.1: Operational Elements of Collaboration.

the robot and the human might work together for target searching, while the robot might assist the human to deal with an emergency.

Collaboration is a form of teamwork that benefits from an explicit representation of shared intent. The theory of shared intent suggests that both the human and the robot need to have a mental model for the task to be performed and another mental model for how other team members will act [98]. The primary contribution of this paper is a framework for developing a task-based mental model from a human-robot collaboration perspective, including the ability to represent and reason about contributions of other team members to the mission and estimation of how other team members' actions affect performance.

2.2 Shared mental model

From studies of cognitive psychology, the concept of a shared mental model has been proposed as a hypothetical construct, which has been used to model and explain certain coordinated behaviors of teams. Shared mental models provide a framework of mutual awareness, which serves as the means by which an agent selects actions that are consistent and coordinated

with those of its teammates. According to [75], in order to perform collaboratively as a team, members of a team must have the following:

- **Teammate Model**: knowledge of teammates skills, abilities and tendencies.
- **Team Interaction Model**: knowledge of roles, responsibilities, information sources, communication channels and role interdependencies.
- **Team Task Model**: knowledge of procedures, equipment, situations, constraints.

These elements determine (a) how an agent makes decisions as a member of the team and (b) how diverse capabilities and means of interactions are managed within an organizational context. These concepts have been incorporated as important elements in existing human-robot team designs [84]. From a robot's perspective, operating within the context of a human-robot team, the robot's shared mental model will help the robot predict information and resource requirements of its teammates. Importantly, a better understanding of task demands and how teammates will likely respond will enhance the robot's ability to support team-level adaptations to changes in the world.

Given a shared intent from the team, the robot is assigned or adopts tasks, either as an autonomous agent or as a collaborating teammate. What does the robot need to collaborate? We address two fundamental elements: (1) How should the robot model the task? (2) How should the robot model a human performing the task? We then illustrate how the concept of a shared mental model is applied within a search task by providing an example computational model that responds to these two questions.

2.3 Robot wingman in a search task

We introduce the shared mental model to a human-robot team search problem. In the problem, the search region is modeled with the belief of where the target objects are, and the search process works as constantly updating this belief by observations. Thus, teams of

humans and robots manage a region of space subject to particular time or timing constraints [95].

From prior work in search theory, search efficiency is usually considered as one of the essential factors to a task success, and is therefore a central element of the team's model of a search task. There are several parameters to measure the efficiency in a search task [95], which are determined by the observation capability of a search agent. In this paper, we are interested in:

- **Sweep Width:** a measure of how wide an area a searcher can, on average, effectively cover. More specifically, it represents how well a sensor (e.g., the human eye) can detect specific objects as a function of distance from sensor to object.
- **Coverage:** a simple measure of how well a segment was covered by all of the searchers. Coverage is a ratio calculated by summing up the area that each searcher covered and dividing by the area of the search segment.
- **Probability of Detection:** a measure of the probability of success. Search managers need a way to determine the probability that a lost object would have been found if it was actually in the segment that was searched.

Effective swept width and *coverage* are determined by the sensor model of a search agent; the sensor model encodes the characteristics and capabilities of the agent's sensors. This model defines what the observation range of an agent is, and how the observation uncertainty might change with the distance of a target object. By contrast, the *probability of detection* shows the probability that an object would have been detected if in the area, which can be modeled as (a) an agent's prior belief that an object is in the search region and (b) the quality of the agent's observation. Since all agents are imperfect detectors, there exist differences in detection success and detection times among agents.

There is relevant prior work on applying these concepts to human-robot teams. [23] and [37] import robots into urban search and rescue so that human unreachable locations

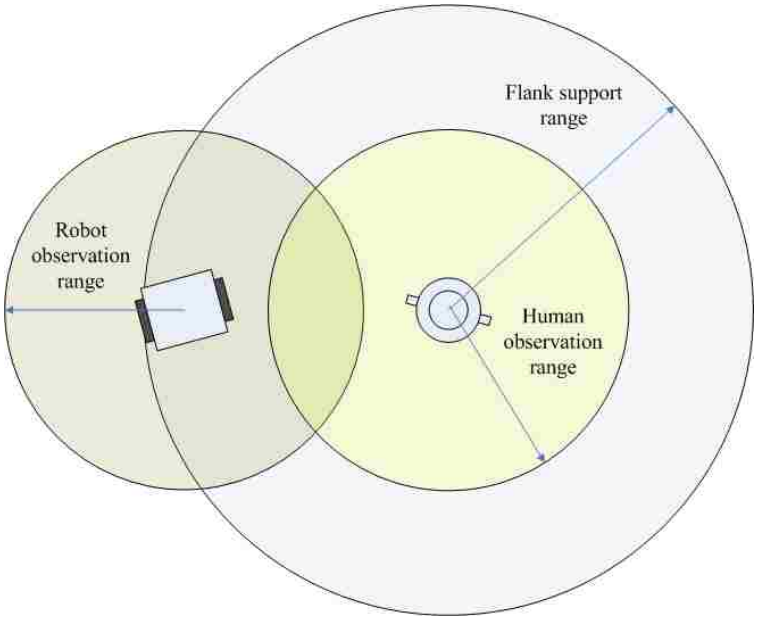


Figure 2.2: A Robot Wingman framework.

can be explored, which greatly extends the coverage of search task execution. Integrating various types of sensors, like radars, laser rangefinders, ultrasonic sensors etc. [97] [21], greatly expands the sweep width of a search team. [87] and [22] propose a way to improve the probability of detection using information fusion across multiple agents. Modeling the team as a distributed information fusion process exploits the asymmetric perception capabilities of humans and robots to enhance the search efficiency of the team.

In our proposed human-robot search team, we assume that the human is better at strategy and decision making and the robot is better at raw data collection. This assumption forms the basis for the robot's model of its teammate. We propose the notion of a *robot wingman* to support a human in a collaborative search task, which is to have a robot that accompanies a human as he or she navigates through some space. Since a robot may be able to detect certain types of signals not perceivable by a human (e.g., radio signals or chemical gradients), it is possible for the wingman robot to extend the team's perception not only in space but also in the type of data perceivable by the team. As shown in Fig.2.2, as a flank support range that constrains where the wingman can move. The robot wingman is expected to stay in an area determined by the flank support range around the human, when the human

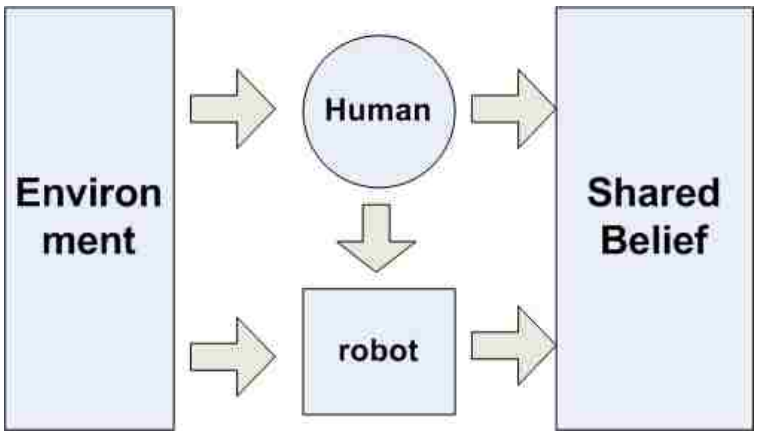


Figure 2.3: Information Flow in a Wingman Human-Robot team.

is moving for the search task. Doing so guarantees that the robot rapidly respond to the human needs assistance, which maintains a reasonable distance for supporting communication and coordination. In a shared human-robot search problem, the robot's role not only includes staying within the flank support range, but also includes gathering information about the world around the team.

The organization of the team determines how information flows when executing the search task. Thus, the organization is an important element of the team interaction model, with information flow acting as the currency of interaction. Information flow shapes the process of fusing asymmetric information for collaboration. In the next section, we model the belief of the locations of the search objects by a shared task model. The information comes from the observations from both the human and the robot. Meanwhile, the robot predicts how the human will work and what the information collected by human is like, and this prediction is used to make a decision on how to run the search operations as in Fig.2.3.

2.4 A Bayesian Approach

We present a Bayesian model for how information flows in the world and how information is altered in the world by the locations and perceptions of both humans and robots. Building from this model, we can specify the requirements for a computational mental model of the human teammate to understand where and how the human processes information in the

world. The robot can then select its behaviors to support the team objective, subject to a set of mission constraints.

The world is represented as a discrete set of cells. For each cell, we wish to determine the probability that an object of interest is in a particular cell given a set of observations. Let S_i^t and O_i^t denote state and observation random variables that encodes whether an object of interest is in cell i at time t. Given a set of N cells, we will move or position the robot such that we gather a series of observations that provide information about all of the cells or some subset of those cells.

Since observations will be taken over time and since objects of interest can move over time, we formulate the problem as a sequential Bayes estimation problem. Given t sequential observations about cell i, our belief that an object of interest is in cell i at time t is given by the following:

$$bel^t(s_i) = P_{S_i^t|O_i^t,O_i^{t-1},\cdots O_i^1}(s_i^t \mid o_i^t, o_i^{t-1} \cdots o_i^1). \tag{2.1}$$

Equation (2.1) is the a posteriori estimate that an object of interest in cell i has been detected given all observations to that point position. Adopting the standard conditional independence assumptions of the Bayes filter [113], the sequential estimate becomes

$$bel^t(s_i) = \alpha P_{O_i^t|S_i^t}(o_i^t \mid s_i)\overline{bel}^t(s_i), \tag{2.2}$$

$$\overline{bel}^t(s_i) = \sum_j \sum_{s_j \in S} [P_{S_i^t|S_j^{t-1}}(s_i \mid s_j) bel^{t-1}(s_j)], \tag{2.3}$$

where $\overline{bel}^t(s_i)$ is the predicted distribution of objects of interest, α is the normalizing constant required by Bayes rule (equal to one divided by the prior predictive distribution), $P_{O_i^t|S_i^t}(o_i \mid s_i)$ is the detection likelihood, and $P_{S_i^t|S_j^{t-1}}(s_i \mid s_j)$ is the model for how objects move in the world.

In this paper, $s = T$ or $s = F$ indicate that the cell contains an object of interest or not. For each cell, we track the belief that an object of interest is in that cell as a function of time. Given a prior belief about objects in the cell, we predict the probability that an object of interest will still be in that cell given (a) the presence or absence of an object in that cell on the previous time step, and (b) the presence or absence of objects in neighboring cells in the previous time step. Thus, Equation (2.3) includes a double summation, one for all cells in the world (the sum over j) and the other over the presence or absence of objects in that cell.

The process of a search task can also be considered as information gathering. From (2.2) and (2.3), we can see that information from observation updates the belief of the search region, which results in uncertainty reduction. We select entropy, which is a commonly used criterion for measuring uncertainty [32], to quantify information collection. It is written as:

$$H(bel^t(s_i)) = -\sum_{s_i \in S} [bel^t(s_i) \log(bel^t(s_i))]. \tag{2.4}$$

2.5 Case Study

Consider a two-dimensional simplified representation of the world and adopt an occupancy grid representation of information in the world. We create a hexagonal tessellation of the world with the dimension of the hexagon determined by the perceptual capabilities of the human. The hexagonal tessellation is useful because it is one in which the distance from the center of one cell to any of its immediate neighbors is constant.

Before exploring the search region, we have no information on this area. We use the entropy of the shared belief to define the uncertainty in equantion (2.4). More formally, we will assume that the prior probability that a cell is occupied by an object of interest is equal to 0.5, which means that the probabilities of the search object in the cell or not are equivalent.

2.5.1 Teammate Model

From the teammate model of human behavior, the wingman robot can predict how the human will move. This yields a sequence of cells that the human plans to traverse, which is denoted by $Y = [y^1; y^2; \cdots y^D]$. Each y^t corresponds to a physical location in the tessellation, so $y^t = i$ means that the human was in cell i at time t.

We adopt a very simple model of agent perception, albeit one based in search theory. The model is that the likelihood of detecting an object of interest in cell i is certain if the human occupies that cell, is zero for cells outside a fixed radius of detection, and is constant for all cells within the radius of detection. Let $N(i)$ denote the set of all cells that are within R units of cell i, in which R defines a radius,

$$N(i) = \{j : j \text{ is no further than } R \text{ cells from } i\}. \tag{2.5}$$

Let $\lambda \in (0, 1)$ be the constant of detection for all cells within $N(i)$. Thus, an agent's probability of detection at position x^t is given by Equation (2.6).

$$P_{O_i^t|S_i^t}(F \mid T) = \begin{cases} 0 & \text{if } i = x^t \\ 1 - \lambda & \text{if } i \in N(x^t) \\ 1 & \text{otherwise} \end{cases} \tag{2.6}$$

By definition, $P_{O_i^t|S_i^t}(T \mid T) = 1 - P_{O_i^t|S_i^t}(F \mid T)$. In (2.6), we assume a search agent can do perfect observation in the cell he is in. However, there exist distinctions on probabilities of detection in the neighbor cells, which come from the difference on agent perception capabilities. To differentiate the observation range, we use $N^{human}(y^t)$ and $N^{robot}(x^t)$ for the set of observed cells by human and robot at time t.

2.5.2 Team Interaction Model

The team interaction model uses the flank support range R^{human}_{flank} to determine the set of cells for the wingman robot motion. This is based on the human's tolerance for how far the robot can wander before being out of position. We use (2.5) to translate R^{human}_{flank} into a set of feasible cells, $N^{human}_{flank}(y^t)$, in which y^t is the human's position. Given a motion range of the wingman robot at a time step, R^{robot}_{motion}, we have

$$\forall y^t \ , \ x_t \in N^{human}_{flank}(y^t) \cap N^{robot}_{motion}(x^{t-1}) \tag{2.7}$$

to define the wingman robot motion dependence on the human motion.

In this paper, we assume that the teammate model provides enough information to estimate the human's path via prediction, Equation (2.6) can be used to determine the posterior probability of likely target location after the human has moved. The posterior from the human is then used as the prior for the robot. In essence, this means that the shared belief about the world passes through two phases: a refinement that comes because the human has moved through the environment and a refinement that comes because the robot is going to move through the environment.

2.5.3 Team Task Model

When the robot plans to fulfill its role for the task model, it assumes that objects of interest do not move, appear, or disappear over time, but this will change in future work. Given this assumption, the prior estimate for the target object's location at time t is equal to the posterior estimate for target object location at time $t-1$. In future work, if the object of interest can move, then a predictive step is required and a full Bayes filter can be applied [113].

Since the human's path has been obtained from the teammate model (the robot is supporting the human), our goal is to control the robot's path to maximize the amount of

information gathered by the human and robot combined. When the robot is at x^t, it will update the beliefs of all the neighbor cells defined by the radius of detection, R^{robot}. We denote the information gain at position x^t as

$$F(x^t) = \sum_{i \in N^{agent}(x^t) \cup x^t} [H(bel_i^{t-1}) - H(bel_i^t)]. \tag{2.8}$$

In (2.8), $H(bel_i^t)$ denotes the entropy of the belief of cell i at time t, and $F(x^t)$ shows the uncertainty reduction at time t from the all the observed cells.

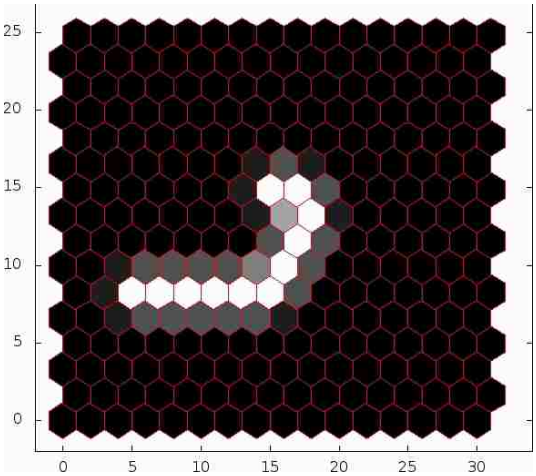
In order to keep synchronization with human motion, a requirement imposed by the interaction and task model, we assume that the robot starts in the same location as the human and intends to plan a time length identical with the predicted human motion length. We set the initial position by $x_0 = y_0$ and the planning time length to D. These constraints yield a natural tree structure for the problem, which forms a tree by finding all "visitable" cells by (2.7). The problem is thus to find the sequence $\mathbf{X} = [x^1; x^2; \cdots x^D]$ of robot positions such that

$$\sum_{t=1}^{D} F(x^t) = H(bel^D) - H(bel^0) \tag{2.9}$$

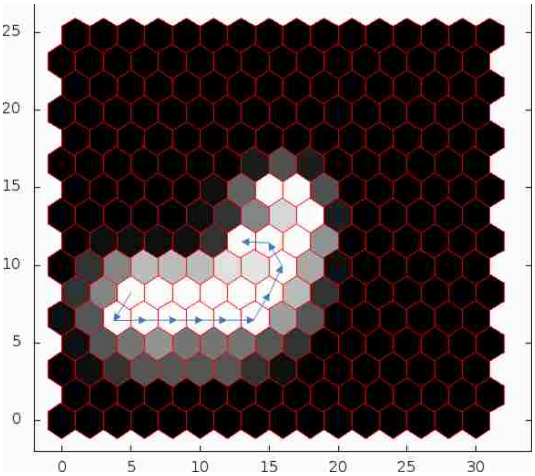
is as large as possible. Performance can either be described as a summation of information gain at each time step or the total information gain reward. Putting this together with the constraint (2.7) from the team interaction model yields the constrained optimization problem for the team task model.

$$\begin{gathered} \max_{x^1 \cdots x^D} \textstyle\sum_{t=1}^{D} F(x^t) \\ \text{subject to } x^t \in N_{flank}^{human}(y^t) \cap N_{motion}^{robot}(x^{t-1}). \end{gathered} \tag{2.10}$$

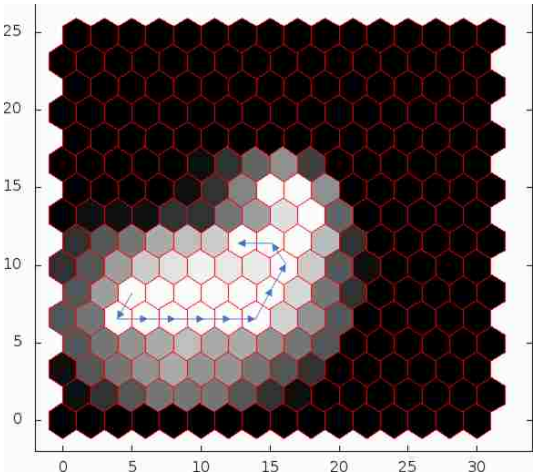
Fig.2.4 shows a case of how the entropy of the shared belief of the search region has been updated by the search of the human-robot team. To visualize the entropy, we color the maximum value in black and minimum value in white, which determines the gray transition



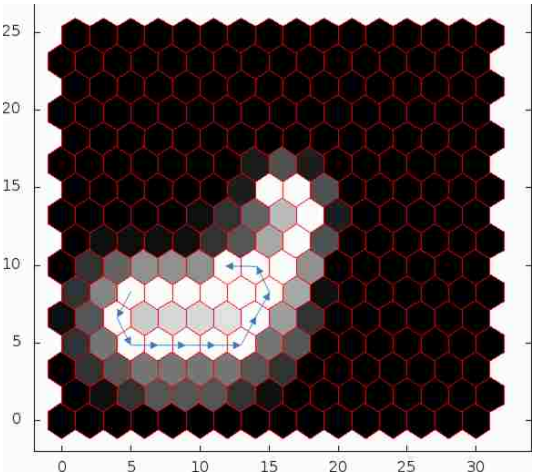
(a) After Human Visited



(b) After Wingman Robot Visited, $FSR = 2$, $ROR = 2$



(c) After Wingman Robot Visited, $FSR = 2$, $ROR = 3$



(d) After Wingman Robot Visited, $FSR = 3$, $ROR = 2$

Figure 2.4: The entropy of shared belief of the search region changed after observation, FSR is short for $FlankSupportRange$, ROR is short for $RobotObservationRange$.

for the values in between. Before the search begins, we assume we have no idea on the location of the search object so that all the cells have been colored black which shows the largest uncertainty. For visited cells, the entropy is reduced to zero. This means the entropy of the shared belief of this cell has been reduced to zero so that this cell has been colored white.

2.5.4 Simulation

Fig.2.4a shows how the entropy of the shared belief of the search region has been changed by a human search agent. The robot's teammate model assumes, via Equation (2.6), that the human has imperfect observation in neighboring cells, so the entropy of the believes on neighboring cells has been reduced less than the visited cells. The value of the gray color is determined by how the observation model is defined.

Based on the human path and how the shared belief of the environment has changed, the robot wingman plans a path to optimize the team information gain, subject to the teammate interaction model via Equation (2.7). Fig.2.4b, 2.4c and 2.4d show the entropy of the shared belief of the search region after the search of the robot wingman within different parameters. We use arrows to label the planned path of the robot wingman. We can see that increasing the observation range of the robot will usually not influence the planned path for the robot wingman, as Fig.2.4b and 2.4c have the same path shapes. However, increasing the flank support range, which gives more motion freedom to the robot wingman, will lead to a new generated path, as shown in Fig.2.4d.

2.6 Conclusion And Future Work

Based on shared mental model and search theory, we model team-based search as an information-based task using a Bayesian approach. A wingman robot has been introduced for this problem, with robot decision algorithms designed to support collaborative human-robot interaction. Using the entropy of the belief of the search region as a way of information

measurement, we illustrate how the robot wingman will do path planning for collaborating with the human as an optimization problem. Using a specific case, we illustrate how the human robot collaboration on a search task will change the entropy of the belief of the search region, which works as a shared model on the environment from the team perspective. Here we only use a depth-first exhaustive search to find the optimal solution for wingman path planning. Future work will be focused on proposing an efficient and applicable solution for wingman path planning. Moreover, we will add more features on modeling the search environment, like obstacles and stochastic dynamics. Finally, we will relate problem modeling assumptions to the requirement of a shared mental model.

Chapter 3

Informative Path Planning with a Human Path Constraint [1]

Abstract

One way for a human and a robot to collaborate on a search task is for the human to specify constraints on the robot's path and then allow the robot to find an optimal path subject to these constraints. This paper presents an anytime solution to the robot's path-planning problem when the human specifies a path constraint and an acceptable amount of deviation from this path. The robot's objective is to maximize information gathered during the search subject to this constraint. We first discretize the path constraint and then convert the resulting problem into a multi-partite graph. Information maximization becomes a submodular orienteering problem on this topology structure. Backtracking is used to generate an efficient heuristic for solving this problem, and an expanding tree is used to facilitate an anytime algorithm.

[1] Published in 2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC). Authors are Daqing Yi, Michael A. Goodrich and Kevin D. Seppi.

3.1 Introduction

In problems that require searching for an object of interest, robots can make human efforts more effective because robots can be more robust to environmental contamination and can sense things beyond of human capabilities. Designing the interactions between a human and a robot for search is no longer constrained to Sheridan's levels of autonomy [106]. Alternatives to Sheridan's levels for a search task include interactions where the human manages the robot either by shaping the information used by the robot to make decisions [70] or by imposing constraints on robot and then allowing the robot to flexibly plan within those constraints [16, 28, 70].

In this paper, we consider constraint-based interactions between a human and a robot for a search task. Rather than consider constraints like no-fly zones or strict waypoints, we explore path constraints imposed by the human and then allow the robot to deviate from the path within some specified tolerance. The source of these constraints range from the robot operating as a "wingman", to a co-located human searcher, to a human telling the robot to approximate the shortest path to an object of interest while gathering maximal information. Furthermore, we assume that the robot's path-planner operates on a discretized representation of the environment and that the robot's sensor footprint covers multiple cells in the discrete representation. Finally, we assume that the robot's sensor becomes less accurate as the distance between the robot and an object of interest grows. These assumptions make the path-planning problem a constrained version of the submodular orienteering problem on a graph.

This paper presents an anytime approximate solution to this problem that uses backtracking to generate an efficient heuristic and an expanding tree. Section 3.3 shows how a multi-partite graph is generated using the human-path constraint and formulates the problem into a class of submodular orienteering on a multi-partite graph. Section 3.4 describes the algorithm in the context of solving the submodular orienteering problem and presents a proof that the algorithm will always find the optimal solution, given enough time. Section 3.5

introduces a robot wingman problem to demonstrate the performance and efficiency of the algorithm.

3.2 Related Work

By modeling the objective of a search task using information measurement, previous work has focused on planning a path for a robot to maximize information gained in a reasonable time, especially in a large problem spaces. In a continuous space, a rapidly-exploring random tree can solve the information maximization path planning problem, and also shows good efficiency in an online optimization [69]. If there exists a temporal logical constraint, a receding horizon planning can be used [56].

If the robot's observation model is a coverage instead of a point, the objective of the path planning becomes *maximum coverage*. Maximum coverage is a classic NP-hard combinatorial optimization problem [80], which includes unignorable overlaps. The total information of a set of observation coverages is measured by mutual information, which implies a property of "nondecreasing submodularity" [108]. A greedy approximation with known performance bound can efficiently exploit the submodularity property of mutual information [108]. A branch and bound approach can also be used in informative path planning [13].

Maximizing the reward collected from a limited-length graph walk is usually known as an *orienteering problem* [118], in which the total reward is a summation of the rewards of visited vertices. If the reward function of a vertex has submodularity as in a *maximum coverage problem*, the problem is defined as a *submodular orienteering problem* [24]. Unfortunately in the submodular orienteering case the location of the robot at time t constrains the reachable locations at time $t + 1$. Thus, naïvely applying a greedy algorithm to the submodular orienteering case, that is, with a "teleport" assumption, yields poor results [64]. For a generalization of the submodular orienteering problem in which the neighboring constraint can be converted into a time budget, a recursive greedy algorithm can be applied [24].

3.3 Problem Statement

In this section we convert a human-path constraint into a multi-partite graph and formulate the informative path planning problem into a class of submodular orienteering on a multi-partite graph.

3.3.1 Information maximization path planning

Consider a discretized map of the world formed by a set of cells $\mathbf{S}$ and suppose that the robot moves with constant speed from a cell to its neighbors. In the search task, each cell in a discretized map is assigned an entropy value to represent the information distribution. Because the robot's path must be connected, the robot's motion is constrained by a graph topology determined by the cell neighborhood. In a period of time of length T, we denote the robot's path as $X = [x_1, x_2, \cdots, x_T]$, and note that this path must satisfy the connection constraint on the discretized map. We adopt an observation coverage model for the robot, which means that the robot can observe not only the cell it currently occupies but also neighboring cells within a given range. Let the observation at time step t be O_t^X, which describes both the observed cells and how well they are observed. Thus the robot's path, X, induces a sequence of observations $\mathbf{O}^X = \{O_1^X, \cdots, O_{T-1}^X, O_T^X\}$.

We assume that the observation coverage model follows Bayes rule. Thus we can define the information gain of the robot using mutual information $I(\mathbf{S} \mid \mathbf{O}^X) = H(\mathbf{S}) - H(\mathbf{S}, \mathbf{O}^X)$. The entropy reduction over the problem space $\mathbf{S}$ by the observation $\mathbf{O}^X$ is the *information gain* to the robot.

3.3.2 Human path constraint

As discussed in the introduction, there are several ways that a human can constrain the robot's path. Without loss of generality, we adopt a "wingman" approach and assume we have a model that can predict the human's path. We denote the human's T-step path as $Y^h = [y_1^h, y_2^h, \cdots, y_T^h]$. We define a neighbor function $N()$ that represents the assumption

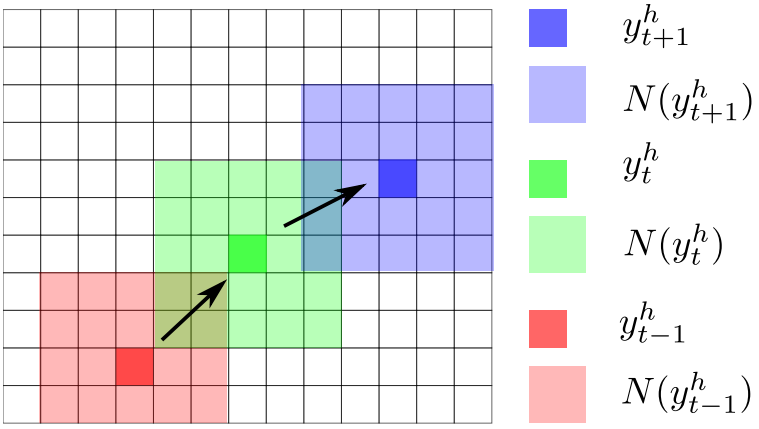


Figure 3.1: How the multi-partite graph is obtained.

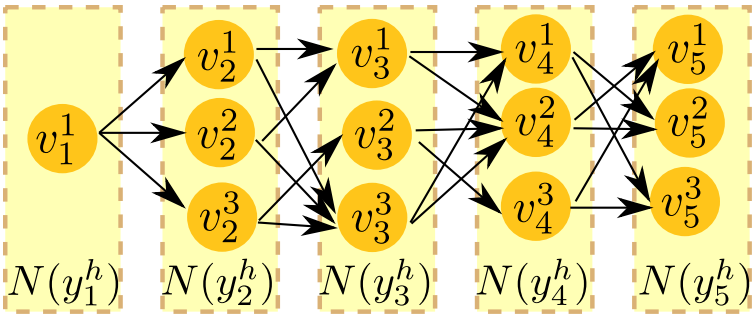


Figure 3.2: A multi-partite graph from a human path constraint.

from the introduction that the robot can deviate from a constrained path by no more than a given tolerance. At each time step, this neighborhood induces the set of *visitable cells* for the robot, which is denoted by $N(y^h_t)$. Figure 3.1 gives an example. By organizing the set of visitable cells at time t into a partition of vertices, we can construct a multi-partite graph $G = (V, E, T)$ from the constrained path. A partition $V(t) \in V$ is obtained from the cells in $N(y^h_t)$. The edge set E is determined by the neighborhood of each cell from the discretized map.

Imposing the path constraint Y^h, we define the multi-partite graph as follows. Figure 3.2 illustrates how the path constraint induces the multi-partite graph for a notional human path. Note that a cell in the discretized map might appear in multiple partitions due to overlaps between sets by $N(y^h_t)$ at different t.

Definition 1 (**Multi-Partite Graph**). *The multi-partite graph $G = (V, E, T)$ is defined as a graph of T partitions. The vertex set V is defined as $V = \cup_{t=1}^{T} V(t)$. Each partition $V(t)$ is a set of vertices v_t^i, where t indicates which partition the vertex x is in and i indicates the index of this vertex. Edges are directed, originating from vertices in set $V(t)$ to vertices in set $V(t+1)$. Let $v_t^i \in V(t)$ and $v_{t+1}^j \in V(t+1)$. A directed edge (v_t^i, v_{t+1}^j) connects vertices v_t^i and v_{t+1}^j.*

In order to guarantee that the search process on a multi-partite graph $G = (V, E, T)$ always ends with a path of length T, we use a pruning process to ensure that each vertex can be reached from the previous partition and is connected to a vertex in the next partition. The pruning process includes a forward pruning and a backward pruning. The forward pruning traverses from partition $V(2)$ to partition $V(T)$ and removes any vertex that has no incoming edge; all edges incident to this vertex are also removed. The backward pruning traverses from partition $V(T-1)$ to partition $V(1)$, and removes any vertex that has no outgoing edge; all edges incident to this vertex are also removed.

3.3.3 The Optimization Problem Model

Without loss of generality, we assume all paths start from the same vertex. Thus, we have only one vertex in partition $V(1)$, as illustrated in Figure 3.2. Because the objective of the path-planning problem is to maximize mutual information, and because mutual information is a submodular function [108], we find it convenient to shift from the bulky notation for mutual information, $I(\mathbf{S}; \mathbf{O}^X)$, to the more concise notation of a general submodular function, $f(X)$. Because the mutual information $I(\mathbf{S}; \mathbf{O}^X)$ is independent of the sequence of the vertices in a path, we write X as a set in $f()$ for simplicity. $f()$ supports multiple vertices representing the same cell, which is like choosing same position multiple times in a sensor coverage placement problem.

The objective of the search task is to maximize information gain subject to the path constraint. Since the path constraint is encoded as the multi-partite graph, we can restate

the objective as maximizing information gain on the multi-partite graph. This yields a submodular orienteering problem on a multi-partite graph $G = (V, E, T)$, which is given as

$$\begin{aligned} Objective: & X^* = \arg\max_X f(X); \\ Constraint: & |X| = T, x_t \in V(t), (x_t, x_{t+1}) \in E. \end{aligned} \tag{3.1}$$

Exhaustive search could find the optimal path but the time-complexity of such a search makes this unacceptable in all problems except in small problems. A greedy search is efficient but the performance of greedy search on a submodular problem is not guaranteed given a topological constraint [64]. Instead of a greedy heuristic, we develop an alternative heuristic based on the property of mutual information.

Mutual information has two desirable properties that we exploit. First, it is independent of the sequence of vertices on a path and, second, it follows a chain rule. This chain rule property can be written as $f(x_1, x_2, \cdots, x_T) = f(x_1) + f(x_2 \mid x_1) + \cdots + f(x_T \mid x_1, \cdots, x_{T-1})$, which yields a structured Bellman-like equation $\hat{x}_t = \arg\max_{X_t}[f(x_t \mid x_1, \cdots, x_{t-1}) + \max_{X_{t+1}, \cdots, X_T} f(x_{t+1}, \cdots, x_T \mid x_1, \cdots, x_t)]$. These structures lead to two key terms: *maximum future reward* and *maximum total reward*.

Definition 2 (**Maximum Future Reward**). *Define the maximum future reward as*

$$h(x_1, \cdots, x_{t'}) = \max_{V(t'+1), \cdots, V(T)} f(x_{t'+1}, \cdots x_T \mid x_1, \cdots, x_{t'}),$$

given the topology constraint $\forall \tau \in \{t'+1, \cdots, T\}, x_\tau \in V(\tau)$ *and* $\forall \tau \in \{t'+2, \cdots, T-1\}, (x_{\tau-1}, x_\tau) \in E$.

Definition 3 (**Maximum Total Reward**). *Define the maximum total reward from choosing* x_t *after* $x_1 \cdots, x_{t'}$ *have been chosen as,* $\forall t > t'$,

$$u(x_t \mid x_1, \cdots, x_{t'}) = f(x_t \mid x_1, \cdots, x_{t'}) + h(x_1, \cdots, x_{t'}, x_t).$$

If we could obtain the values $u(x_t \mid x_1, \cdots, x_{t'})$, then we could greedily chose values for $\hat{x}_t$ as those that maximize $u()$; this would yield an optimal solution as $\hat{x}_t \to x_t$. Unfortunately, the calculation on $u(x_t \mid x_1, \cdots, x_{t'})$ is hard due to the submodularity of $f()$ and the topology constraint. In the next section, we present a heuristic for $u()$ that yields good performance in empirical studies.

3.4 Approximate Anytime Solution

In this section, we use backtracking to estimate the maximum total reward and use this estimate as our search heuristic. We then use an expanding tree to create an anytime algorithm approximate solution to the submodular orienteering problem on the multi-partite graph.

3.4.1 Using the Search Heuristic from Backtracking

In a graph search process, if a sub-path $\{v_1, \cdots, v_{t'}\}$ has been visited, we can use the following property to approximate the maximum future reward.

Property 1.

$$h(x_1, \cdots x_{t'}) = \max_{x_{t'+1} \in V(t'+1) \wedge (v_{t'}, x_{t'+1}) \in E} u(x_{t'+1} \mid x_1, \cdots x_{t'}).$$

Property 1 implies that the maximum future reward at partition $V(t)$ can be estimated from the maximum total reward at partition $V(t+1)$. This means that the maximum total rewards could be estimated by using a backtracking process. We propose a backtracking process in Algorithm 1.

Algorithm 1 estimates the maximum total rewards of the vertices in $V(t'+1)$. The backtracking starts at partition $V(T)$ and goes back to $V(t'+1)$ in order to propagate the estimated maximum future rewards. For a vertex $v(t)$, the maximum future reward is estimated based on the estimated maximum total rewards of all the connected vertices in

Algorithm 1 BT$(\{v_1, \cdots, v_{t'}\}, G)$ - Backtracking

Input: a sub-path $\{v_1, \cdots, v_{t'}\}$, and multi-partite graph $G = (V, E, T)$
Output: $\hat{u}(v_1, \cdots, v_{t'}, v_{t'+1}), \forall v_{t'+1} \in V(t'+1)$

```
1:
2: for t = T : −1 : t' + 1 do
3:     for v_t ∈ V(t) do
4:         if t == T then
5:             û(v_T | v_1, ··· , v_t') = f(v_T | v_1, ··· , v_t')
6:         else
7:             ĥ(v_1, ··· , v_t', v_t) = max_{x_{t+1} ∈ V(t+1) ∧ (v_t, x_{t+1}) ∈ E} û(x_{t+1} | v_1, ··· , v_t')
8:             û(v_t | v_1, ··· , v_t') = f(v_t | v_1, ··· , v_t') + ĥ(v_1, ··· , v_t', v_t)
   return û(v_1, ··· , v_t', v_{t'+1}), ∀v_{t'+1} ∈ V(t'+1)
```

partition $V(t+1)$. The estimated total reward is then obtained by adding the estimated instant reward of $v(t)$ with the estimated maximum future reward of $v(t)$. The backtracking process in Algorithm 1 satisfies Lemma 1.

Lemma 1. *"Backtracking" in Algorithm 1 never underestimates the maximum total reward, which means*

$$\forall t \geq t', \hat{u}(x_t \mid v_1, \cdots, v_{t'}) \geq u(x_t \mid v_1, \cdots, v_{t'}). \tag{3.2}$$

Proof. The proof is given in Appendix 3.6. □

Note that Lemma 1 holds even when there are multiple vertices in a multi-partite graph generated from same cell. This is because the submodularity of $f()$ is preserved and the proof depends primarily on submodularity. However, multiple vertices generated from same cell in a path increase the degree to which the reward is overestimated.

3.4.2 Expanding Tree Search

Since the heuristic is not guaranteed to produce an optimal solution, we create an anytime algorithm that allows us to continue the search process until a time limit is exceeded or the search is completed exhaustively. In order to track the anytime search process, the algorithm uses an expanding tree. The expanding tree is the tree produced by repeated depth-first traversals of the multi-partite graph [96].

Definition 4 (**Expanding Tree**). *An expanding tree $G_T = (N, L, T)$ obtained from a multi-partite graph $G = (V, E, T)$ is the tree produced by a depth first traversal of G. T is the depth of the tree, which is determined by the number of partitions in a multi-partite graph G. N is the node set. Each $n_t^{i(j)} \in N$ indicates the relevant vertex in the multi-partite graph G, in which t shows the index of the time partition, i shows the index of the corresponding vertex from within that partition and (j) shows the index of a vertex in $V(t-1)$ that has an out edge to vertex i. L is the directed link set. $(n_t^{i(k)}, n_{t+1}^{j(i)}) \in L$ is determined by $(v_t^i, v_{t+1}^j) \in E$.*

We assign the *type* to each node, which are *New* (a node has been created but not expanded), *Expanded* (a node that has all child nodes created) and *Frozen* (a node that has been created but will not be expanded). Each path in an expanding tree is derived from a unique depth-first traversal of the corresponding mutli-partite graph. We use $v(n_{t+1}^{j(i)})$ to denote a vertex mapped from a node. For a node $n_t^{i(j)}$, we use $path(n_t^{i(j)})$ to denote the implicit path from the start position to the corresponding vertex of the multi-partite graph, the cardinality of which is t.

We can now present Algorithm 2 for a single search iteration. It is used as one run in the anytime framework.

Algorithm 2 NERB$(n_{t'}, G, G_T)$ - Node Expanding with Recursive Backtracking

Input: Expanding Node $n_{t'}$, Multi-partite graph $G = (V, E, T)$, Expanding tree $G_T = (N, L, T)$

Output: *solution* of a complete path

1: $solution = path(n_{t'})$
2: **for** $t = t' : 1 : T - 1$ **do**
3: Create all $child(n_{t'}) = \{n_{t'+1} \mid v(n_{t'+1}) \in V(t'+1) \wedge (v(n_{t'}), v(n_{t'+1})) \in E\}$
4: Add $child(n_{t'})$ as the child nodes of $n_{t'}$
5: $n_{t'}.state = Expanded$
6: $\hat{u}(v_{t'+1} \mid path(n_{t'})) = \mathbf{BT}(path(n_{t'}), G)$
7: $\hat{n}_{t'+1} = \arg\max_{n_{t'+1} \in child(n_{t'})} \hat{u}(n_{t'+1} \mid path(n_{t'}))$
8: $solution = solution \bigcup \{\hat{n}_{t'+1}\}$
return *solution*

3.4.3 Adding node freeze to the expanding tree

Because Lemma 1 tells us that the backtracking process never underestimates the maximum total reward of a node, we can use the estimated maximum total reward of a node to evaluate whether the node might lead to a path that returns a bigger reward than the current best one. A node is not in a path that has bigger reward if its estimated value is smaller than the current best solution. We can freeze this node, which means that we are not going to expand any of its descendant nodes. At each iteration we find a new solution, we can call a node freeze process to update the states of the nodes in the expanding tree. This process is given in Algorithm 3.

Algorithm 3 NF(G_T, θ^*) - Node Freeze

Input: an expanding tree $G_T = (N, L, T)$, the reward of found maximum reward path θ^*

1: **for** $n_t \in N$ **And** $n_t.state == New$ **do**
2: **if** $f(path(n_t)) + \hat{h}(path(n_t)) \leq \theta^*$ **then**
3: $n_t.state = Frozen$

Algorithm 4 combines Algorithm 1, Algorithm 2, and Algorithm 3 to yield the anytime algorithm. The expanding tree starts with just a root node, which is the start vertex. When a node is created, the state of the node is *New*. Expanding a node in Algorithm 4 means creating all of its children nodes and changing the state of this node to *Expanded*. When a child node is created, the estimated maximum total reward is calculated using Algorithm 1 and stored. Each run of Algorithm 2 returns a complete path as a solution. When a new complete path has been returned, the freeze process defined in Algorithm 3 is executed by checking estimated maximum total rewards stored in each nodes in the state of *New*. The next run of the search starts from the *N*ew node n_t that has the largest estimated reward $f(path(n_t)) + \hat{h}(path(n_t))$. Starting from this node, the next call to Algorithm 2 generates another complete path. This anytime algorithm stops at a pre-specified number of iterations or when there is no *New* node remaining.

Algorithm 4 is optimal as shown in Theorem 1 given here.

Algorithm 4 Anytime Algorithm Framework

Input: Expanding Tree $G_T = (N, L, T)$, and multi-partite graph $G = (V, E, T)$;

1: Initial expanding tree $G_t(N, L, T)$ with v_1 as root node
2: $maxPath = NULL$, $newPath = NULL$
3: $n' = G_T.root$
4: **while** $n'! = NULL$ **do**
5: $\quad newPath = \mathbf{NERB}(n', G, G_T)$
6: $\quad$ **if** $(f(newPath) > f(maxPath))$ **then**
7: $\quad\quad maxPath = newPath$
8: $\quad\quad$ post $maxPath$
9: $\quad \mathbf{NF}(G_T, f(maxPath))$
10: $\quad n' = \arg\max_{\{n|n \in N \wedge n.state == New\}}(f(path(n)) + \hat{h}(path(n)))$

Theorem 1. *The anytime algorithm framework in Algorithm 4 can always find an optimal solution given enough time.*

Proof. The proof is by contradiction and is similar in spirit to the proof of optimality for the well-known A* algorithm. Since Algorithm 4 keeps expanding until no *New* nodes remain, as long as any node in the optimal path will never be frozen, the search will reach the optimal terminal node.

Assume that one of the nodes n_t^* in the optimal path can be frozen. This means that $f(path(n_t^*) + \hat{h}(path(n_t^*))) \leq f(path(n_T'))$, in which n_T' is another terminal node but not the terminal node for the optimal path. As a result, when n_T' has been reached, it will freeze node n_t^*.

However, since node n_t^* is in a path to an optimal terminal node, $f(path(n_t^*)) + h(path(n_t^*)) > f(path(n_T'))$. Also we have $f(path(n_t^*)) + \hat{h}(path(n_t^*)) \geq f(path(n_t^*)) + h(path(n_t^*))$ by Lemma 1. Thus we have $f(path(n_t^*)) + \hat{h}(path(n_t^*) > f(path(n_T'))$, which is a contradiction.

Therefore, a node in a path to an optimal terminal node will never be frozen by any non-optimal terminal node. □

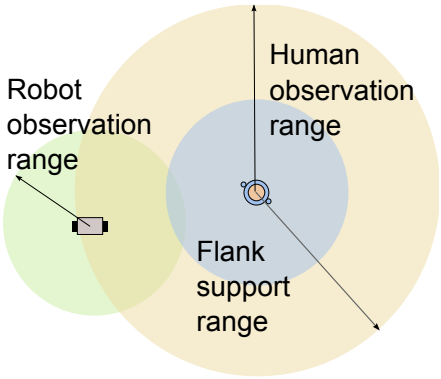


Figure 3.3: A Robot Wingman Framework.

3.5 An Application to Robot Wingman

In this section, we apply Algorithm 4 to the robot wingman problem [42]. Let R_{flank} denote the *flank support range*, which determines the area that a robot wingman is expected to stay in when a human is moving; this is illustrated in Figure 3.3. The robot wingman constraint requires that $\forall t, ||x_t - y_t^h|| \leq R_{flank}$ or, equivalently, $\forall t \in T, x_t \in N(y_t^h)$.

Consider a two-dimension search space discretized into a world of hexagonal cells. This discretization gives a constant distance from the center of one cell to any of its immediate neighbors, which facilitates modeling the agent observation range. Moreover, a hexagonal tessellation is consistent with the assumption that we made that the robot would move at constant speed from one cell to another; in a hexagonal tessellation, the distances between the centers of all neighboring cells and the current cell is constant.

The observation model of an agent uses the likelihood of detecting an object of interest in cell i and follows Bayes rule in updating the posterior [42]. Since we know the human's path, Y^h, we can use the human's observation model to predict what the human could observe and update the prior distribution of information to reflect this. The simulation results we present assume that human observations have already been factored into the prior distribution of information.

3.5.1 Performance

We simulate the use of the algorithm in a search space in which the entropy of each cell is randomly generated. The simulation results are aggregated from 20 runs of each case. The parameters $R_{flank} = 2$ is used for the neighboring function of the human path constraint and $R_{obs}^{robot} = 2$ is used for the robot's observation range.

We use a "fully expanded tree size" to measure the *problem size*, which depends on the planning length and the vertex connectivities. Due to the human constraint, the planning length is determined by the human's path length. The performance of the heuristic is measured by the *percentage of optimal at first iteration*, that is a percentage computed from the value obtain in the first iteration of Algorithm 4 over the optimal value. High values of this metric indicate that the heuristic is useful. The greedy heuristic [108], which chooses the maximum next step, is imported to compare with the backtracking heuristic. Figure 3.4 shows the comparison between two types of heuristic on the percentages of the optimal as a function of planning lengths. We can see that the performance of the backtracking heuristic significantly surpasses that of the greedy heuristic.

Naturally, as the size of the search space expands, the difficulty in finding the optimal solution using an exhaustive search grows. Since we want to understand how well our anytime algorithm performs for problems that are too big to search exhaustively, we bound the payoff for the optimal path by using a "teleport" search in which the robot can bounce from region to region without following a connected path. Figure 3.5 shows the rewards collected using the path produced by the backtracking heuristic normalized by the rewards collected by the"teleport" path for large search spaces. Again, the backtracking heuristic is much better than the greedy solution (similarly normalized).

We use *percentage of nodes explored* to indicate the efficiency of the anytime algorithm framework. In particular, we are interested in whether freezing nodes improves search efficiency. Figure 3.6 shows that the percentage of nodes explored decreases significantly when the problem size is expanded. Since the anytime algorithm becomes an exhaustive

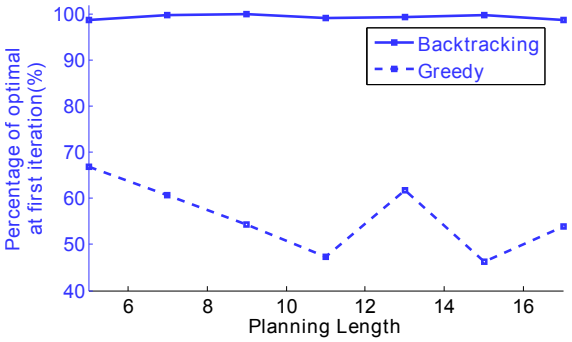


Figure 3.4: The performance comparison between the backtracking heuristic and the greedy heuristic.

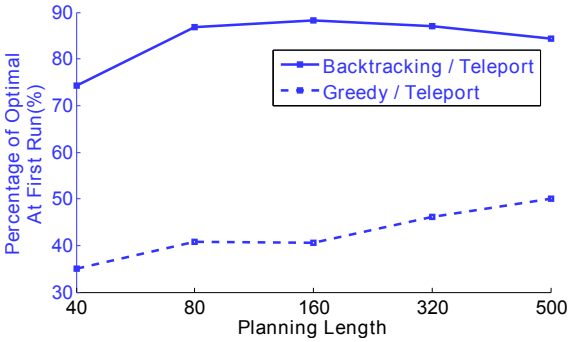


Figure 3.5: The performance comparison between the backtracking heuristic and the greedy heuristic in a large problem space.

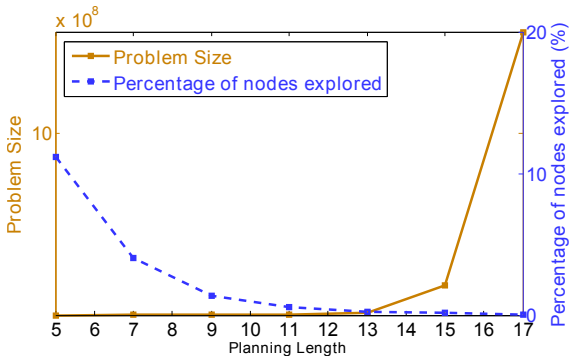


Figure 3.6: Problem size and exploration ration with different planning lengths.

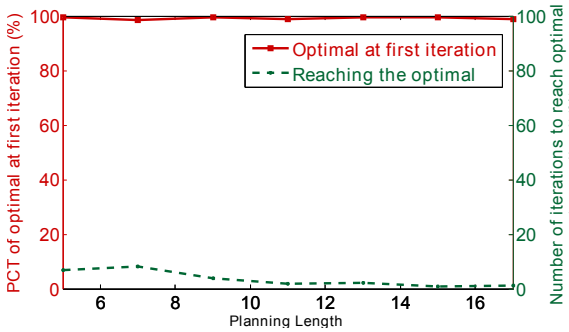


Figure 3.7: Percentages of optimal at first iteration and number of iterations reaching optimal with different planning lengths.

search in the absence of freezing nodes (and hence follows the size of the search space in the figure), this figure indicates that the percentage of nodes expanded is significantly decreased by freezing nodes. In the anytime algorithm, the exploration might not stop when the optimal is found, due to the existence of overestimation. If current best of a search can reach the optimal very quickly, it means that a best solution found in a fixed time has high probability of being optimal. We use *number of iterations reaching optimal (normalized)* to measure this optimal search capability of Algorithm 4. Figure 3.7 illustrates that the anytime algorithm can find the optimal solution relatively quickly.

3.5.2 Robustness

We extend the search environment from *random* to *uniform* and *multimodal*. Uniform indicates that the entropies in different cells are identical, and multimodal indicates that the

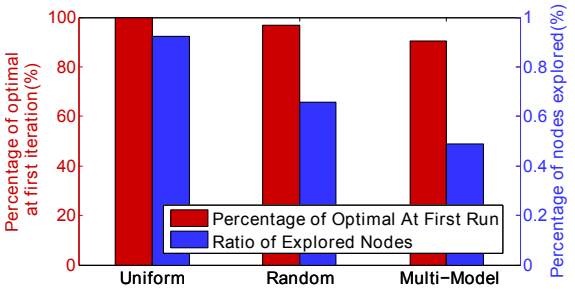
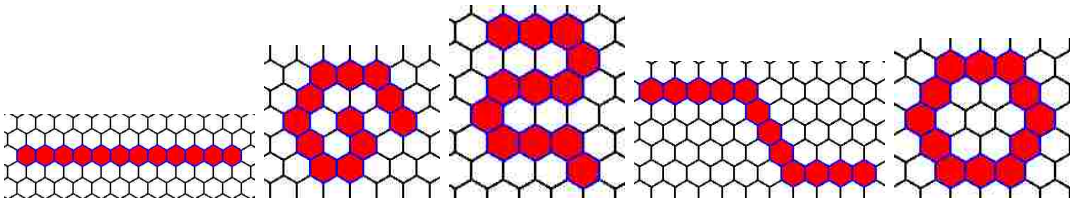


Figure 3.8: Performance in different types of environments.



(a) Line. (b) Spiral. (c) Lawn-mower. (d) Arc. (e) Loitering.

Figure 3.9: Different patterns of human path.

entropy distribution among cells is a multi-modal spatial distribution. Figure 3.8 shows that Algorithm 4 consistently performs well in different types of search environment.

In order to illustrate how well Algorithm 4 adapts to different human path constraints, we introduce five common patterns of paths executed by a human in a search task, which are *line*, *spiral*, *lawn-mower*, *arc* and *loitering*. Figure 3.9 shows examples on these five patterns. Due to the wingman constraint, different human paths lead to different problem sizes and different ratios of overlap in the coverage at two different time steps. For this comparison we hold the number of time steps fixed at 11 over different patterns as in Figure 3.9.

Figure 3.10 shows that the problem size varies significantly depending on the type of path, though the planning length is identical. Interestingly, Algorithm 4 shows better efficiency in larger problem size. In Figure 3.11, we can see that the ratios of explored nodes are relatively smaller in the patterns of "spiral", "lawn-mower" and "loitering", in which the problem sizes are relatively larger in Figure 3.10. We can see that the percentage of optimal at first iteration are all close to the optimum in all the patterns in Figure 3.12, which implies the goodness of the backtracking heuristic.

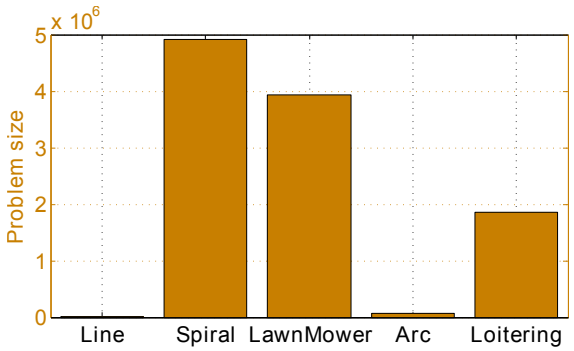


Figure 3.10: Problem size in different patterns of human paths.

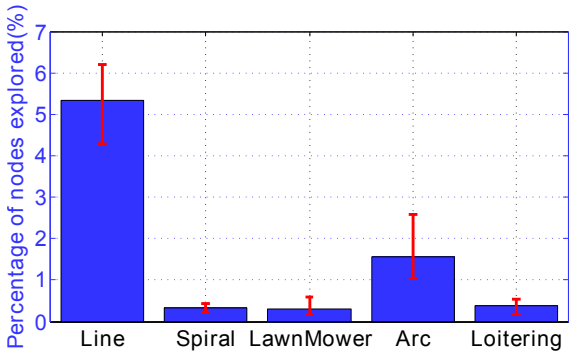


Figure 3.11: Exploration ratios in different patterns of human paths.

Figure 3.12: Percentages of optimal at first iteration in different patterns of human paths.

3.6 Summary

In this paper, we use a human path to form a path constraint and seek to maximize the information gathered by a robot gathered in a search task. The resulting information maximization path planning is identified as a constrained submodular orienteering problem on a multi-partite graph. We present an anytime algorithm that used a planning heuristic based on backtracking to efficiently find a high quality path. We use a node freeze process to avoid an exhaustive search, yet we prove that this process always preserves the ability of the algorithm to find an optimal solution. We have also shown empirically that this approach substantially reduces the complexity of the resulting search.

Appendix

Proof of a Useful Property

The following property is necessary to prove Lemma 1.

Property 2.

$$\begin{aligned} u(x_t \,|x_1, \cdots, x_{t'}) &= f(x_t \mid \tilde{X}(x_t), x_1, \cdots, x_{t'}) \\ &+ \max_{x_{t+1} \in V(t+1) \wedge (x_t, x_{t+1}) \in E} u(x_{t+1} \mid x_1, \cdots, x_{t'}), \end{aligned} \tag{3.3}$$

in which

$$\tilde{X}(x_t) = \arg \max_{V(t+1) \cdots V(T)} f(x_{t+1} \cdots x_T \mid x_1, \cdots, x_{t'}) \tag{3.4}$$

subject to the constraint

$$\forall \tau \in \{t+1, \cdots, T\}, x_\tau \in V(\tau) \wedge (x_{\tau-1}, x_\tau) \in E. \tag{3.5}$$

Proof. By chain rule, we have $u(x_t \mid x_1, \cdots, x_{t'}) = f(\tilde{X}(x_t) \mid x_1, \cdots, x_{t'}) + f(x_t \mid x_1, \cdots, x_{t'}, \tilde{X}(x_t))$. By decomposing the constraint in (3.5) into $x_{t+1} \in V(t+1) \wedge (x_t, x_{t+1}) \in E$ and $\forall t'' \in [t+2, T], x(t'') \in V(t'') \wedge (x_{t''-1}, x_{t''}) \in E$, equation (3.4) can be $f(\tilde{X}(x_t) \mid x_1, \cdots, x_{t'}) =$

$\max_{V_{t+1}} u(x_{t+1} \mid x_1, \cdots, x_{t'})$ subject to the constraint $x_{t+1} \in V(t+1) \wedge (x_t, x_{t+1}) \in E$. Thus equation (3.3) can be obtained. □

Proof of Lemma 1

Proof. Equation (3.2) can be proven using induction as follows. We have following two propositions, corresponding to the *basis case* and *induction step*, which are

- **proposition 1** $\forall x_T \in V(T), \hat{u}(x_T \mid v_1, \cdots, v_{t'}) = u(x_T \mid v_1, \cdots, v_{t'})$;
- **proposition 2** If $\forall x_{t+1} \in V(t+1), \hat{u}(x_{t+1} \mid v_1, \cdots, v_{t'}) \geq u(x_{t+1} \mid v_1, \cdots, v_{t'})$, then $\forall x_t \in V(t), \hat{u}(x_t \mid v_1, \cdots, v_{t'}) \geq u(x_t \mid v_1, \cdots, v_{t'})$.

Basis: At time T, we have $u(x_T \mid v_1, \cdots, v_{t'}) = f(x_T \mid v_1, \cdots, v_{t'})$ and $\hat{u}(x_T \mid v_1, \cdots, v_{t'}) = f(x_T \mid v_1, \cdots, v_{t'})$. Thus proposition 1 is true.

Induction Step: The definition of $u(x_t \mid v_1, \cdots, v_{t'})$, Property 2, and the definition of $\hat{u}(x_t \mid v_1, \cdots, v_{t'})$ in Algorithm 1, imply

$$\begin{aligned}\hat{u}(x_t \mid v_1, \cdots, v_{t'}) - u(x_t \mid v_1, \cdots, v_{t'}) = [f(x_t \mid v_1, \cdots, v_{t'}) - f(x_t \mid v_1, \cdots, v_{t'}, \tilde{x}_{t+1}, \cdots \tilde{x}_T)] \\ + [\max_{x_{t+1} \in V(t+1) \wedge (x_t, x_{t+1}) \in E} \hat{u}(x_{t+1} \mid v_1, \cdots, v_{t'}) - \max_{x_{t+1} \in V(t+1) \wedge (x_t, x_{t+1}) \in E} u(x_{t+1} \mid v_1, \cdots, v_{t'})].\end{aligned} \tag{3.6}$$

By submodularity, we know that $f(x_t \mid v_1, \cdots, v_{t'}) - f(x_t \mid v_1, \cdots, v_{t'}, \tilde{x}_{t+1}, \cdots \tilde{x}_T) \geq 0$.

Define the following two values

$$\begin{aligned} x^a_{t+1} &= \arg \max_{x_{t+1} \in V(t+1) \wedge (x_t, x_{t+1}) \in E} \hat{u}(x_{t+1} \mid v_{t'}, \cdots, v_1) \\ x^b_{t+1} &= \arg \max_{x_{t+1} \in V(t+1) \wedge (x_t, x_{t+1}) \in E} u(x_{t+1} \mid v_1, \cdots, v_{t'}). \end{aligned}$$

Both x^a_{t+1} and x^b_{t+1} belong to the set of vertices that satisfy the constraint $x_{t+1} \in V(t+1) \wedge (x_t, x_{t+1}) \in E$. Since x^a_{t+1} is the answer to $\arg\max \hat{u}(\cdot)$, we have $\hat{u}(x^a_{t+1} \mid v_1, \cdots, v_{t'}) \geq \hat{u}(x^b_{t+1} \mid v_1, \cdots, v_{t'})$. By the induction hypothesis, $\hat{u}(x^b_{t+1} \mid v_1, \cdots, v_{t'}) \geq u(x^b_{t+1} \mid v_1, \cdots, v_{t'})$. By transitivity, we have $\hat{u}(x^a_{t+1} \mid v_1, \cdots, v_{t'}) \geq u(x^b_{t+1} \mid v_1, \cdots, v_{t'})$. By the definitions x^a_{t+1} and x^b_{t+1},

which equals to $\max_{x_{t+1} \in V(t+1) \wedge (x_t, x_{t+1}) \in E} \hat{u}(x_{t+1} \mid v_1, \cdots, v_{t'}) - \max_{x_{t+1} \in V(t+1) \wedge (x_t, x_{t+1}) \in E} u(x_{t+1} \mid v_1, \cdots, v_{t'}) \geq 0$. Thus proposition 2 is true.

Conclusion: Since the basis case and induction step are true, Equation (3.2) follows.

□

Chapter 4

Supporting Task-oriented Collaboration in Human-Robot Teams using Semantic-based Path Planning [1]

Abstract

Improvements in robot autonomy is changing the human-robot interaction from low-level manipulation to high-level task-based collaboration. For a task-oriented collaboration, a human assigns sub-tasks to robot team members. In this paper, we consider task-oriented collaboration of humans and robots in a cordon and search problem. We focus on a path-planning framework with natural language input. By the semantic elements in a shared mental model, a natural language command can be converted into optimization objectives. We import multi-objective optimization to facilitate modeling the "adverb" elements in natural language commands. Finally, human interactions are involved in the optimization search process in order to guarantee that the found solution correctly reflects the human's intent.

[1]Published in Proc. SPIE 9084, Unmanned Systems Technology XVI, 90840D. Authors are Daqing Yi and Michael A. Goodrich.

4.1 Introduction

As robot sensing, perception and decision-making improves, the human's role in human-robot interaction progressively shifts from teleoperator to supervisor to teammate [103]. This shift toward human-robot teaming means that the relationships between the human and the robot in a team to execute tasks together appear more and more to be collaboration. Although there exists asymmetries between the capabilities and properties of a human and a robot, concepts from human-human teaming can be useful and important. Specifically, the concept of a shared mental model, which originates from the theories of human collaboration, has been applied to analyze the collaborative process between humans and robots as well [68].

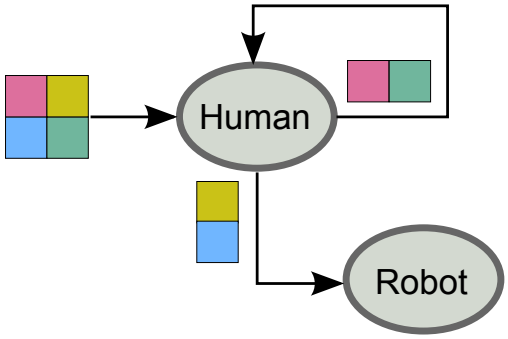


Figure 4.1: An example of task-oriented human-robot collaboration.

Consider a problem where team-wide collaboration is driven by a task shared by the team members. The collaboration can be viewed as a parallel performance of the sub-tasks by different agents. This type of collaboration can be modeled as a task decomposition. In this type of models, the team supervisor takes the responsibility of task decompositions and distributes the sub-tasks to different team members. In a human-robot team, the team supervisor is usually a human. Figure 4.1 illustrates an example on task decomposition and allocation. A human supervisor decomposes the task into sub-tasks and then assign some to human team members and some to robot team members.

This type of relationship indicates the importance of the communication between a human and a robot. Effective communication determines the execution efficiency and the correctness of the outcome. One of the functions of the shared mental model is to facilitate the

mutual understanding and ground communication among the team members. Of particular importance is how shared mental models enable a more collaborative approach to problem solving, facilitated by communications that operate at a higher, more tactical or strategic level of abstraction. A complex process of parameter setup for a human to define a sub-task will reduce the team performance in a cordon and search task [42].

In a cordon and search problem, a robot could be assigned to "screen" a sub-region that is not easily accessible to a human. Instead of relying on teleoperation, the robot now can be autonomous enough to execute the sub-task alone. In the collaborative perspective, the human only needs to express the requirements of the sub-task and provide information that the robot needs in sub-task execution. In this paper, we assume that the team supervisor assigns the sub-tasks using a verbal command. More specifically, we assume that the human supervisor issues directives to a robot. Furthermore, we assume that from a small set of possible commands, the directives are grounded using spatial references that specify key locations for performing specific sub-tasks. Finally, we assume that the directives are associated with a small number of adverbial modifiers that provide qualitative information about intent.

This does not require the robot to understand perfectly what the human said, but rather to model the verbal command. The semantic elements in natural languages are often extracted and formed into a graphical model, Bayesian inference can be applied to infer the meaning of a sentence [112]. A learning process can also be imported to tune the likelihood of the primitives by using an HMM [79]. For tasks like cordon and search, plenty of the elements in a sentence depend on the spatial information in the workspace. Thus, spatial labeling is imported to help convert a human's command into robot's navigation primitives [63]. This enables robot motion planning to be generated by a semantic interpreter [40]. In this paper, we are interested in modeling and solving a robot path-planning problem from a human's verbal command in a framework of task-oriented human-robot collaboration.

Current technologies can already support the language parsing process. A semantic structure is usually composed of the key elements of a sentence, like a noun, verb and etc. Ignoring some other elements leads to information loss. When researchers consider adverbial cues in human languages, an adverbial modifier may be modeled as belief revision [19]. We notice that a verbal command contains information not only what to do but also how to do. It means that we should extract the search objectives and constraints from a verbal command to obtain the criteria. The criteria evaluates the performance of a sub-task execution.

In this paper, we propose a framework to support the path-planning problem from a verbal command. Using the idea of a shared mental model, we import a semantic labeling process to generate a semantic model of the workspace in Section 4.2. The labeled elements can be used to help the teammate interaction and task execution. Specifically, we show how an optimization problem can be created by translating a verbal command. We are interested in extracting information, like adverb elements, to create the multi-objective optimization in Section 4.3. We propose an interactive method to find the optimal solution of a path-planning problem. In Section 4.4, we propose the system framework and the solutions on the robot path-planning.

4.2 Semantic labeling and command

A shared mental model provides a common ground among the teammates of the collaborative process. When the interaction between a human and a robot is based on natural language, the semantic objects must be shared by the team members so that the teammates can understand each other. In a cordon and search task, the information depends greatly on the semantic labels of spatial objects. It is natural to introduce a labeling process to generate the semantic objects on the map of the workspace. These *semantic labels* will be used in sub-task definitions and teammate interactions in the shared mental model. We also need a *task grammar* to define the sub-tasks and organize the semantic elements. Because terms and sentences usually imply different meanings in different types of sub-tasks, a task grammar

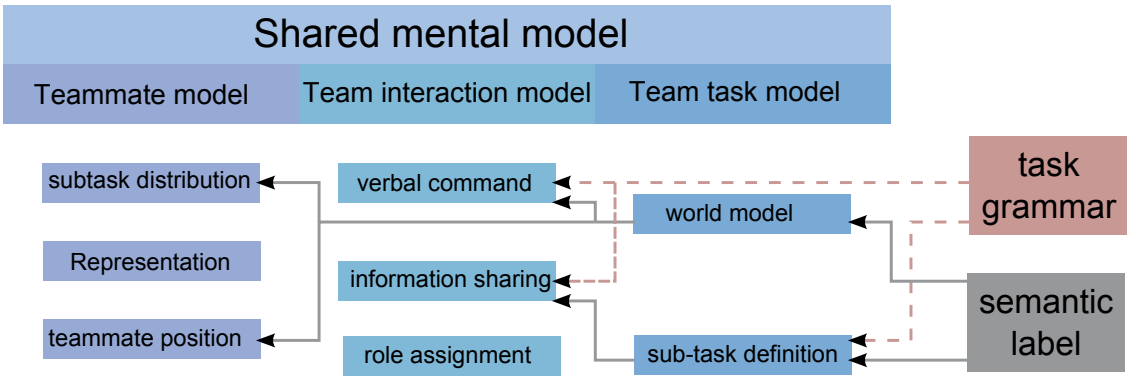


Figure 4.2: How task grammar and semantic labels support a shared mental model.

could help the teammates understand the purposes of each other correctly. In this way, a verbal command can be viewed as an action with logical constraints on a set of semantic elements. The form is determined by the task grammar.

A shared mental model of a cordon and search team can be decomposed [42] into three sub-models. We list only some elements that are relevant with a cordon and search task as following.

- **A teammate model** provides the knowledge of teammates skills, abilities and tendencies.
  - A *sub-task distribution* indicates how the sub-tasks are assigned to team members.
  - The *teammate positions* indicate the positions of the teammate, localized relative to the spatial semantic objects.
  - The *representation* indicates how the teammate encodes information and problems.
- **A team interaction model** provides the knowledge of roles, responsibilities, information sources, communication channels and role interdependencies.
  - A *role assignment* defines the roles of the members in a team.
  - A *verbal command* describes the format of a command.
  - An *information sharing* represents the information exchange format between team members.

A team task model provides the knowledge of procedures, equipment, situations, constraints.

  - A *semantic world model* is a representation of the workspace with semantic labels.
  - A *sub-task definition* defines the sub-tasks and its objectives.

Figure 4.2 illustrates how these elements in a shared mental model depend on the task grammar and the semantic labels. The task grammar and semantic labels support the world model and the sub-task definitions. The world model and the sub-task definitions can then be used to generate verbal commands and help information sharing.

Consider the cordon and search for a human-robot team in an urban area. The labeling process is run before the cordon and search starts. Semantic labels are assigned to the sub-regions and the objects in the map. Supplementary information can be attached to expand the support to different tasks. We also expect that semantic labels are used to support a more flexible grammar. We categorize the labels into three types.

- **Indoor** The "indoor" label defines the region of an indoor environment in the search space.
- **Outdoor** The "outdoor" label defines the region of an outdoor environment in the search space. There are several sub-types on an outdoor label. For the purpose of this paper, we consider only three.
  - *market*: The "market" label usually defines sources of information, where there is high probability of interested events occurring.
  - *unknown*: The "unknown" label indicates lack of prior knowledge. This implies that these regions have potential risks.
  - *normal*: We usually consider unlabeled regions as "normal" by default, which indicate that there is nothing special to be noticed in these regions.

- **Feature** The "feature" label defines the objects in the search space. They can be used for objects of interest, location indicators and etc. This will be used for There are two types of feature labels, which are
  - *2D*: "2D" label defines the objects on the ground.
  - *2.5D*: "2.5D" label defines the objects on the walls of the architectures.

Figure 4.3 illustrates possible semantic labels for a notional world.

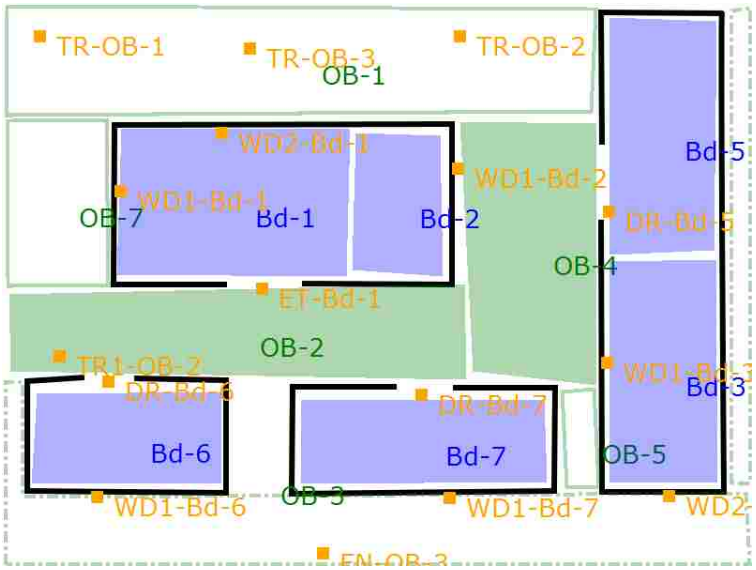


Figure 4.3: A labeled map of an urban environment.

Given the semantic labels of a spatial world model, we can define a task grammar by the characteristics of the sub-tasks. We assume a task grammar that specifies a task, one or more constraints, and one or more adverbs that specify how the task should be performed or how the constraints should be managed. Equation 4.1 is an example of a task grammar that

is used in a cordon and search task.

$$
\begin{aligned}
< Start > &\rightarrow < CommandPhrase > \\
< CommandPhrase > &\rightarrow < ScreenCommand > | < ProceedCommand > \\
< ScreenCommand > &\rightarrow < Adverb > \text{ Screen the } < FeatureQuantifier > \\
&\quad < Feature > \text{ of the } < BlockId > \\
< ProceedCommand > &\rightarrow \text{Proceed } < Adverb > < PrepPhrase > \text{ to the} \\
&\quad < FeatureQuantifier > < Feature > \text{ of the } < BlockId > \\
< Adverb > &\rightarrow \text{covertly } | \text{ safely } | \text{ quickly } | \text{carefully} \\
< FeautreQuantifier > &\rightarrow \text{back } | \text{ front } | \text{ side} \\
< Feature > &\rightarrow \text{door } | \text{ window } | \text{ exits} \\
< BlockType > &\rightarrow \text{BD- } | \text{ OB-} \\
< PrePhrase > &\rightarrow < PrepSegment > \text{ the } < BlockId > | \text{ between the} \\
&\quad < BlockId > \text{ and the } < BlockId > \\
< BlockId > &\rightarrow < BlockType > < Id > \\
< PrepSegment > &\rightarrow \text{around } | \text{ left-of } | \text{ right-of}
\end{aligned}
\tag{4.1}
$$

For example, if a human tells a robot to "carefully screen the OB-2", this command defines a sub-task as a "screen" action. "OB-2" is a semantic label, which constrains the task to specific work region. Besides what to do in a sub-task, this verbal command also implies how to evaluate the performance of this sub-task. Some of the objectives inherits from the properties of a screen action, the other objectives are from the adverb, e.g. "carefully". This turns the path-planning problem in the sub-task into a multi-objective optimization problem as described in the next section.

4.3 Interactive multi-objective optimization

The adverb in a sentence can be very important and informative. In a cordon and search task, using "carefully", "quickly" or "covertly" imply very different ways of performing the task. More generally, a verbal command from a human contains multiple objectives and constraints, which means that the robot's path-planning problem is a multi-objective optimization problem. Table 4.1 gives an example on different objectives implied by different adverbs in cordon and search. Four adverbs indicate different objectives that the robot's path planner may need to respect.

Table 4.1: Different objectives implied by different adverbs.

Adverb	Covertly	Safely	Quickly	Carefully
Objectives	min visibility max smoothness	min exposure min danger	min path length	max smoothness min collision risk

Notice, as illustrated in Table 4.1, that the robot requires a precise definition so that the problem is mathematically solvable. By contrast, a human may convey and process the information in fuzzier terms. In terms of the representation element of the teammate model, this means that there is a mismatch between the human and the robot. More specifically, there is a mismatch between the precise mathematical objectives required by the robot and the possibly ambiguous adverbial modifier specified by the human. To solve such a problem, we propose a posterior method that allows the robot to specify a range of possible solutions, and allows the human to select from this range.

Specifically, we use the notion of *Pareto optimality* to evaluate the solutions in a multi-objective optimization problem. A solution is called "Pareto optimal" if no other solution has better fitness values in all the objectives. This means that a Pareto optimal

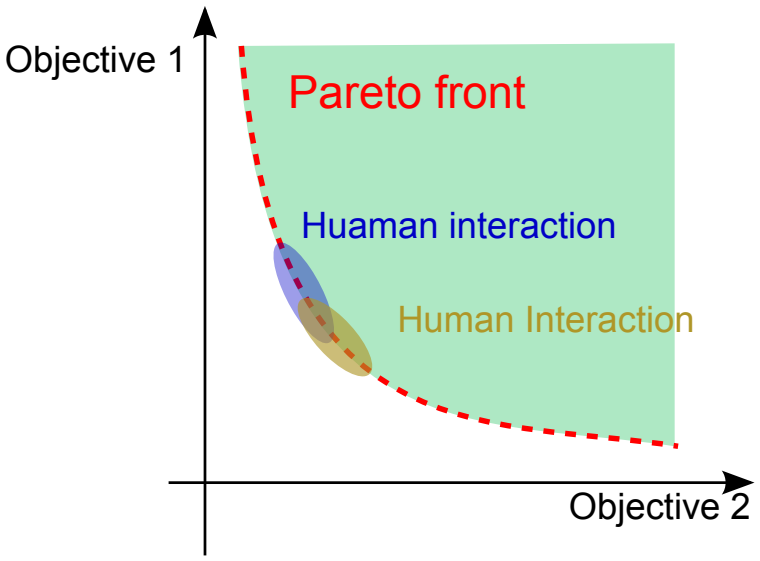


Figure 4.4: Interactive multi-objective optimization.

solution cannot be improved on one objective without downgrading other objectives. A set that consists of all the Pareto optimal solutions is a *Pareto front*. Figure 4.4 illustrates a Pareto front for a minimization problem.

The task grammar from Section 4.2 allows two different types of information to be shared among the human supervisor and the robot: the specific task to be performed (encoded as the verb and noun) and an adverbially qualifier on how the task should be performed. The verb and noun specify hard constraints that must be satisfied by the solution generated by the robot's path planner. By contrast, the adverbial modifier represents a soft constraint that the path should satisfy.

We assume that the soft constraint is fuzzy, meaning that there are several possible paths that would satisfy the soft constraint, and selecting from among these possible paths. requires an ability to balance various tradeoffs. The Pareto front represents all possible tradeoffs, so a specific adverbial modifier does not specify a single point in the Pareto front, but rather a region of the Pareto front; any solution within this region might match the human's intent. This is illustrated on Figure 4.4 as the shaded ovals. We are developing a tool that allows a human to interactively explore this region to select a path that balances

tradeoffs which satisfy human intent. This tool bridges the difference between the way a human and a robot represent a task, and thus facilitates more effective shared mental models.

In terms of the robot's path planner, since the solutions generated from the multi-objective planner are Pareto optimal, information communicated from the planners to the human allow the human to refine their intent by selecting among these tradeoffs. We introduce this interactive multi-objective optimization to solve the path planning problem modeled from a human verbal command.

Unfortunately, the solution space of a path planning has been greatly expanded. This increases the difficulty of solving the multi-objective optimization problem. Following related work on blending metric-based and topological approaches to path planning, we are developing a robot path representation using waypoints and trajectories that connect two neighboring waypoints. This enables a two layer planning in order to enhance the planning efficiency:

- A coarse layer generates the waypoints.
- A fine layer generates the trajectories between the waypoints.

The planning in both layers follow the same objectives and constraints.

When we have a shared mental model with semantic labels and a path planner that solves the multi-objective optimization problem, we can provide an efficient framework of optimized path-planners that can be flexible and adaptive to new forms of objective definitions from new scenarios and new information sources.

4.4 System framework

We propose the system flow shown in in Figure 4.5. Either the human supervisor or an automated object recognizer labels the search space to initialize a shared mental model. With the shared mental, a parse converts a verbal command from the human into a sub-task abstraction for robot. In the path planning problem, this abstraction is encoded as a mathematical model of the optimization problem. Each adverb in the task grammar is

associated with a different planning objective. The path planner generates the set of Pareto optimal solutions and the human selects one that matches his/her intent. After a solution is found, it is sent to a robot to execute.

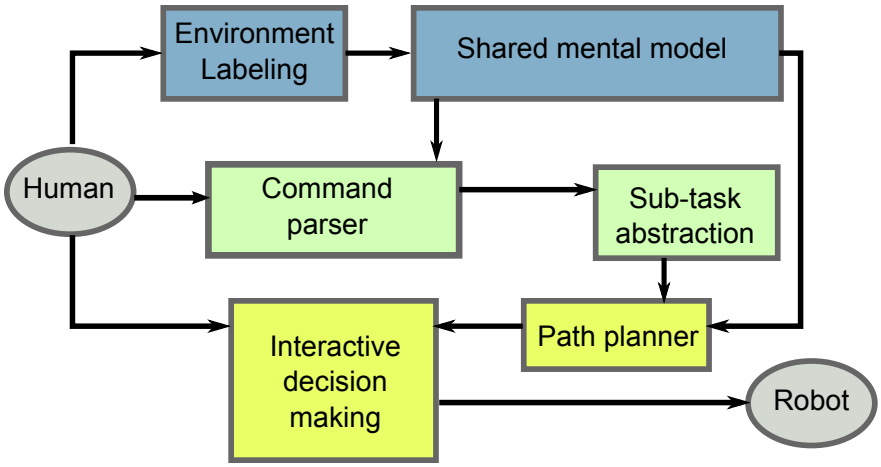
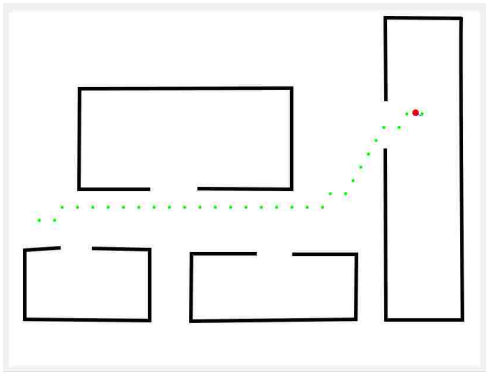
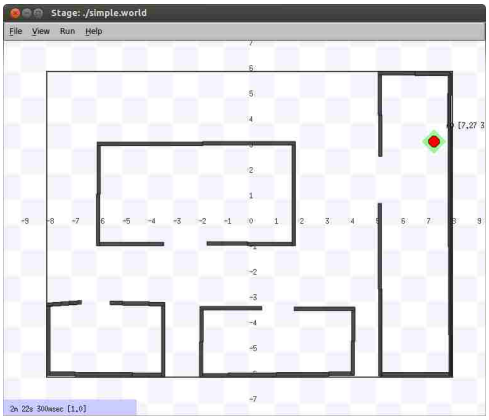


Figure 4.5: The process of a semantic-based path planning.

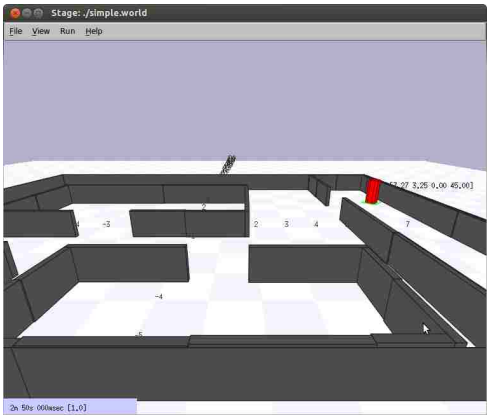
We are currently developing tools to test our approach. The map of a workspace is firstly labeled by a supervisor. A labeled map is shown in Figure 4.3. We assign semantic IDs to different sub-regions by using "indoor" and "outdoor" labels. We label the doors and windows of several regions, which are frequently used in the verbal commands of a cordon and search task. Moreover, we label some significant objects on the ground to facilitate localizing the positions of task execution. Within the labeled map, a verb command, "go quickly to location TR1-OB-2", is read and parsed into multi-objectives and constraints by the task grammar. A path planner finds an "optimal" solution through interactive multi-objective optimization. The path is modeled as a sequence of waypoints. Before implementing on a real robot, we test the task execution in the Stage simulator by using a virtual robot, which are illustrated in Figure 4.6b and 4.6c. A execution monitor GUI, shown in Figure 4.6a, displays the sequence of waypoints of a planned path and continuously receives the position updates from the virtual robot, which is used to check how the task is executed. The process is shown in Figure 4.6.



(a) Planned path execution monitor.



(b) 2D view on Stage simulator.



(c) 3D view on Stage simulator.

Figure 4.6: Simulation with the Stage simulator.

4.5 Summary

In order to support the task-oriented collaboration in a human-robot team, natural language can be used for the interaction between the human and the robot. A shared mental model is needed for the collaboration to help the team members understand each other correctly. The shared mental model is initialized through a semantic labeling process. With a labeled world model and a task grammar, a robot can translate a verbal command from the supervisor into a multi-objective path planning problem. A human interactive decision making process is introduced to find the preferred solution and correct the potential bias from the problem modeling. The planned path can be interactively obtained from the Pareto solution set.

Chapter 5

MORRF*: Sampling-based Multi-Objective Motion Planning [1]

Abstract

Many robotic tasks require solutions that maximize multiple performance objectives. For example, in path-planning, these objectives often include finding short paths that avoid risk and maximize the information obtained by the robot. Although there exist many algorithms for multi-objective optimization, few of these algorithms apply directly to robotic path-planning and fewer still are capable of finding the set of Pareto optimal solutions. We present the MORRF* (Multi-Objective Rapidly exploring Random Forest*) algorithm, which blends concepts from two different types of algorithms from the literature: Optimal rapidly exploring random tree (RRT*) for efficient path finding [59] and a decomposition-based approach to multi-objective optimization [130]. The random forest uses two types of tree structures: a set of *reference trees* and a set of *subproblem trees*. We present a theoretical analysis that demonstrates that the algorithm asymptotically produces the set of Pareto optimal solutions, and use simulations to demonstrate the effectiveness and efficiency of MORRF* in approximating the Pareto set.

[1] Published in Twenty-Fourth International Joint Conference on Artificial Intelligence(IJCAI15). Authors are Daqing Yi, Michael A. Goodrich and Kevin D. Seppi.

5.1 Introduction

Many tasks assigned to robots are complex, can be performed in several different ways, and must maximize several different performance objectives. For example, a robot in a search task may be expected to maximize the area that it covers while minimizing energy consumption and avoiding risk (see, for example [81, 123]). As another example, a robot manipulator may need to satisfy performance criteria related to movement, joint velocities, joint accelerations, etc. [89].

A common method for finding a solution to a multi-objective optimization problem is to optimize a single objective created by a weighted sum of the multiple objectives. In path-planning the properties of the path produced by this method depend strongly on how each objective is weighted. This means that a programmer, designer, or human teammate must decide how to assign the weights so that the qualitative behavior matches what is intended. In addition to the burden this places on the human operator, optimizing a weighted sum does not work when the multiple objectives are very difficult to compare or are expressed in incommensurate units.

In response to these challenges, it is useful to find *the set of Pareto optimal solutions* to the multi-objective path-planning problem, meaning the set of solutions for which there is no other solution that produces better payoffs for every objective. If an algorithm could produce the set of Pareto optimal solutions then a human could interactively explore this set to find one or more solutions that matches his or her expectations. The objective of this paper is to create an algorithm that efficiently finds the Pareto set in a multi-objective path-planning problem.

Most popular methods in multi-objective optimization do not naturally apply to path-planning problems [34, 130]. The main reason for this is that path-planning often represents the problem to be solved as a semi-structured tree with an exponential number of possible trajectories through the tree, and the number of evaluations of the objective function required by existing algorithms do not scale well when there are an exponential number of

solutions. One approach to addressing this issue is to change the representation for a path by, for example, coding a path as a sequence of fixed-length line segments represented by direction [4, 52] or waypoints [89, 111] so that an evolutionary algorithm can be applied. This produces an encoding that can be "fed into" an appropriate evolutionary algorithm to search for the Pareto set. Unfortunately, these approaches do not scale well for large problems, because the number of segments required to represent the paths grows too quickly and estimating the required number of segments *a priori* is very challenging. Another approach is to represent the path as a point in a very high-dimensional vector space. In this approach a path is represented as a point in a $n * d$ dimensional space formed by n d-dimensional way-points. If the number of way-points can be held constant, we can use standard approaches to explore the space. However the search can be more difficult if we allow the number of way-points, and therefore the dimensionality of the optimization problem, to vary. Indeed, we will use this to guide our solution, but the algorithm we present works when the obstacles in the path-planning space introduce discontinuities in these high-dimensional spaces, which limits the applicability of heuristic-based search approaches [111, 130].

The RRT (Rapidly exploring Random Tree) algorithm is popular for finding feasible solutions from a start position to a goal position in continuous or very large search spaces; it also works well when environments have complex obstacles. The reason that RRT is popular is that the tree structure tends to find solutions very efficiently. The RRT* algorithm was a recently introduced modification to RRT that is guaranteed to find an optimal path given enough sampling time [59, 60].

The remainder of the paper presents the MORRF* (Multi-Objective Rapidly exploring Random Forest*) algorithm, which we used to find a set of Pareto optimal paths. MORRF* blends concepts from RRT* and decomposition-based approach to multi-objective optimization [130].

5.2 Related Work

Prior work has modeled the search space as a graph and applied a multi-objective A* search to find the solution [72]. The limitation of this approach is that it requires an *a priori* discretization rather than a discretization that is guided by the objectives as is done in RRT*; a coarse discretization throws away potentially valuable information and a fine discretization increases complexity and adds redundancy in the resulting graph structures. Obstacles can make it more difficult to determining which cells in the discretized space are connected to which others, especially when searching a space of more than 2 dimensions such as in planning the trajectory for a robotic manipulator. Another approach that uses an *a priori* discretization (and suffers from these limitations) is to encode a path as a sequence of directions from one cell to next cell and then using the NSGA-II algorithm to find a set of Pareto optimal solutions [4]. Constrained splines have been introduced to interpolate a sequence of way points into a trajectory that avoids obstacles [3], but the effect of the interpolation on the quality of the solution has not been evaluated. In addition to the sorting approach used in NSGA-II, evolutionary algorithms based on the decomposition method have also been proposed [34].

Evolutionary algorithms can be used to fine the Pareto set, but these approaches tend to be inefficient when applied to spaces with high dimensions [73]. For such spaces, small deviations in possible solutions may need to be considered in order to find an optimal solution, but this means exploring many possible solutions for problems with non-linearities or multiple local maxima. A path in a fixed-length search tree of depth d can be considered as a point in $\Re^d$, so tree-based approaches followed by an evolutionary "fine-tuning" stage risk running into the problems just listed with evolutionary approaches.

In contrast to searching through and comparing solutions in order to find the Pareto optimal set, decomposition-based methods provide an attractive alternative. In this paper we use a decomposition-based method similar to MOEA-D [130]. MOEA-D is an algorithm that decomposes a multi-objective optimization problem into a set of subproblems. Each

subproblem uses a weighed combination of the objectives to find specific points in the Pareto set or to guide the search for such points. Let $\boldsymbol{\lambda} = [\lambda_1, \cdots, \lambda_K]^T$ be a weighting vector such that $\sum_{k=1}^{K} \lambda_k = 1$. Let $\{c_1(\cdot), c_2(\cdot), \ldots c_K(\cdot)\}$ denote the K-element set of objective functions, let $\boldsymbol{c}(x) = [c_1(x), c_2(x), \ldots, c_K(x)]^T$, and let x denote a potential solution. Finally, let $\boldsymbol{z}^{\text{utop}} = [z_1^*, \cdots, z_K^*]^T$ denote the so-called Utopia reference vector. Three types of decomposition methods have been used in prior work [130]; however we will use only the two methods described below, leaving the third (the boundary intersection method) to future work.

$$\arg\max_x \sum_{k=1}^{K} \lambda_k c_k(x) \quad \text{weighted sum} \tag{5.1}$$

$$\arg\min_x \max_{1 \le k \le K} \{\lambda_k \left(|c_k(x) - \boldsymbol{z}_k^{\text{utop}}|\right)\} \quad \text{Tchebycheff} \tag{5.2}$$

The solutions generated by each method are a subset of the Pareto optimal set.

Sampling-based path planning works effectively in continuous space. The RRT (Rapidly exploring Random Tree) has been one of the most popular tools, which efficiently explores the space by randomly sampling the search space; this algorithm tends to work well in the presence of complex obstacles. Unfortunately, RRT has been shown to fail in optimality guarantee [59]. In response, the RRT* algorithm was proposed, which uses a *Rewire* process to gradually update the tree structure when new samples of the space indicate that this is needed. Thus RRT* is asymptotically optimal [59, 60].

5.3 Multi-Objective Rapidly exploring Random Forest*

In this section, we present an algorithm that explores the solution space using RRT*-based tree structures but uses multiple trees in the spirit of decomposition-based multi-objective optimization. Because a set of trees are constructed in the exploration process, we call the algorithm MORRF* (Multi-Objective Rapidly exploring Random Forest*).

Consider a multi-objective path planning problem defined on a bounded, connected open set $X \subset \mathbb{R}^d$ of possible solutions, and K different objectives $\{c_1(\cdot), c_2(\cdot), ...c_K(\cdot)\}$. Without loss of generality, assume that the objective is to minimize these functions. Since the Pareto optimal set is not enumerable, the goal is to find a representative, finite (M-element) subset of the Pareto optimal set.

Definition 1. ***Multi-Objective Path Planning*** *Consider a bounded, connected open set $X \subset \mathbb{R}^d$, an obstacle space X_{obs}, an initial state x_{init}, and a goal region X_{goal}. Consider the set of K objectives determined by a vector function $\mathbf{c}(\cdot) = [c_1(\cdot), \ldots, c_K(\cdot)]^T$ defined by $\boldsymbol{c} : \mathbb{X} \to \mathbb{R}^K$. Denote the obstacle-free space by $X_{free} = X \setminus X_{obs}$. Note that $\boldsymbol{c}$ is defined for all points in X both those in free space and obstacle space.*

Define a path *in X as a continuous curve parameterized by s, denoted by $\sigma : [0, s] \to X$. Define the cost of the path as the vector-valued function $\mathbf{c}(\sigma) = \int_\sigma \mathbf{c}(x)ds$. The goal is to find M Pareto optimal paths $\sigma^* \in \Sigma^*$ that (a) $\forall \tau \in [0, s], \sigma^*(\tau) \in X_{free}$; (b) $\sigma^*(0) = x_{init}$ and $\sigma^*(s) = X_{goal}$; (c) There does not exist σ that $\forall k \in K, c_k(\sigma) \leq c_k(\sigma^*)$ and $\exists k' \in K, c_{k'}(\sigma) < c_{k'}(\sigma^*)$.*

Adopting the idea from the MOEA-D algorithm [130], the M elements in the solution set Σ^* will be obtained by decomposing the multi-objective problem into M subproblems. In this paper, we use the Tchebycheff approach from MOEA-D. The Tchebycheff approach requires us to define a Utopia reference vector $\boldsymbol{z}^{\text{utop}}$ in the fitness space. As illustrated in Figure 5.1, the Utopia reference vector is defined as that point in cost space that would be obtained if it were possible to find a solution that produced the minimum value for all objectives, that is the k^{th} element of $\boldsymbol{z}^{\text{utop}}$ is given by $\boldsymbol{z}_k^{\text{utop}} = \arg\min_{x \in X} c_k(x)$.

We will need one type of RRT* structure to explore in an attempt to find the Utopia reference vector in payoff space and another type of RRT* structure to find paths that minimize the Tchebycheff condition. Thus, there are two types of tree structures used for the optimization process.

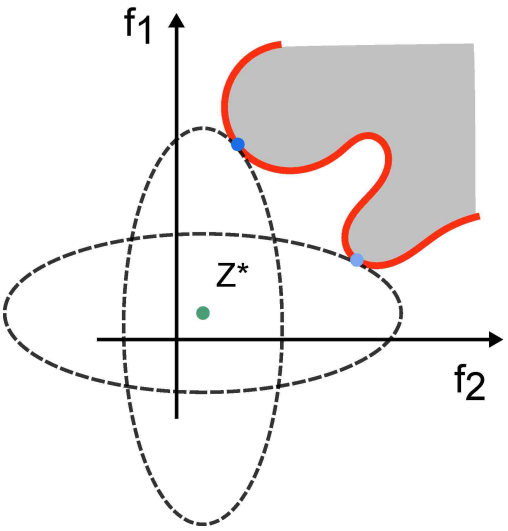


Figure 5.1: Tchebycheff method of finding Pareto front.

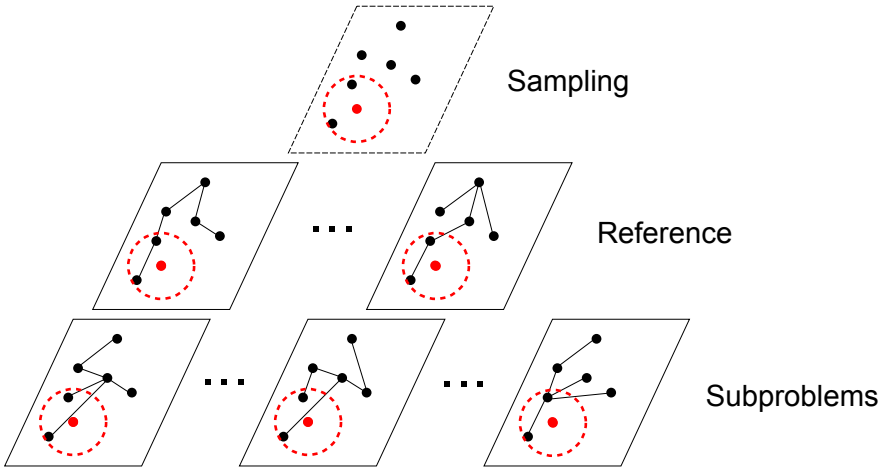


Figure 5.2: Rapidly exploring process

- Each *reference tree* explores using a single objective $c_k(x), k \in K$. The cost of each vertex is calculated using the k^{th} objective function.
- Each *subproblem tree* explores a subproblem $g_m(x \mid \lambda_m, \boldsymbol{z}^{\text{utop}}), m \in M$. The cost associated with each vertex is calculated using $g_m(x)$ defined by the corresponding approach.

The K reference trees and M subproblem trees constitute the exploration forest.

The main flow of the MORRF* algorithm is given in Algorithm 1. Each reference and subproblem tree are a collection of edges and vertices, $G_r = (V_r, E_r)$ and $G_s = (V_s, E_s)$,

respectively, and the collection of reference trees and subproblem trees are denoted by $\mathbf{G}_r = \{G_r : r \in \{1, \ldots, K\}\}$ and $\mathbf{G}_s = \{G_s : s \in \{q, \ldots, M\}$.

Note that each tree, reference and subproblem, uses the same set of vertices, meaning they all share the same points in configuration space. The differences between the trees is the edge set; each reference tree and each subproblem tree has a different way of connecting the vertices.

In each iteration, x_{rand} is generated by randomly sampling from the configuration space. The set of vertices is then searched to find that vertex whose position is nearest to the random point; since all trees share the same set of vertices, any tree $G \in \mathbf{G}_r \cup \mathbf{G}_s$ may be used to find the nearest point. The location of this vertex is labeled $x_{nearest}$. The process of finding x_{new} is represented in the top layer of Figure 5.2.

The exploration at each iteration is given in Algorithm 1. Like RRT*, when the algorithm stops, each reference tree and subproblem tree returns a path, and the set of all these paths forms the solution set.

Algorithm 1 Multi-Objective Rapidly exploring Random Forest*

for each $V_r \in \mathbf{V}_r$ **do**
  $V_r \leftarrow \{x_{init}\}$; $E_r \leftarrow\leftarrow$; $i \leftarrow 0$
for each $V_s \in \mathbf{V}_s$ **do**
  $V_s \leftarrow \{x_{init}\}$; $E_s \leftarrow \emptyset$; $i \leftarrow 0$
while $i < N$ **do**
  $x_{rand} \leftarrow$ SAMPLE (i) ; $i \leftarrow i + 1$
  G is arbitrary graph from $\mathbf{G}_r \cup \mathbf{G}_s$.
  $x_{nearest} \leftarrow$ NEAREST(G, x_{rand})
  $x_{new} \leftarrow$ STEER$(x_{nearest}, x_{rand}, \eta)$
  if OBSTACLEFREE$(x_{nearest}, x_{new})$ **then**
    for each $G_r \in \mathbf{G}_r$ **do**
      $G_r \leftarrow$ EXTEND$_{Ref}$ $(G_r, x_{new}, x_{nearest}, r)$
    for each $G_s \in \mathbf{G}_s$ **do**
      $G_s \leftarrow$ EXTEND$_{Sub}$ $(G_s, x_{new}, x_{nearest}, s)$

We now define several functions, using appropriately modified definitions from [59].

- SAMPLE(): Returns independent uniformly distributed samples from X_{free}.

- NEAREST(): Returns a position of the vertex whose position is closest to point x. $\text{NEAREST}(G = (V, E), x) = \arg\min_{v \in V} \|x - v\|$.
- STEER(): Given two points x and y, returns a point z on the line segment from x to y that that is no greater than η from y. $\text{STEER}(\ x, y, \eta\ ) = \arg\min_{z \in \mathbb{R}^d, \|z-x\| \leq \eta} \|z - y\|$.
- OBSTACLEFREE(x, x'): Returns True if $[x, x'] \subset X_{free}$, which is the line segment between x and x' lies in X_{free}.

As illustrated in Figure 5.2, second layer, edges to the reference trees are added before the edges to the subproblem trees. This allows us to compute the Utopia reference vector using the path costs for each reference tree, each reference tree returning a path that approximates the minimum cost for one objective. The Utopia reference vector is then used to determine which edges should be added for each subproblem.

Consider the second layer in Figure 5.2, which shows the exploration process for the reference trees. When a new position is obtained (red dot in Figure 5.2), all reference trees add a vertex that corresponds to this new location. Each reference tree then connects this new vertex to existing nodes by "rewiring" a set of neighboring vertices within a specified radius (red dash circle in Figure 5.2). The process of rewiring consists of adding edges between existing vertices and the new vertex. This is done using the EXTEND method, given in Algorithm 2.

The precise definitions of the methods used in the Algorithm 2 are given below.

- NEAR(G, x, η): Returns a set of all vertices within the closed ball of radius r_n centered at x, in which $r_n = \min\{(\frac{\gamma}{\xi_d}\frac{\log n}{n})^{1/d}, \eta\}$. The volume of the ball is $\min\{\gamma\frac{\log n}{n}, \xi_d \eta^d\}$.
- LINE$(x, x') : [0, s] \leftarrow X_{free}$ denotes the path defined by line segment from x to x'.
- COST(v): Returns the cost of the unique path (because G is a tree) from x_{init} to the vertex $v \in V$. $\text{COST}(x_{\text{init}}) = 0$.

Consider the third layer in Figure 5.2, which illustrates how the subproblem trees "rewire" to connect to the new vertex. The Utopia reference vector, $\hat{\boldsymbol{z}}_k^{\text{utop}}$ is defined as the

Algorithm 2 EXTEND$_{Ref}$ $(G, x_{new}, x_{nearest}, k)$

1: **if** $x_{new} = x_{nearest}$ **then return** $G = (V, E)$
2: $V' \leftarrow V \cup \{x_{new}\}$
3: $x_{min} \leftarrow x_{nearest}$
4: $X_{near} \leftarrow \text{NEAR}(G, x_{new}, |V|)$
5: **for each** $x_{near} \in X_{near}$ **do**
6: **if** OBSTACLEFREE(x_{new}, x_{near}) **then**
7: $c'_k \leftarrow \text{COST}_k(x_{near}) + c_k(\text{ LINE}(x_{near}, x_{new}))$
8: **if** $c'_k < \text{COST}_k(x_{new})$ **then**
9: $x_{min} \leftarrow x_{near}$
10: $E' \leftarrow E' \cup \{(x_{min}, x_{new})\}$
11: **for each** $x_{near} \in X_{near} \setminus \{x_{min}\}$ **do**
12: **if** OBSTACLEFREE(x_{new}, x_{near}) **then**
13: $c'_k \leftarrow \text{COST}_k(x_{new}) + c_k(\text{ LINE}(x_{new}, x_{near}))$
14: **if** $c'_k < \text{COST}_k(x_{near})$ **then**
15: $x_{parent} \leftarrow \text{PARENT}(x_{near})$
16: $E' \leftarrow E' \setminus \{(x_{parent}, x_{near})\}$
17: $E' \leftarrow E' \cup \{(x_{new}, x_{near})\}$
return $G' = (V', E')$

k-dimensional vector constructed from each reference tree. The minimum cost of each path from the starting vertex over any other vertex is computed for each reference tree. Using the Utopia reference vector, each subproblem tree connects its new vertex and rewire neighboring vertices in a radius as well. Algorithm 3 precisely follows Algorithm 2 except that instead of computing the cost using one of the objectives, the cost is computed using the Tchebycheff method; each of the m^{th} subproblem trees corresponds to a different weighting vector λ_m. This is performed using the FITNESS method.

The FITNESS method computes costs using one of the cost functions in Equations (5.1)-(5.2). Different values of λ_m are obtained using the pattern in the MOEA-D algorithm: (a) pre-deterimining the range of the K-cost functions, $\{c_k() : 1 \ldots K\}$ and (b) sampling from the K-dimensional hypercube defined by these ranges. The M samples from this hypercube can be obtained by either creating a uniform (hyper)-grid or by doing uniform sampling across the space.

Algorithm 3 $\textsc{Extend}_{Sub}$ $(G, x_{new}, x_{nearest}, m)$

if $x_{new} = x_{nearest}$ **then return** $G = (V, E)$
$V' \leftarrow V' \cup \{x_{new}\}$
$x_{min} \leftarrow x_{nearest}$
$X_{near} \leftarrow \textsc{Near}(G, x_{new}, |V|)$
for each $x_{near} \in X_{near}$ **do**
    if $\textsc{ObstacleFree}(x_{new}, x_{near})$ **then**
        $\boldsymbol{c}' \leftarrow \textsc{Cost}(x_{near}) + \boldsymbol{c}(\ \textsc{Line}(x_{near}, x_{new})\ )$
        $\eta' = \textsc{Fitness}(\ \boldsymbol{c}', \hat{\boldsymbol{z}}^{\text{utop}} \mid \lambda_m\ )$
        $\boldsymbol{c}_{new} = \textsc{Cost}(x_{new})$
        $\eta_{new} = \textsc{Fitness}(\ \boldsymbol{c}_{new}, \hat{\boldsymbol{z}}^{\text{utop}} \mid \lambda_m\ )$
        if $\eta' < \eta_{new}$ **then**
            $x_{min} \leftarrow x_{near}$
$E' \leftarrow E' \cup \{(x_{min}, x_{new})\}$
for each $x_{near} \in X_{near} \setminus \{x_{min}\}$ **do**
    if $\textsc{ObstacleFree}(x_{new}, x_{near})$ **then**
        $\boldsymbol{c}' \leftarrow \textsc{Cost}(x_{new}) + \boldsymbol{c}(\ \textsc{Line}(x_{new}, x_{near})\ )$
        $\eta' = \textsc{Fitness}(\ \boldsymbol{c}', \hat{\boldsymbol{z}}^{\text{utop}} \mid \lambda_m\ )$
        $\boldsymbol{c}_{near} = \textsc{Cost}(x_{near})$
        $\eta_{near} = \textsc{Fitness}(\ \boldsymbol{c}_{near}, \hat{\boldsymbol{z}}^{\text{utop}} \mid \lambda_m\ )$
        if $\eta' < \eta_{near}$ **then**
            $x_{parent} \leftarrow \textsc{Parent}(x_{near})$
            $E' \leftarrow E' \setminus \{(x_{parent}, x_{near})\}$
            $E' \leftarrow E' \cup \{(x_{new}, x_{near})\}$
return $G' = (V', E')$

5.4 Analysis

The analysis depends on the following restrictions on the cost functions and obstacle placement required by the RRT* algorithm [59]. We claim without argument that the cost functions and obstacle placements used in the simulation studies satisfy the restrictions.

Assumption 1. *(Additivity of the objective functions) For a path constructed by composing two other paths (to create a discontinuous path),* $\forall k \in K, \sigma_1, \sigma_2 \in X_{\text{free}}$, $c_k(\sigma_1 \circ \sigma_2) = c_k(\sigma_1) + c_k(\sigma_2)$.

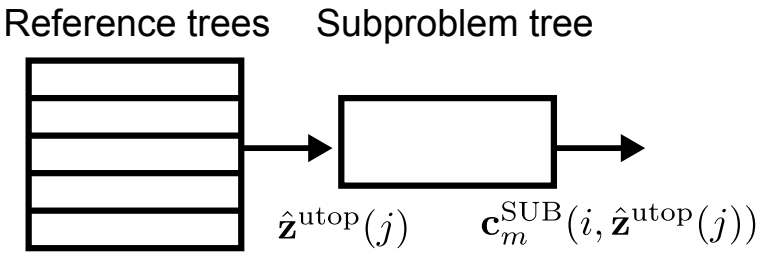


Figure 5.3: The dependency of the trees in MORRF*.

Assumption 2. *(Continuity of the cost functions) For all* $k \in K$*, the cost function* c_k *is Lipschitz continuous, that is, for all paths* $\sigma_1 : [0, s_1] \to X_{free}$ *and* $\sigma_2 : [0, s_2] \to X_{free}$*, there exists a constant* $\kappa(k) \in \mathbb{R}_+ \cup \{0\}$ *such that* $|c_k(\sigma_1) - c_k(\sigma_2)| \leq \kappa(k) \sup_{\tau \in [0,1]} \|\sigma_1(\tau s_1) - \sigma_2(\tau s_2)\|$.

Assumption 3. *(Obstacle spacing) There exists a constant* $\delta \in \mathbb{R}_+$ *such that* $\forall x \in X_{free}$ *,* $\exists x' \in X_{free}$ *such that*

- *the* δ*-ball centered at* x' *lies inside* X_{free}*;*
- x *lies inside the* δ*-ball centered at* x'*.*

Lemma 1. *If the Utopia reference vector satisfies* $\forall k, \sigma \; z_k^{\text{utop}} \leq c_k(\sigma)$*, then any solution of Eq.* (5.2) *is Pareto optimal.*

Proof. The proof is by contradiction. Let the weighting vector λ be arbitrary subject to $\forall k\ \lambda_k \geq 0$, and let $\sigma^* = \sigma^*(\lambda)$ be a solution given that weighting vector. By definition,

$$\sigma^* = \arg\min_{\sigma} \max_{k \in K} \lambda_k |c_k(\sigma) - z_k^{\text{utop}}|. \tag{5.3}$$

Assume that the path σ^* is not Pareto optimal. Then there exist another path σ^o that dominates σ^* and the Utopia reference vector that satisfies $\forall k \in K$, $z_k^{\text{utop}} \leq c_k(\sigma)$, it follows that $\forall k \in K, z_k^{\text{utop}} \leq c_k(\sigma^o) \leq c_k(\sigma^*)$ and $\exists k' \in K, z_k^{\text{utop}} \leq c_{k'}(\sigma^o) < c_{k'}(\sigma^*)$. These equations imply

$$\begin{aligned} &\forall k \in K, \quad \lambda_k |c_k(\sigma^*) - z_k^{\text{utop}}| \geq \lambda_k |c_k(\sigma^o) - z_k^{\text{utop}}|; \\ &\exists k' \in K, \quad \lambda_{k'} |c_{k'}(\sigma^*) - z_k^{\text{utop}}| > \lambda_{k'} |c_{k'}(\sigma^o) - z_k^{\text{utop}}|; \end{aligned}$$

which yields the following contradiction to Eq (5.3):

$$\max_{k \in K} \lambda_k |c_k(\sigma^*) - z_k^{\text{utop}}| > \max_{k \in K} \lambda_k |c_k(\sigma^o) - z_k^{\text{utop}}|.$$

□

Lemma 2. *If σ^* is Pareto optimal then there exists a weighting vector $\boldsymbol{\lambda}$, where $\forall k\ \lambda_k \geq 0$ and $\sum_{k=1}^{K} \lambda_k = 1$, such that σ^* is a solution of Eq.* (5.2)*.*

Proof. This is a proof by construction over cases. When σ^* is Pareto optimal, there exist two cases: (a) $\exists k, c_k(\sigma^*) = z_k^{\text{utop}}$ and (b) $\forall k, c_k(\sigma^*) > z_k^{\text{utop}}$.

Case (a): $\exists k, c_k(\sigma^*) = z_k^{\text{utop}}$

Define $P(\sigma^*) = \{j \mid c_j(\sigma^*) = z_j^{\text{utop}}\}$ and let $\overline{P} = \{1, \ldots, K\} \setminus P$. Define the weight vector $\boldsymbol{\lambda}$ as $\forall k \in P(\sigma^*), \lambda_k = \frac{1}{|P|}$ and $\forall k \in \overline{P}(\sigma^*), \lambda_k = 0$. For these weights, Eq. (5.2) returns a set of solution paths, all of which have the same cost for the k-cost functions when $k \in P$ but different possible costs for $k \in \overline{P}$. σ^* is trivially in this set of solution paths.

Case (b): $\forall k, c_k(\sigma^*) > z_k^{\text{utop}}$

For all k, define the weights as $\lambda_k = \frac{\ell_k}{\sum_{j=1}^{K} \ell_j}$, where $\ell_k = \frac{1}{|c_k(\sigma^*) - z_k^{\text{utop}}|}$. The Tchebycheff cost (Eq. (5.2)) becomes

$$g^{te}(\sigma^*) = \max_{k \in K} \frac{|c_k(\sigma^*) - z_k^{\text{utop}}|}{|c_k(\sigma^*) - z_k^{\text{utop}}|} \frac{1}{\sum_{j=1}^{K} \ell_j} = \frac{1}{\sum_{j=1}^{K} \ell_j}$$

Given any other path σ, we can represent the Tchebycheff cost as follows:

$$\begin{aligned} g^{te}(\sigma) &= \max_{k \in K} \frac{\ell_k}{\sum_{j=1}^{K} \ell_j} |c_k(\sigma) - z_k^{\text{utop}}| \\ &= \frac{1}{\sum_{j=1}^{K} \ell_j} \max_{k \in K} \left| 1 + \frac{c_k(\sigma) - c_k(\sigma^*)}{c_k(\sigma^*) - z_k^{\text{utop}}} \right| \end{aligned}$$

Because σ^* is Pareto optimal, $[\exists k' \in K, c_{k'}(\sigma) > c_{k'}(\sigma^*)] \vee [\forall k \in K, c_{k'}(\sigma) = c_{k'}(\sigma^*)]$ for any σ. As $\forall k, c_k(\sigma^*) > z_k^{\text{utop}}$, we have $\forall k, c_k(\sigma^*) - z_k^{\text{utop}} > 0$. This implies $\exists k' \in K, \frac{c_{k'}(\sigma) - c_{k'}(\sigma^*)}{c_{k'}(\sigma^*) - z_{k'}^{\text{utop}}} \geq 0$, which, in turn, implies that $\max_{k \in K} \left| 1 + \frac{c_k(\sigma) - c_k(\sigma^*)}{c_k(\sigma^*) - z_k^{\text{utop}}} \right| \geq 1$. Therefore, $g^{te}(\sigma) \geq \frac{1}{\sum_{j=1}^{K} \ell_j} = g^{te}(\sigma^*)$. Thus, σ^* is a solution to Eq. (5.2).

□

By Lemma 1 and Lemma 2, we have the following:

Theorem 1. *A path is Pareto optimal if and only if it is a solution to Eq.* (5.2) *for some weight vector.*

Theorem 1 implies that we can use the Tchebycheff method to find the Pareto set for the multi-objective path-planning problems. The next question that needs to be answered is whether the subproblem tree can find the optimal solution of its assigned subproblem.

The way that the RRT* algorithm works is that it incrementally constructs a tree from a root position. The cost of the path from the position of the root node to the positions of every other node converges to the minimal possible cost between the positions as the number of iterations approaches infinity. We restate this as a lemma, and note that it corresponds exactly to that given for Theorem 22 in [59].

Lemma 3. *Given Assumptions 1-3, the cost of the minimum cost path from the root to any vertex in RRT* converges to the optimal cost almost surely.*

Lemma 3 and Theorem 1 imply that each reference tree converges to the optimal path from the root to any node in the tree, including a node arbitrarily close to the goal node. This means that the costs returned by those trees for the path from the start to the goal for the cost function c_k converges to the k^{th} element of the Utopia reference vector $\boldsymbol{z}^{\text{utop}}$. We state this as a lemma.

Lemma 4. *Given Assumptions 1-3, the cost of the minimum cost path from the root to any vertex in k^{th} reference tree converges to z_k^* almost surely.*

We now turn to the proof that the subproblem trees converge to paths in the Pareto set. The proof of this claim requires that we know $\boldsymbol{z}^{\text{utop}}$ to compute the Tchebycheff cost associated with the cost used in the subproblem. If we knew that the reference trees had already converged to $\boldsymbol{z}^{\text{utop}}$, then we could simply instantiate Lemma 3. Unfortunately, the reference trees are converging at the same time that the subproblem trees are converging. We now address this problem.

Let $\hat{\boldsymbol{z}}^{\text{utop}}(v;i)$ denote the approximate Utopia reference vector for position v on iteration i, estimated by the cost from the root to position x from the k-reference trees. Recall that the m^{th} subtree attempts to generate a solution to Eq. (5.2) for a given weight vector λ^m. Let

$$\boldsymbol{c}_m^{\text{SUB}}(\boldsymbol{z}) = \arg\min_x \max_{k\in K} \lambda_{m,k}|x_k - \boldsymbol{z}_k| \tag{5.4}$$

denote the cost vector in m^{th} subproblem tree given the reference vector $\boldsymbol{z}$ and let $\hat{\boldsymbol{c}}_m^{\text{SUB}}(i,\boldsymbol{z})$ denote its estimation at iteration i. A subproblem tree obtains $\hat{\boldsymbol{z}}^{\text{utop}}(v)$ for vertex v in the reference trees and generate the corresponding $\boldsymbol{c}_m^{\text{SUB}}(v;i,\hat{\boldsymbol{z}}^{\text{utop}}(v))$. This forms a cascade structure from the reference trees to the subproblem tree. By Lemma 4, we have the convergence of the reference trees.

We introduce Assumption 4 to get Lemma 5.

Assumption 4. *(Lipschitz continuity)* $\boldsymbol{c}_m^{\mathrm{SUB}}(\boldsymbol{z})$ *in Eq.* (5.4) *and its estimation* $\hat{\boldsymbol{c}}_m^{\mathrm{SUB}}(i, \boldsymbol{z})$ *are Lipschitz continuous, i.e.* $\|\boldsymbol{c}_m^{\mathrm{SUB}}(\boldsymbol{z}_a) - \boldsymbol{c}_m^{\mathrm{SUB}}(\boldsymbol{z}_b)\| \leq K\|\boldsymbol{z}_a - \boldsymbol{z}_b\|$.

Lemma 5. *Given Assumptions 1-4 , the cost of the solution of* m^{th} *subproblem tree converges to the corresponding cost of the* m^{th} *subproblem* $\boldsymbol{c}_m^*$ *almost surely.*

Proof. By Lemma 4, we have $\lim_{j\to\infty}\|\boldsymbol{z}^* - \hat{\boldsymbol{z}}(j)\| = 0$. By Lemma 3, we have $\lim_{i\to\infty} \hat{\boldsymbol{c}}(i, \hat{\boldsymbol{z}}(j)) = \boldsymbol{c}(\hat{\boldsymbol{z}}(j))$. Thus, $\lim_{i\to\infty}\|\boldsymbol{c}(\boldsymbol{z}^*) - \hat{\boldsymbol{c}}(i, \hat{\boldsymbol{z}}(j))\| = \|\lim_{i\to\infty} \boldsymbol{c}(\boldsymbol{z}^*) - \lim_{i\to\infty} \hat{\boldsymbol{c}}(i, \hat{\boldsymbol{z}}(j))\| = \|\boldsymbol{c}(\boldsymbol{z}^*) - \boldsymbol{c}(\hat{\boldsymbol{z}}(j))\|$.

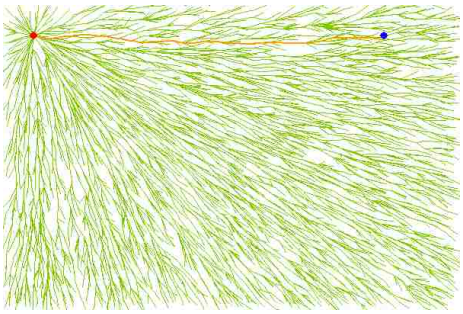
Since $\boldsymbol{c}(\boldsymbol{z})$ and $\hat{\boldsymbol{c}}(i, \boldsymbol{z})$ are Lipschitz continuous; $\lim_{i\to\infty}\|\boldsymbol{c}(\boldsymbol{z}^*) - \hat{\boldsymbol{c}}(i, \hat{\boldsymbol{z}}(j))\| \leq K\|\boldsymbol{z}^* - \hat{\boldsymbol{z}}(j)\|$. As $j \to \infty$, we have $\hat{\boldsymbol{z}}(j) \to \boldsymbol{z}^*$, thus $\lim_{i\to\infty}\|\boldsymbol{c}(\boldsymbol{z}^*) - \hat{\boldsymbol{c}}(i, \hat{\boldsymbol{z}}(j))\| \to 0$. This implies $P(\{\lim_{\substack{i\to\infty \\ j\to\infty}} \boldsymbol{c}_m^{\mathrm{SUB}}(i, \hat{\boldsymbol{z}}(j)) = \boldsymbol{c}_m^*\}) = 1$. □

Now, we can prove that the solution from MORRF* almost surely converges to a subset of the Pareto optimal set.

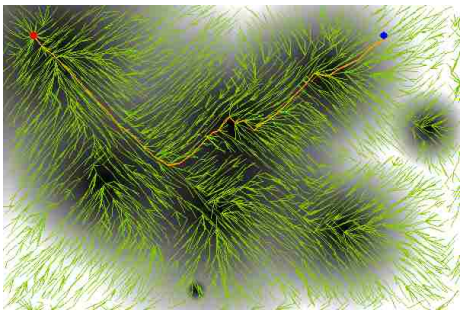
Theorem 2. *Given Assumptions 1-4 , the solution generated by MORRF* converges to a subset of the Pareto optimal set almost surely.*

5.5 Simulation

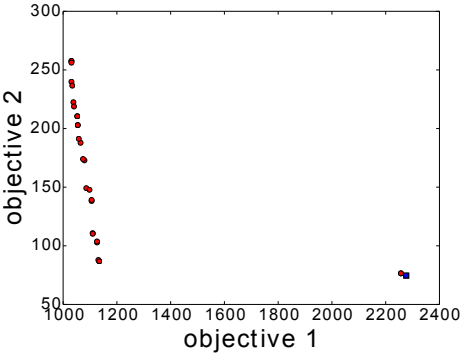
We now present a series of simulation studies that provide evidence that MORRF* produces a representative set of samples from the Pareto set. Results from MORRF* are obtained for path-planning problems with two objectives and three objectives, and are compared to a modified version of the NSGA-II multi-objective path-planning algorithm [4] as well as a variant of MORRF* that uses a weighted sum rather than the Tchebycheff approach. NSGA-II was selected because evidence suggests that it provides more uniform samples from the Pareto set than other approaches [35]. We modified the NSGA-II algorithm for this problem to use paths as inputs, represented by a series of waypoints connected by line segments; the cost calculation is identical with that in MORRF*, calling $\textsc{Line}(x_1, x_2)$ to



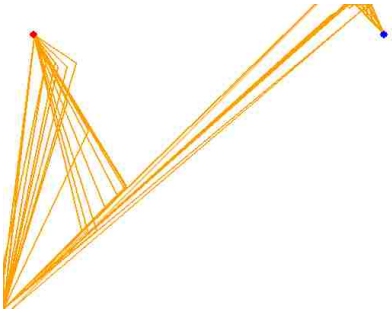
(a) Min distance



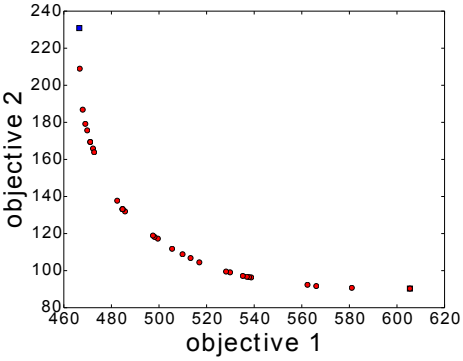
(b) Min cost 1



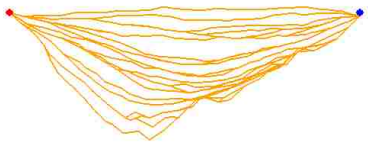
(c) Pareto set: NSGA-II



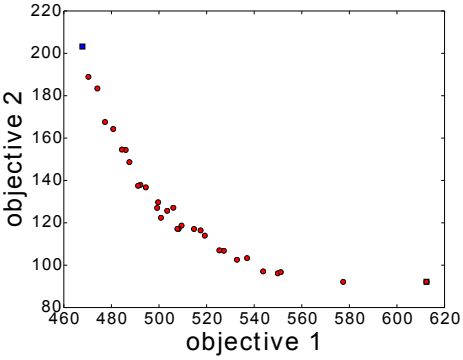
(d) Pareto paths: NSGA-II



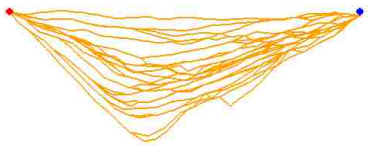
(e) Pareto set: weighted sum



(f) Pareto paths: weighted sum



(g) Pareto set: Tchebycheff



(h) Pareto paths: Tchebycheff

Figure 5.4: Path planning with two objectives.

calculate the cost between two way points x_1 and x_2. The weighted sum approach was chosen because evidence suggests that it works well only when all the objectives are convex [130] whereas the Tchebycheff approach should bring better diversity in the solutions [130]. The weighted sum approach uses the same sampling method for weights as that used to generate the λ_i in MORRF*. Each method was run for 5000 iterations and restricted to 30 solutions.

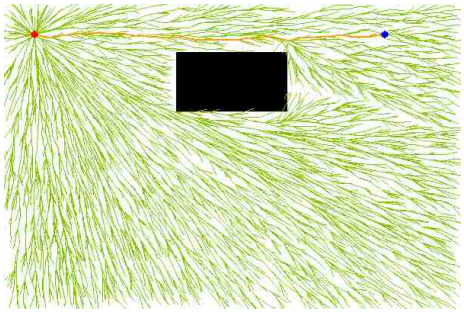
The first simulation study compares three algorithms in an obstacle-free world with two objectives: minimize Euclidean distance, see Figure 5.4a, and minimize a cost function, see Figure 5.4b. The first thing to note is that the convergence of NSGA-II-based path-planning is very slow. This is indicated in Figures 5.4c-5.4d, which show the approximation to the Pareto set and corresponding paths, respectively, after 5000 iterations; observe how the quality of the paths and sampling of the Pareto set is uneven and unsatisfactory. By contrast, the weighted sum approach returns a set of high-quality solutions close to the Pareto optimal set, see Figures 5.4e and 5.4f; Finally, note the somewhat uneven clustering of solutions on Pareto front for MORRF* using weighted sum, and compare this to the slightly more uniform clustering of MORRF* using the Tchebycheff approach in Figures 5.4g-5.4h.

We therefore compared results for the two approaches for an environment with obstacles, omitting results for NSGA-II because convergence is so slow. The results are shown in Figure 5.5. As before, observe that the Tchebycheff approach yields a more uniform sampling, albeit one that appears to be somewhat noisy approximation to the Pareto set.

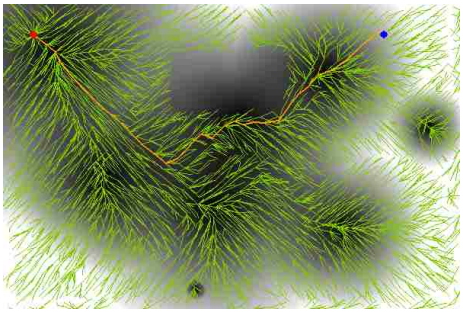
Finally, we evaluated how MORRF* performs three objectives: Euclidean distance and the two other objectives are shown in Figures 5.6a-5.6c. As shown in Figure 5.6, the Pareto front uses the Utopia reference vector (Green point) to better approximate the Pareto set than the weighted sum approach.

5.6 Summary and Future Work

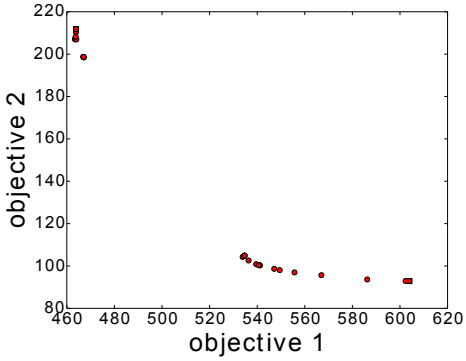
This paper presented the MORRF* algorithm for the multi-objective path-planning problems on continuous spaces. The algorithm blends principles from the RRT* algorithm with princi-



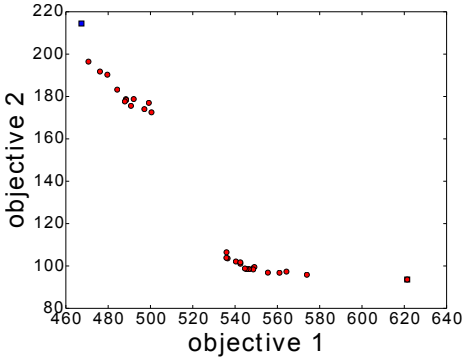
(a) Min distance



(b) Min cost 1

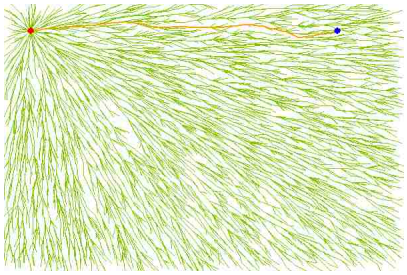


(c) Pareto set: weighted sum

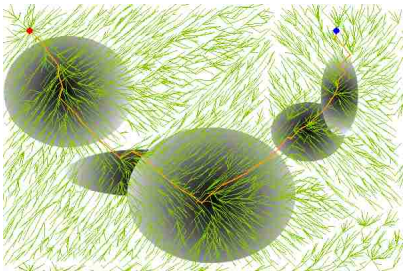


(d) Pareto set: Tchebycheff

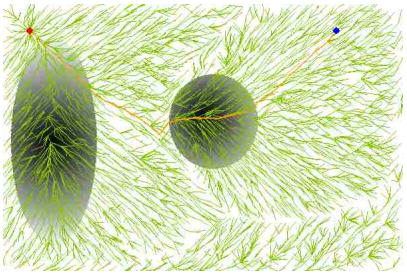
Figure 5.5: Path planning with two objectives and an obstacle.



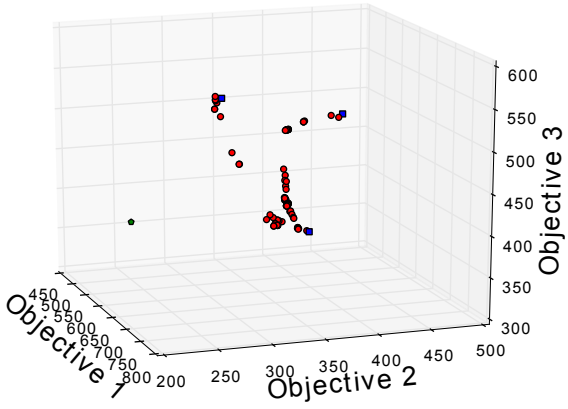
(a) Distance



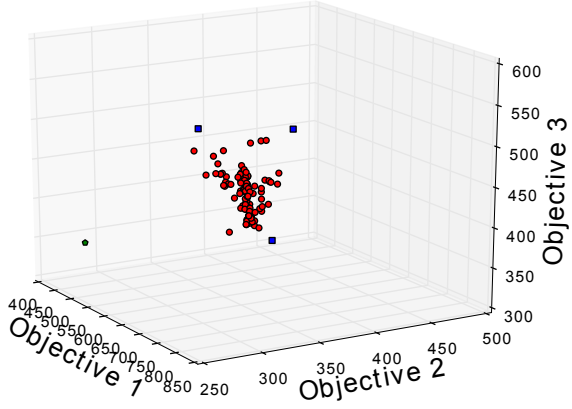
(b) Cost 1



(c) Cost 2



(d) Pareto set: weighted sum



(e) Pareto set: Tchebycheff

Figure 5.6: Path planning with three objectives.

ples from multi-objective optimization to produce an algorithm that provides a reasonable approximation of the Pareto set, outperforms a common multi-objective optimization problem on a path-planning problem, and has guaranteed best-case performance.

Future work should extend the algorithm to include not only the weighted sum and Tchebycheff approach but also the boundary intersection approach, which results from [130] suggest might have even better diversity. MORRF* could also be made more efficient by, for example, using prior information to improve the set of sample points.

Another area of future work is to combine MORRF* with Bellman's principle of optimality. This could be done by setting a goal position as the root node in the algorithm and then generating a set of Pareto optimal paths. The algorithm should then converge to the set of Pareto optimal from any vertex in the tree to the goal.

Chapter 6

Toward Multi-Objective Path-Planning using Decomposition-based Sampling[1]

Abstract

There are usually multiple factors in determining how a robot should perform a task. For example, an autonomous driving system must consider both efficiency and safety. Consequently, multiple objectives should be included in planning paths, which changes the planning problem from finding the single best path to finding a set of non-dominant solutions. Although there exist many algorithms for multi-objective optimization, few of these algorithms apply directly to robotic path-planning and fewer still are capable of finding the set of Pareto optimal solutions.

We present MORRF* (Multi-Objective Rapidly-exploring Random Forest*) algorithm, which blends concepts from two different types of algorithms from the literature: Optimal rapidly-exploring random tree (RRT*) for efficient path finding [59] and a decomposition-based approach to multi-objective optimization [130]. The random forest uses two types of tree structures: a set of *reference trees* and a set of *subproblem trees*. Each reference tree explores a single objective, and the estimates from the set of reference trees are used to estimate what is called the *Utopia reference vector*. This vector is required by the multi-objective optimization part of the algorithm. Each subproblem tree explores the space, seeking to find an optimal solution to the subproblem created by blending different objectives. We present a theoretical analysis that demonstrates that the algorithm asymptotically produces the set of

[1] In preparation to be submitted to JAIR (The Journal of Artificial Intelligence Research). Authors are Daqing Yi, Michael A. Goodrich, Kevin D. Seppi.

Pareto optimal solutions, and use simulations to demonstrate the effectiveness and efficiency of MORRF* in approximating the Pareto set.

6.1 Introduction

Many important robot planning tasks cannot be translated into single objective to optimize. A robot in a search task may be expected to maximize the area that it covers while minimizing energy consumption and avoiding risk (see, for example [81, 124]). Similarly, a robot manipulator may need to satisfy performance criteria related to movement, joint velocities, joint accelerations, etc. [89]. This suggests a need for a multi-objective planning algorithm. This paper presents such an algorithm, focused on multi-objective path-planning.

A common approach for finding a path to a multi-objective path-planning problem is to find a path that optimizes a single objective created by a weighted sum of the multiple objectives [10, 109]. The weighting on each objective determines which path is selected. In order that the resulting path matches what is intended, a programmer, designer or human teammate must know how to assign the weights to produce a path that matches his or her intent. This requires some level of expertise and can impose high levels of cognitive workload. Moreover, when the multiple objectives are very difficult to compare or are expressed in incommensurate units, the solution that is found by a naive weighted-sum approach can be differ greatly from the operator's intent.

In response to these challenges, it is useful to find *the set of Pareto optimal solutions* to a multi-objective path-planning problem. A solution is Pareto optimal solution if there is no other solution that produces better payoffs for every objective. If an algorithm could produce a set of Pareto-optimal paths then a human could interactively explore this set to find one or more solutions that matches his or her intent. The objective of this paper is to create an algorithm that efficiently finds a Pareto-optimal set in a multi-objective path-planning problem.

Popular algorithms for multi-objective optimization [34, 130] have been applied to path-planning problems. The main challenge is that path-planning often represents the problem to be solved as a semi-structured tree with an exponential number of possible trajectories through the tree. The number of evaluations of the objective function required by

existing algorithms do not scale well when there are an exponential number of solutions. One approach to addressing this issue is to change the representation for a path by, for example, coding a path as a sequence of fixed-length line segments represented by direction [4, 52] or waypoints [89, 131] so that an evolutionary algorithm can be applied. This produces an encoding that can be "fed into" an appropriate evolutionary algorithm to search for the Pareto set. Unfortunately, these approaches do not scale well for large problems, because the number of segments required to represent the paths grows too quickly and estimating the required number of segments *a priori* is very challenging. Another approach to solving the multi-objective path-planning problem is to represent the path as a point in a very high-dimensional vector space. In this approach a path is represented as a point in a $n * d$ dimensional space formed by n d-dimensional way-points. If the number of way-points can be held constant, we can use standard approaches to explore the space. However the search can be more difficult if we allow the number of way-points, and therefor the dimensionality of the optimization problem, to vary. Indeed, we will use this to guide our solution, but the algorithm we present works when the obstacles in the path-planning space introduce discontinuities in these high-dimensional spaces, which limits the applicability of heuristic-based search approaches [111, 130].

In order to avoid the dilemma of workspace discretization (curse of dimensionality), sampling-based path-planning algorithms [67] are popular for finding feasible solutions from a start position to a goal position in continuous or very large search spaces; sampling-based approaches also work well when environments have complex obstacles. RRT [65] is one the most popular sampling-based algorithms because the tree structure tends to find solutions very efficiently. The RRT* algorithm was a recently introduced modification to RRT that is guaranteed to find an optimal path given enough sampling time [59, 60].

In this paper, we present MORRF* (Multi-Objective Rapidly exploring Random Forest*) algorithm, which can be used to find a set of Pareto optimal paths. MORRF* blends concepts from RRT* with a decomposition-based approach to multi-objective optimiza-

tion [130]. The random forest uses two types of tree structures: a set of *reference trees* and a set of *subproblem trees*. Each reference tree explores a single objective, and the estimates from the set of reference trees are used to estimate what is called the *Utopia reference vector* required by the adapted multi-objective optimization algorithm. Each subproblem tree explores the space, seeking to find an optimal solution to the subproblem created by blending different objectives.

Before proceeding with the remainder of the paper, it is useful to introduce some notation; other notation will be introduced as needed throughout the paper. Let $\boldsymbol{\lambda} = [\lambda_1, \cdots, \lambda_K]^T$ be a weighting vector such that $\sum_{k=1}^{K} \lambda_k = 1$. Let $\{c_1(\cdot), c_2(\cdot), \ldots c_K(\cdot)\}$ denote the K-element set of objective functions, let $\boldsymbol{c}(x) = [c_1(x), c_2(x), \ldots, c_K(x)]^T$, and let x denote a potential solution. Finally, let $\boldsymbol{z}^{\text{utop}} = [z_1^*, \cdots, z_K^*]^T$ denote the so-called Utopia reference vector, described in more detail in the next section.

This paper is an extension of a conference paper [125]. We introduce a new approach in subproblem creation, an adaptive weight adjustment mechanism to enhance the diversity of solutions and more complete simulations. The paper is structured as follows. Section 6.2 reviews relevant work in multi-objective optimization and sampling-based path-planning. Section 6.3 presents the MORRF* algorithm, Section 6.4 proves that the algorithm produces the Pareto frontier given enough sampling time. Section 6.5 presents the MORRF*-AWA algorithm enhances the diversity of Pareto-optimal paths by adaptively modifying subproblem trees. Section 6.6 uses simulations to illustrate the algorithm's performance.

6.2 Related Work

A solution is Pareto optimal if no other solution is better with respect to every objective. A naive search algorithm would identify Pareto-optimal solutions by comparing every solution to every other solution. This works for problems with a small, discrete set of solutions, but does not work for continuous spaces or problems with exponential numbers of potential solutions. Marler and Arora observed that "no single approach is superior [for all problems].

Rather, the selection of a specific method depends on the type of information given in the problem" [73]. We note that (single objective) path-planning in a continuous state space is often efficient using RRT* or other sampling-based approaches. This suggests that we use a discretization-based approach, ruling out, for example, variational approaches to solving the problem.

A common way to plan a path is to create a graph structure to represent a workspace. In a multi-objective planning problem, the cost of each edge becomes a vector, the dimension of which is determined by the number of objectives. Then, Multi-Objective A* [72] can be applied to find Pareto-optimal paths. The limitation of this approach is that it requires an *a priori* discretization rather than a discretization that is guided by the objectives as is done in RRT*; a coarse discretization throws away potentially valuable information and a fine discretization increases complexity and adds redundancy in the resulting graph structures. Obstacles can make it more difficult to determining which cells in the discretized space are connected to which others, especially when searching a space of more than 2 dimensions such as in planning the trajectory for a robotic manipulator. Another approach that uses an *a priori* discretization (and suffers from these limitations) is to encode a path as a sequence of directions from one cell to next cell and then using the NSGA-II algorithm to find a set of Pareto optimal solutions [4]. Constrained splines have been introduced to interpolate a sequence of way points into a trajectory that avoids obstacles [3], but the effect of the interpolation on the quality of the solution has not been evaluated.

Evolutionary algorithms can be used to find the Pareto set, but these approaches tend to be inefficient when applied to spaces with high dimensions [73]. In addition to the sorting approach used in NSGA-II, evolutionary algorithms based on the decomposition method have also been proposed [34]. Approximating a path by a fixed-number sequence of waypoints in the workspace enables converting a path-planning problem into a high-dimension point-space optimization. Multi-objective evolutionary algorithms (e.g. Multi-Objective PSO [131]) can then be introduced to solve the problems. For such spaces, small deviations

in possible solutions may need to be considered in order to find an optimal solution, but this means exploring many possible solutions for problems with non-linearities or multiple local maxima. A path in a fixed-length search tree of depth d can be considered as a point in $\Re^d$, so tree-based approaches followed by an evolutionary "fine-tuning" stage risk running into the problems just listed with evolutionary approaches.

In contrast to searching through and comparing solutions in order to find the Pareto optimal set, decomposition-based methods provide an attractive alternative. In this paper we use a decomposition-based method similar to MOEA-D [130]. MOEA-D is an algorithm that decomposes a multi-objective optimization problem into a set of subproblems. Each subproblem $g_m(x)$ uses a weighed combination of the objectives to find specific points in the Pareto set or to guide the search for such points. Three types of decomposition methods have been used in prior work [130].

$$\arg\min_x \sum_{k=1}^{K} \lambda_k c_k(x) \qquad \text{Weighted sum} \tag{6.1}$$

$$\arg\min_x \max_{1\leq k\leq K} \{\lambda_k \left(|c_k(x) - \boldsymbol{z}_k^{\text{utop}}|\right)\} \qquad \text{Tchebycheff} \tag{6.2}$$

$$\arg\min_x \{d \mid \boldsymbol{z}^{\text{utop}} - \boldsymbol{c}(x) = d\boldsymbol{\lambda}\} \quad \text{Boundary intersect} \tag{6.3}$$

A general boundary-intersection is proposed in [130], which is an improved version of the boundary-intersection approach. The subproblem $g_m(x)$ is defined in Equation (6.4).

$$g_m(x) = d_1 + \theta d_2, \tag{6.4}$$

in which $d_1 = \frac{||(\boldsymbol{c}(x) - \boldsymbol{z}^{\text{utop}})^T \boldsymbol{\lambda}||}{||\boldsymbol{\lambda}||}$ and $d_2 = ||(\boldsymbol{c}(x) - (\boldsymbol{z}^{\text{utop}} + d_1\boldsymbol{\lambda})||$. The solutions generated by each method are a subset of the Pareto optimal set.

Sampling-based path-planning can work effectively in continuous spaces. The RRT (Rapidly exploring Random Tree) has been a popular algorithm, which efficiently explores the space by randomly sampling the search space; this algorithm tends to work well in the

presence of complex obstacles. Unfortunately, RRT fails to guarantee optimality [59]. In response, the RRT* algorithm was proposed, which uses a *Rewire* process to gradually update the tree structure when new samples of the space indicate that this is needed. RRT* is asymptotically optimal [59, 60].

We now present an algorithm for multi-objective path-planning problem that decomposes the multi-objective optimization into a set of subproblems and introduces the power of RRT* to solve each subproblem.

6.3 Multi-Objective Rapidly-exploring Random Forest*

In this section, we present an algorithm that explores the solution space using RRT*-based tree structures but uses multiple trees in the spirit of decomposition-based multi-objective optimization. Because a set of trees are constructed in the exploration process, we call the algorithm MORRF* (Multi-Objective Rapidly exploring Random Forest*).

Consider a multi-objective path planning problem defined on a bounded, connected open set $X \subset \mathbb{R}^d$ of possible solutions, and K different objectives $\{c_1(\cdot), c_2(\cdot), ...c_K(\cdot)\}$. Without loss of generality, assume that the objective is to minimize these functions. Since the Pareto optimal set is not enumerable, the *goal is to find a representative, finite (M-element) subset of the Pareto optimal set.*

Definition 1. ***Multi-Objective Path-Planning*** *Consider a bounded, connected open set $X \subset \mathbb{R}^d$, an obstacle space X_{obs}, an initial state x_{init}, and a goal region X_{goal}. Consider the set of K objectives determined by a vector function $\boldsymbol{c}(\cdot) = [c_1(\cdot), \ldots, c_K(\cdot)]^T$ defined by $\boldsymbol{c} : \mathbb{X} \to \mathbb{R}^K$. Denote the obstacle-free space by $X_{free} = X \setminus X_{obs}$. Note that $\boldsymbol{c}$ is defined for all points in X both those in free space and obstacle space.*

Define a path *in X as a continuous curve parameterized by s, denoted by $\sigma : [0, s] \to X$. Define the cost of the path as the vector-valued function $\boldsymbol{c}(\sigma) = \int_\sigma \boldsymbol{c}(x)ds$. The goal is to find M Pareto optimal paths $\sigma^* \in \Sigma^*$ that (a) $\forall \tau \in [0, s], \sigma^*(\tau) \in X_{free}$; (b) $\sigma^*(0) = x_{init}$*

and $\sigma^*(s) = X_{goal}$*; and (c) there does not exist* σ *that (i)* $\forall k \in K, c_k(\sigma) \leq c_k(\sigma^*)$ *and (ii)* $\exists k' \in K, c_{k'}(\sigma) < c_{k'}(\sigma^*)$.

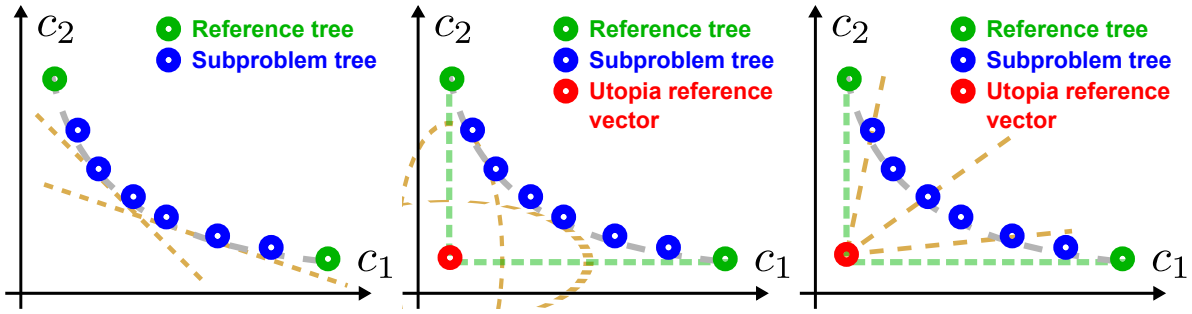
Note that the last condition enforces a very strict form of payoff dominance; extensions to the algorithm can be made for weaker forms of payoff dominance. Without loss of generality, we could set $s = 1$ for all paths, but we retain the s-parameter because it is a convenient way of signaling that two paths may be different lengths.

Adopting the idea from the MOEA-D algorithm [130], the M elements in the solution set Σ^* will be obtained by decomposing the multi-objective problem into M subproblems. We use RRT*-based sampling to solve each subproblem. We will need one type of RRT* structure to explore in each objective and another type of RRT* structure to find paths that minimize each subproblem. Thus, there are two types of tree structures used for the optimization process.

- Each *reference tree* explores using a single objective $c_k(x), k \in K$. The cost of each vertex is calculated using the k^{th} objective function.
- Each *subproblem tree* explores a subproblem $g_m(x \mid \lambda_m, \boldsymbol{z}^{\text{utop}}), m \in M$. The cost associated with each vertex is calculated using $g_m(x)$ defined by the corresponding approach.

Thus K reference trees are used, one each to explore the minimum of each objective, and M subproblem trees are used, one for each weighting vector, λ_m, which is described below. The K reference trees and M subproblem trees constitute the exploration forest.

Figure 6.1 illustrates the three approaches of decomposition in two objective problem. Recall from the introduction that $\boldsymbol{\lambda} = [\lambda_1, \cdots, \lambda_K]^T$ is a weighting vector such that $\sum_{k=1}^{K} \lambda_k = 1$. In creating subproblems, $\boldsymbol{\lambda}$ is randomly sampled from a K-dimension simplex. In the weighted-sum approach, a single objective is created by summing all the objectives by weights from $\boldsymbol{\lambda}$. The weighted-sum approach in a two-objective problem is illustrated in Figure 6.1a. In the Tchebycheff and boundary-intersection approaches, a Utopia reference vector $\boldsymbol{z}^{\text{utop}}$



(a) Weighted-sum (b) Tchebycheff (c) Boundary-intersection

Figure 6.1: Approaches of subproblem creation.

in identified in the objective space. The Tchebycheff approach seeks to find the solution $x^{the} \in X$ given by $x^{te} = \arg\min_x \max_{1 \leq k \leq K} \{\lambda_k | c_k(x) - \boldsymbol{z}_k^{\text{utop}}|\}$. As illustrated in Figure 6.1b, the Utopia reference vector is defined as that point in cost space that would be obtained if it were possible to find a solution that produced the minimum value for all objectives, that is the k^{th} element of $\boldsymbol{z}^{\text{utop}}$ is given by $\boldsymbol{z}_k^{\text{utop}} = \arg\min_{x \in X} c_k(x)$. It is called the *Utopia reference vector* because it represents a point that very best that could conceivably be achieved for an ideal point in the solution space. Similarly, the boundary-intersection approach uses the Utopia reference vector but defines distance as $\boldsymbol{z}^{\text{utop}} - \boldsymbol{c}(x) = d\boldsymbol{\lambda}$. Solving the subproblem equals minimizing the distance d. An example is illustrated in Figure 6.1c.

Given this brief overview of different decomposition methods, we are now in a position to describe the main flow of the MORRF* algorithm; see Algorithm 1. The first four steps initialize two types of trees, the reference tree with vertices and edges denoted by V_r and E_r, respectively, and the subproblem tree with vertices and edges denoted by V_s and E_s, respectively. Each reference and subproblem tree is a collection of edges and vertices, $G_r = (V_r, E_r)$ and $G_s = (V_s, E_s)$, respectively, and the collection of reference trees and subproblem trees is denoted by $\boldsymbol{G}_r = \{G_r : r \in \{1, \ldots, K\}\}$ and $\boldsymbol{G}_s = \{G_s : s \in \{q, \ldots, M\}\}$.

Because this is a path-planning problem, each vertex in each tree corresponds to a physical location in workspace. In the examples used later, the workspace is a plane, so the distances will all be computed using Euclidean distance. Note that each tree, reference and subproblem, uses the same set of vertices, meaning they all share the same points in

workspace. The differences between the trees is the edge set; each reference tree and each subproblem tree has a different way of connecting the vertices. An example is given in Figure 6.2.

The Utopia reference vectors are estimated from reference trees. Because all the trees use the same sampled points in generating vertices, the expansion of reference trees provides the estimated Utopia reference vector of newly sampled point. The estimated Utopia reference vector is used in the calculation of subproblems. It is noticeable that the creation of subproblems in the weighted-sum approach does not depend on Utopia reference vectors. Reference trees can only be used as subproblems with particular weights. The k-th reference tree equals to a subproblem tree with the normalized weight $\boldsymbol{\lambda}$, in which the sum over λ_k equals one.

For each $i \in \{1, \ldots, N\}$, where N is the maximum number of iterations allowed by the algorithm, a new position, x_{rand} is generated by randomly sampling from the workspace. The set of vertices is then searched to find that vertex whose position is nearest to the random point; since all trees share the same set of vertices, any tree $G \in \boldsymbol{G}_r \cup \boldsymbol{G}_s$ may be used to find the nearest point. The location of this vertex is labeled $x_{nearest}$. It is possible that the distance between the newly sampled position and the vertex nearest to this position is very large. A tolerance parameter, η, is used to guarantee that the distance between the vertices in the tree and the new position is within some exploration tolerance; this is accomplished in the Steer method. If the distance between x_{rand} and $x_{nearest}$ is less than the tolerance η, then $x_{new} = x_{nearest}$. Otherwise, a line segment between x_{rand} and $x_{nearest}$ is constructed and the point on this line segment that is distance η from $x_{nearest}$ becomes the new point x_{new}. The process of finding x_{new} is represented in Figure 6.2, with the dashed circle around the new point indicating η.

Given the location of a new vertex for each tree, edges are added, potentially different edges for each reference and each subproblem tree. We require that each new vertex be connected to some existing vertex, so a feasibility check is added using the ObstacleFree

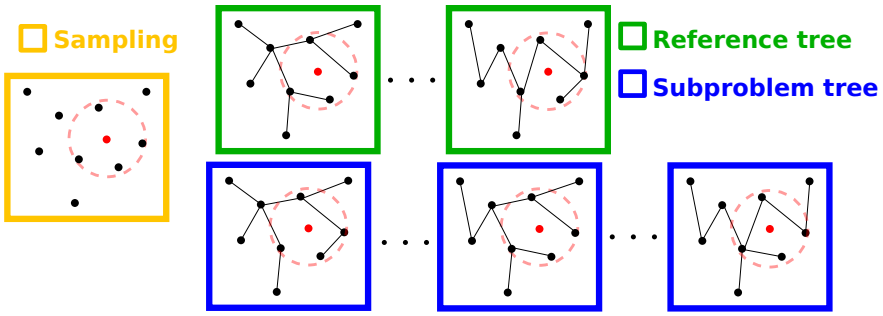


Figure 6.2: Rapidly exploring process

method. This method checks whether the line segment connecting x_{new} and $x_{nearest}$ enters obstacle space. If it does, no new vertices and no new edges are added, and another iteration of the algorithm begins. This check guarantees that when the algorithms are called that determine which edge to add to the reference tree or subproblem tree, there is always at least one possible edge that can be added; this is consistent with the RRT* algorithm.

We now define several functions, using appropriately modified definitions from [59].

- SAMPLE(): Returns independent uniformly distributed samples from X_{free}.
- NEAREST(): Returns a position of the vertex whose position is closest to point x. $\text{NEAREST}(G = (V, E), x) = \arg\min_{v \in V} \|x - v\|$.
- STEER(): Given two points x and y, returns a point z on the line segment from x to y that that is no greater than η from y. $\text{STEER}(\ x, y, \eta\ ) = \arg\min_{z \in \mathbb{R}^d, \|z-x\| \leq \eta} \|z - y\|$.
- OBSTACLEFREE(x, x'): Returns True if $[x, x'] \subset X_{free}$, which is the line segment between x and x' lies in X_{free}.

The exploration at each iteration is given in Algorithm 1. Like RRT*, when the algorithm stops, each reference tree and subproblem tree returns a path, and the set of all these paths forms the solution set.

Because the convergence of the tree structure in RRT* means that the path from the root to any vertex is an optimal path of the defined cost. We use the same sampling position to extend all the trees in one iteration. Thus the vertices in the reference trees could be used as reference for estimating the cost in constructing the subproblem trees.

Algorithm 1 MORRF*

1: **for each** $V_r \in \boldsymbol{V}_r$ **do**
2: $V_r \leftarrow \{x_{init}\}$; $E_r \leftarrow \emptyset$; $i \leftarrow 0$
3: **for each** $V_s \in \boldsymbol{V}_s$ **do**
4: $V_s \leftarrow \{x_{init}\}$; $E_s \leftarrow \emptyset$; $i \leftarrow 0$
5: **while** $i < N$ **do**
6: $x_{rand} \leftarrow$ SAMPLE (i) ; $i \leftarrow i + 1$
7: G is arbitrary graph from $\boldsymbol{G}_r \cup \boldsymbol{G}_s$.
8: $x_{nearest} \leftarrow$ NEAREST(G, x_{rand})
9: $x_{new} \leftarrow$ STEER$(x_{nearest}, x_{rand}, \eta)$
10: **if** OBSTACLEFREE$(x_{nearest}, x_{new})$ **then**
11: **for each** $G_r \in \boldsymbol{G}_r$ **do**
12: $G_r \leftarrow$ EXTEND$_{Ref}$ $(G_r, x_{new}, x_{nearest}, r)$
13: **for each** $G_s \in \boldsymbol{G}_s$ **do**
14: $G_s \leftarrow$ EXTEND$_{Sub}$ $(G_s, x_{new}, x_{nearest}, s)$

We can now discuss how edges are added to the reference and subproblem trees. As illustrated in Figure 6.2, first layer, edges to the reference trees are added before the edges to the subproblem trees. This allows us to estimate the Utopia reference vector using the path costs for each reference tree, each reference tree returning a path that approximates the minimum cost for one objective. The Utopia reference vector is then used to determine which edges should be added for each subproblem. Stated another way, the edges added to each tree are determined by the definition of the fitness (cost function) for each tree.

Like all the sampling-based optimization, the random positions are uniformly sampled from the workspace. This means that all the tree have equivalent vertices constructed from the same position set, but they are connected by different measurements of the costs, either a single objective or a cost from subproblem definition.

Consider the first layer in Figure 6.2, which shows the exploration process for the reference trees. When a new position is obtained (red dot in Figure 6.2), all reference trees add a vertex that corresponds to this new location. Each reference tree then connects this new vertex to existing nodes by "rewiring" a set of neighboring vertices within a specified radius (red dash circle in Figure 6.2). The process of rewiring consists of adding edges

between existing vertices and the new vertex. This is done using the EXTEND method, given in Algorithm 2.

The idea of the algorithm is to add the new vertex, x_{new}, to the graph and then look for existing vertices in the graph that are close to the new vertex. Line 1 checks to see if the randomly generated position is precisely on top of the position of a vertex already in the graph. If so, no edges are added to the graph; this prevents self-loops, and is necessary because of finite precision numerical representations.

Otherwise, the new vertex is added to the graph and all vertices in the graph within a particular radius are identified. The radius is determined by the current number of vertices in the graph. The radius and volume are precisely defined in the NEAR method, given below. The equation in this method is a standard equation from the RRT* algorithm. It is based on the expected density of the vertices in the graph, given as a spatial distribution in d-dimensions, and is an important element in the convergence guarantees of RRT*.

Because of the way that x_{new} is defined, we are guaranteed that $|X_{near}| \geq 1$, that is, there is at least one vertex "near" the new position.

Lines 5-10 then find the vertex, x_{min} that produces the minimum obstacle-free cost and adds the edge between x_{min} and x_{new} to the graph. Because of the check in in Algorithm 1, there is always at least one such edge. Observe that this cost is computed using the k^{th} cost function, which is, after all, the role of the reference tree.

Lines 11-17 "rewire" the graph. This only occurs if there are more than two vertices in the neighborhood of x_{new}. The lines step through every vertex x_{near} in the neighborhood (other than the minimum cost vertex – see line 11) and compares (a) the cost of the current path through the tree to x_{near} to (b) the cost of the same path to the parent x_{parent} of x_{near} plus the cost from x_{parent} to the new node x_{new}. If the cost of the path to x_{new} is less than the cost of the path to x_{near}, the tree is rewired so that the parent forgets x_{near} and connects instead to x_{new} – see lines 16-17.

The precise definitions of the methods used in the Algorithm 2 are given below.

Algorithm 2 EXTEND$_{Ref}$ $(G, x_{new}, x_{nearest}, k)$

1: **if** $x_{new} = x_{nearest}$ **then return** $G = (V, E)$
2: $V' \leftarrow V \cup \{x_{new}\}$
3: $x_{min} \leftarrow x_{nearest}$
4: $X_{near} \leftarrow \text{NEAR}(G, x_{new}, |V|)$
5: **for each** $x_{near} \in X_{near}$ **do**
6: **if** OBSTACLEFREE(x_{new}, x_{near}) **then**
7: $c'_k \leftarrow \text{COST}_k(x_{near}) + c_k(\text{ LINE}(x_{near}, x_{new})\ )$
8: **if** $c'_k < \text{COST}_k(x_{new})$ **then**
9: $x_{min} \leftarrow x_{near}$
10: $E' \leftarrow E' \cup \{(x_{min}, x_{new})\}$
11: **for each** $x_{near} \in X_{near} \setminus \{x_{min}\}$ **do**
12: **if** OBSTACLEFREE(x_{new}, x_{near}) **then**
13: $c'_k \leftarrow \text{COST}_k(x_{new}) + c_k(\text{ LINE}(x_{new}, x_{near})\ )$
14: **if** $c'_k < \text{COST}_k(x_{near})$ **then**
15: $x_{parent} \leftarrow \text{PARENT}(x_{near})$
16: $E' \leftarrow E' \setminus \{(x_{parent}, x_{near})\}$
17: $E' \leftarrow E' \cup \{(x_{new}, x_{near})\}$
return $G' = (V', E')$

- NEAR(G, x, η): Returns a set of all vertices within the closed ball of radius r_n centered at x, in which $r_n = \min\{(\frac{\gamma}{\xi_d}\frac{\log n}{n})^{1/d}, \eta\}$. The volume of the ball is $\min\{\gamma\frac{\log n}{n}, \xi_d\eta^d\}$.
- LINE$(x, x') : [0, s] \leftarrow X_{free}$ denotes the path defined by line segment from x to x'.
- COST(v): Returns the cost of the unique path (because G is a tree) from x_{init} to the vertex $v \in V$. COST$(x_{\text{init}}) = 0$.

Consider the second layer in Figure 6.2, which illustrates how the subproblem trees "rewire" to connect to the new vertex. The Utopia reference vector, $\hat{\boldsymbol{z}}_k^{\text{utop}}$ is defined as the k-dimensional vector constructed from each reference tree. The minimum cost of each path from the starting vertex over any other vertex is computed for each reference tree. The Utopia reference vector is the vector of these costs. Using the Utopia reference vector, each subproblem tree connects its new vertex and rewires neighboring vertices that fall within the radius threshold. Algorithm 3 precisely follows Algorithm 2 except that instead of computing the cost using one of the objectives, the cost is computed using the Tchebycheff, weighted

sum, or boundary intersection method. Each of the m^{th} subproblem trees corresponds to a different weighting vector λ_m. This is performed using the FITNESS method.

Algorithm 3 EXTEND$_{Sub}$ $(G, x_{new}, x_{nearest}, m)$

if $x_{new} = x_{nearest}$ **then return** $G = (V, E)$
$V' \leftarrow V' \cup \{x_{new}\}$
$x_{min} \leftarrow x_{nearest}$
$X_{near} \leftarrow \text{NEAR}(G, x_{new}, |V|)$
for each $x_{near} \in X_{near}$ **do**
    if OBSTACLEFREE(x_{new}, x_{near}) **then**
        $\boldsymbol{c}' \leftarrow \text{COST}(x_{near}) + \boldsymbol{c}(\text{ LINE}(x_{near}, x_{new}))$
        $\eta' = \text{FITNESS}(\boldsymbol{c}', \hat{\boldsymbol{z}}^{\text{utop}} \mid \lambda_m)$
        $\boldsymbol{c}_{new} = \text{COST}(x_{new})$
        $\eta_{new} = \text{FITNESS}(\boldsymbol{c}_{new}, \hat{\boldsymbol{z}}^{\text{utop}} \mid \lambda_m)$
        if $\eta' < \eta_{new}$ **then**
            $x_{min} \leftarrow x_{near}$
$E' \leftarrow E' \cup \{(x_{min}, x_{new})\}$
for each $x_{near} \in X_{near} \setminus \{x_{min}\}$ **do**
    if OBSTACLEFREE(x_{new}, x_{near}) **then**
        $\boldsymbol{c}' \leftarrow \text{COST}(x_{new}) + \boldsymbol{c}(\text{ LINE}(x_{new}, x_{near}))$
        $\eta' = \text{FITNESS}(\boldsymbol{c}', \hat{\boldsymbol{z}}^{\text{utop}} \mid \lambda_m)$
        $\boldsymbol{c}_{near} = \text{COST}(x_{near})$
        $\eta_{near} = \text{FITNESS}(\boldsymbol{c}_{near}, \hat{\boldsymbol{z}}^{\text{utop}} \mid \lambda_m)$
        if $\eta' < \eta_{near}$ **then**
            $x_{parent} \leftarrow \text{PARENT}(x_{near})$
            $E' \leftarrow E' \setminus \{(x_{parent}, x_{near})\}$
            $E' \leftarrow E' \cup \{(x_{new}, x_{near})\}$
return $G' = (V', E')$

The FITNESS method computes costs using one of the cost functions in Equations (6.1)-(6.3). Different values of λ_m are obtained using the pattern in the MOEA-D algorithm: (a) pre-deterimining the range of the K-cost functions, $\{c_k() : 1 \ldots K\}$ and (b) sampling from the K-dimensional hypercube defined by these ranges. The M samples from this hypercube can be obtained by either creating a uniform (hyper)-grid or by doing uniform sampling across the space.

6.4 Theoretical Analysis

The analysis depends on the following restrictions on the cost functions and obstacle placement required by the RRT* algorithm [59]. We claim without argument that the cost functions and obstacle placements used in the simulation studies presented later in the paper satisfy the restrictions.

Assumption 1. *(Additivity of the objective functions) For a path constructed by composing two other paths (to create a discontinuous path),* $\forall k \in K, \sigma_1, \sigma_2 \in X_{\text{free}}$, $c_k(\sigma_1 \circ \sigma_2) = c_k(\sigma_1) + c_k(\sigma_2)$.

Assumption 2. *(Continuity of the cost functions) For all* $k \in K$*, the cost function* c_k *is Lipschitz continuous, that is, for all paths* $\sigma_1 : [0, s_1] \to X_{free}$ *and* $\sigma_2 : [0, s_2] \to X_{free}$*, there exists a constant* $\kappa(k) \in \mathbb{R}_+ \cup \{0\}$ *such that* $|c_k(\sigma_1) - c_k(\sigma_2)| \leq \kappa(k) \sup_{\tau \in [0,1]} \|\sigma_1(\tau s_1) - \sigma_2(\tau s_2)\|$.

Assumption 3. *(Obstacle spacing) There exists a constant* $\delta \in \mathbb{R}_+$ *such that* $\forall x \in X_{free}$ *,* $\exists x' \in X_{free}$ *such that*

- *the* δ*-ball centered at* x' *lies inside* X_{free}*;*
- x *lies inside the* δ*-ball centered at* x'*.*

6.4.1 Optimality of the weighted-sum approach

Lemma 1 states that the solutions found by the weighted-sum approach are weakly Pareto-optimal.

Lemma 1. *Any solution of Equation* (6.1) *is weakly Pareto-optimal.*

Proof. Let the weighting vector $\boldsymbol{\lambda}$ be arbitrary subject to $\forall k \ \lambda_k \geq 0$, and let $\sigma^* = \sigma^*(\boldsymbol{\lambda})$ be a solution given that weighting vector in Equation (6.1). By definition,

$$\sigma^* = \sum_{k=1}^{K} \lambda_k c_k(x). \tag{6.5}$$

Assume that the path σ^* is not weakly Pareto optimal. Then there exist another path σ^o that strictly dominates σ^*, which means $\forall k\ c_k(\sigma^o) < c_k(\sigma^*)$. It equals to that $\forall k\ \lambda_k c_k(\sigma^o) < \lambda_k c_k(\sigma^*)$. Summing all $k \in K$, we can have $\sum_{k=1}^{K} \lambda_k c_k(\sigma^o) < \sum_{k=1}^{K} \lambda_k c_k(\sigma^*)$. It contradicts to Equation (6.5). □

Conversely, each Pareto-optimal solution can be determined by a weight when the multi-objective problem is convex.

Lemma 2. *When the multi-objective problem is convex, if σ^* is Pareto optimal then there exists a weighting vector $\boldsymbol{\lambda}$, where $\forall k\ \lambda_k \geq 0$ and $\sum_{k=1}^{K} \lambda_k = 1$, such that σ^* is a solution of Equation* (6.1)*.*

Proof. The proof depends on the definition of ε-constraint and the generalized Gordan theorem. The details are given in P.79 of [82]. □

By Lemma 1 and Lemma 2, we can conclude that the solutions of Equation (6.1) are Pareto-optimal if the multi-objective optimization problem is convex.

Theorem 1. *When the multi-objective problem is convex, a path is Pareto optimal if and only if it is a solution to Equation* (6.1) *for some weight vector.*

Proof. By Lemma 1, we know that each solution to Equation (6.1) is a solution in Pareto optimal set. By Lemma 2, we know that each solution in Pareto optimal set, there exits a weight that makes the solution an answer to Equation (6.1). Thus, we know that the Pareto optimal set $\{\sigma^*\}$ is the range of the function value defined by Equation (6.1). □

Because the subproblem in the weighted-sum approach is a single-objective optimization problem, the solution can be guaranteed to be asymptotically optimal by the property of RRT* [59]. Define $\Sigma_i^{\text{MORRF}^*}$ as the set of solutions returned by the MORRF* algorithm after running i-th iteration. Thus, we have Theorem 2.

Theorem 2. *Given Assumptions 1-4, when the multi-objective problem is convex, the solution generated by the weighted-sum approach of MORRF* converges to a subset of the Pareto optimal set almost surely, i.e.* $P(\lim_{i\to\infty} \Sigma_i^{MORRF^*} \subset \Sigma^*) = 1$.

6.4.2 Optimality of the Tchebycheff approach

Lemma 3. *If the Utopia reference vector satisfies* $\forall k, \sigma\ z_k^{\text{utop}} \leq c_k(\sigma)$*, then any solution of Equation* (6.2) *is Pareto optimal.*

Proof. The proof is by contradiction. Let the weighting vector $\boldsymbol{\lambda}$ be arbitrary subject to $\forall k\ \lambda_k \geq 0$, and let $\sigma^* = \sigma^*(\boldsymbol{\lambda})$ be a solution given that weighting vector in Equation (6.2). By definition,

$$\sigma^* = \arg\min_{\sigma} \max_{k\in K} \lambda_k |c_k(\sigma) - z_k^{\text{utop}}|. \tag{6.6}$$

Assume that the path σ^* is not Pareto optimal. Then there exist another path σ^o that dominates σ^* and the Utopia reference vector that satisfies $\forall k \in K$, $z_k^{\text{utop}} \leq c_k(\sigma)$, it follows that $\forall k \in K, z_k^{\text{utop}} \leq c_k(\sigma^o) \leq c_k(\sigma^*)$ and $\exists k' \in K, z_k^{\text{utop}} \leq c_{k'}(\sigma^o) < c_{k'}(\sigma^*)$. These equations imply

$$\begin{aligned}
&\forall k \in K, \quad \lambda_k |c_k(\sigma^*) - z_k^{\text{utop}}| \geq \lambda_k |c_k(\sigma^o) - z_k^{\text{utop}}|; \\
&\exists k' \in K, \quad \lambda_{k'} |c_{k'}(\sigma^*) - z_k^{\text{utop}}| > \lambda_{k'} |c_{k'}(\sigma^o) - z_k^{\text{utop}}|;
\end{aligned}$$

which yields the following contradiction to Equation (6.6):

$$\max_{k\in K} \lambda_k |c_k(\sigma^*) - z_k^{\text{utop}}| > \max_{k\in K} \lambda_k |c_k(\sigma^o) - z_k^{\text{utop}}|.$$

□

Before presenting the following lemma and proof, note that we modify the definition of the Utopia vector to satisfy a technical requirement: $z_k^{\text{utop}} = \inf_{\sigma\in X_{\text{free}}} c_k(\sigma) - \delta$ for δ some small real number.

Lemma 4. *If σ^* is Pareto optimal then there exists a weighting vector $\boldsymbol{\lambda}$, where $\forall k\ \lambda_k \geq 0$ and $\sum_{k=1}^{K} \lambda_k = 1$, such that σ^* is a solution of Equation* (6.2).

Proof. This is a proof by construction over cases. When σ^* is Pareto optimal, there exist two cases: (a) $\exists k, c_k(\sigma^*) = z_k^{\text{utop}}$ and (b) $\forall k, c_k(\sigma^*) > z_k^{\text{utop}}$.

Case (a): $\exists k, c_k(\sigma^*) = z_k^{\text{utop}}$

Define $P(\sigma^*) = \{j \mid c_j(\sigma^*) = z_j^{\text{utop}}\}$ and let $\overline{P} = \{1, \ldots, K\} \setminus P$. Define the weight vector $\boldsymbol{\lambda}$ as $\forall k \in P(\sigma^*), \lambda_k = \frac{1}{|P|}$ and $\forall k \in \overline{P}(\sigma^*), \lambda_k = 0$. For these weights, Equation (6.2) returns a set of solution paths, all of which have the same cost for the k-cost functions when $k \in P$ but different possible costs for $k \in \overline{P}$. σ^* is trivially in this set of solution paths.

Case (b): $\forall k, c_k(\sigma^*) > z_k^{\text{utop}}$

For all k, define the weights as $\lambda_k = \frac{\ell_k}{\sum_{j=1}^{K} \ell_j}$, where $\ell_k = \frac{1}{|c_k(\sigma^*) - z_k^{\text{utop}}|}$. The Tchebycheff cost (Equation (6.2)) becomes

$$g^{te}(\sigma^*) = \max_{k \in K} \frac{|c_k(\sigma^*) - z_k^{\text{utop}}|}{|c_k(\sigma^*) - z_k^{\text{utop}}|} \frac{1}{\sum_{j=1}^{K} \ell_j} = \frac{1}{\sum_{j=1}^{K} \ell_j}$$

Given any other path σ, we can represent the Tchebycheff cost as follows:

$$\begin{aligned}
g^{te}(\sigma) &= \max_{k \in K} \frac{\ell_k}{\sum_{j=1}^{K} \ell_j} |c_k(\sigma) - z_k^{\text{utop}}| \\
&= \frac{1}{\sum_{j=1}^{K} \ell_j} \max_{k \in K} \frac{|c_k(\sigma) - z_k^{\text{utop}}|}{|c_k(\sigma^*) - z_k^{\text{utop}}|} \\
&= \left(\sum_{j=1}^{K} \ell_j \right)^{-1} \max_{k \in K} \left| \frac{c_k(\sigma) - z_k^{\text{utop}}}{c_k(\sigma^*) - z_k^{\text{utop}}} \right| \qquad (6.7) \\
&= \frac{1}{\sum_{j=1}^{K} \ell_j} \max_{k \in K} \left| \frac{c_k(\sigma) - c_k(\sigma^*) + c_k(\sigma^*) - z_k^{\text{utop}}}{c_k(\sigma^*) - z_k^{\text{utop}}} \right| \\
&= \frac{1}{\sum_{j=1}^{K} \ell_j} \max_{k \in K} \left| 1 + \frac{c_k(\sigma) - c_k(\sigma^*)}{c_k(\sigma^*) - z_k^{\text{utop}}} \right|
\end{aligned}$$

Because σ^* is Pareto optimal, $[\exists k' \in K, c_{k'}(\sigma) > c_{k'}(\sigma^*)] \vee [\forall k \in K, c_{k'}(\sigma) = c_{k'}(\sigma^*)]$ for any σ. As $\forall k, c_k(\sigma^*) > z_k^{\text{utop}}$, we have $\forall k, c_k(\sigma^*) - z_k^{\text{utop}} > 0$. This implies

$$\exists k' \in K, \exists k' \in K, \frac{c_{k'}(\sigma) - c_{k'}(\sigma^*)}{c_{k'}(\sigma^*) - z_{k'}^{\text{utop}}} \geq 0, \tag{6.8}$$

which, in turn, implies that

$$\max_{k \in K} \left| 1 + \frac{c_k(\sigma) - c_k(\sigma^*)}{c_k(\sigma^*) - z_k^{\text{utop}}} \right| \geq 1. \tag{6.9}$$

Therefore,

$$g^{te}(\sigma) \geq \frac{1}{\sum_{j=1}^{K} \ell_j} = g^{te}(\sigma^*). \tag{6.10}$$

It means,

$$\forall \sigma, g^{te}(\sigma) > g^{te}(\sigma^*). \tag{6.11}$$

Thus, σ^* is a solution to Equation (6.2).

□

By Lemma 3 and Lemma 4, we can conclude Theorem 3, which uses a proof similar to the proof of Theorem 1.

Theorem 3. *A path is Pareto optimal if and only if it is a solution to Equation* (6.2) *for some weight vector.*

Theorem 3 implies that we can use the Tchebycheff method to find the Pareto set for the multi-objective path-planning problems. In terms of the MORRF* algorithm, we can sample the Pareto set by selecting weights, forming a subproblem that can be solved using RRT*. The next question that needs to be answered is whether the subproblem tree can find the optimal solution of its assigned subproblem.

The way that the RRT* algorithm works is that it incrementally constructs a tree from a root position. New nodes are constructed by randomly sampling points that are near

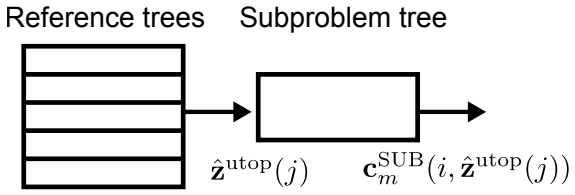


Figure 6.3: The dependency of the trees in MORRF*.

to other locations in the tree, and then wiring the new node and rewiring nearby nodes in the tree so that the path from root to the position of the new node is minimal. The cost of the path from the position of the root node to the positions of every other node converges to the minimal possible cost between the positions as the number of iterations approaches infinity. We restate this as a lemma, and note that it corresponds exactly to that given for Theorem 22 in [59].

Lemma 5. *Given Assumptions 1-3, the cost of the minimum cost path from the root to any vertex in RRT* converges to the optimal cost almost surely.*

Lemma 5 and Theorem 3 imply that each reference tree converges to the optimal path from the root to any node in the tree, including a node arbitrarily close to the goal node. This means that the costs returned by those trees for the path from the start to the goal for the cost function c_k converges to the k^{th} element of the Utopia reference vector $\boldsymbol{z}^{\text{utop}}$. We state this as a lemma.

Lemma 6. *Given Assumptions 1-3, the cost of the minimum cost path from the root to any vertex in k^{th} reference tree converges to z_k^* almost surely.*

We now turn to the proof that the subproblem trees converge to paths in the Pareto set. The proof of this claim requires that we know $\boldsymbol{z}^{\text{utop}}$ to compute the Tchebycheff cost associated with the cost used in the subproblem. If we knew that the reference trees had already converged to $\boldsymbol{z}^{\text{utop}}$, then we could simply instantiate Lemma 5. Unfortunately, the reference trees are converging at the same time that the subproblem trees are converging. We now address this problem.

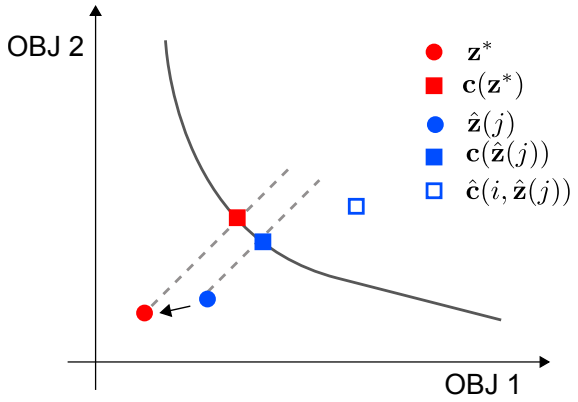


Figure 6.4: The convergence process of trees.

Let $\hat{\boldsymbol{z}}^{\text{utop}}(v; i)$ denote the approximate Utopia reference vector for position v on iteration i, estimated by the cost from the root to position x from the k-reference trees. Recall that the m^{th} subtree attempts to generate a solution to Equation (6.2) for a given weight vector $\boldsymbol{\lambda}^m$. Let

$$\boldsymbol{c}_m^{\text{SUB}}(\boldsymbol{z}) = \arg\min_x \max_{k \in K} \lambda_{m,k} |x_k - \boldsymbol{z}_k| \tag{6.12}$$

denote the cost vector in m^{th} subproblem tree given the reference vector $\boldsymbol{z}$ and let $\hat{\boldsymbol{c}}_m^{\text{SUB}}(i, \boldsymbol{z})$ denote its estimation at iteration i. A subproblem tree obtains $\hat{\boldsymbol{z}}^{\text{utop}}(v)$ for vertex v in the reference trees and generate the corresponding $\boldsymbol{c}_m^{\text{SUB}}(v; i, \hat{\boldsymbol{z}}^{\text{utop}}(v))$ at iteration i for either adding new vertex or rewiring an existing vertex. This forms a cascade structure from the reference trees to the subproblem tree. By Lemma 6, we have the convergence of the reference trees.

We introduce Assumption 4 to get Lemma 7.

Assumption 4. *(Lipschitz continuity)* $\boldsymbol{c}_m^{\text{SUB}}(\boldsymbol{z})$ *in Equation* (6.12) *and its estimation* $\hat{\boldsymbol{c}}_m^{\text{SUB}}(i, \boldsymbol{z})$ *are Lipschitz continuous, i.e.* $\|\boldsymbol{c}_m^{\text{SUB}}(\boldsymbol{z}_a) - \boldsymbol{c}_m^{\text{SUB}}(\boldsymbol{z}_b)\| \leq K \|\boldsymbol{z}_a - \boldsymbol{z}_b\|$.

Lemma 7. *Given Assumptions 1-4 , the cost of the solution of* m^{th} *subproblem tree converges to the corresponding cost of the* m^{th} *subproblem* $\boldsymbol{c}_m^*$ *almost surely.*

Proof. By Lemma 6, we have

$$\lim_{j \to \infty} \|\boldsymbol{z}^* - \hat{\boldsymbol{z}}(j)\| = 0. \tag{6.13}$$

By Lemma 5, we have

$$\lim_{i\to\infty} \hat{\boldsymbol{c}}(i, \hat{\boldsymbol{z}}(j)) = \boldsymbol{c}(\hat{\boldsymbol{z}}(j)). \tag{6.14}$$

Thus,

$$\begin{aligned}
&\lim_{i\to\infty} \|\boldsymbol{c}(\boldsymbol{z}^*) - \hat{\boldsymbol{c}}(i, \hat{\boldsymbol{z}}(j))\| \\
=&\|\lim_{i\to\infty} \boldsymbol{c}(\boldsymbol{z}^*) - \lim_{i\to\infty} \hat{\boldsymbol{c}}(i, \hat{\boldsymbol{z}}(j))\| \\
=&\|\boldsymbol{c}(\boldsymbol{z}^*) - \boldsymbol{c}(\hat{\boldsymbol{z}}(j))\|.
\end{aligned}$$

Since $\boldsymbol{c}(\boldsymbol{z})$ and $\hat{\boldsymbol{c}}(i, \boldsymbol{z})$ are Lipschitz continuous;

$$\lim_{i\to\infty} \|\boldsymbol{c}(\boldsymbol{z}^*) - \hat{\boldsymbol{c}}(i, \hat{\boldsymbol{z}}(j))\| \leq K\|\boldsymbol{z}^* - \hat{\boldsymbol{z}}(j)\|. \tag{6.15}$$

As $j \to \infty$, we have $\hat{\boldsymbol{z}}(j) \to \boldsymbol{z}^*$, thus $\lim_{i\to\infty} \|\boldsymbol{c}(\boldsymbol{z}^*) - \hat{\boldsymbol{c}}(i, \hat{\boldsymbol{z}}(j))\| \to 0$. This implies

$$P(\{\lim_{\substack{i\to\infty \\ j\to\infty}} \boldsymbol{c}_m^{\mathrm{SUB}}(i, \hat{\boldsymbol{z}}(j)) = \boldsymbol{c}_m^*\}) = 1. \tag{6.16}$$

□

Now, we can prove that the solutions from MORRF* almost surely converge to a subset of the Pareto optimal set.

Theorem 4. *Given Assumptions 1-4 , the solution generated by the Tchebycheff approach of MORRF* converges to a subset of the Pareto optimal set almost surely, i.e.*

$$P(\lim_{i\to\infty} \Sigma_i^{MORRF^*} \subset \Sigma^*) = 1.$$

6.4.3 Optimality of the boundary-intersection approach

Equation (6.3) in the boundary-intersection approach "aims at getting boundary points rather than Pareto-optimal points" [107]. Das and Dennis [33] claim that the solutions of the subproblems in boundary-intersection approach may not be Pareto-optimal. Thus,

the boundary-intersection approach of MORRF* can also not guarantee that the solutions converge to Pareto-optimal.

6.5 Adaptive Weight Adjustment

In measuring the quality of solutions that are returned by a multi-objective path-planner, we are interested in two criteria:

- the Pareto-optimality of each solution; and
- the diversity of solutions along the Pareto front.

Diversity, in this context, means that the samples from the Pareto optimal set of solutions are approximately uniform across the set. By Theorem 4, we have that all the solutions obtained by MORRF* asymptotically converge to Pareto optimal (at least for the weighted sum and Tchebycheff approaches). In order to maximize the diversity, we uniformly sample the weights for subproblems from a normalized simplex, $w_1 + \cdots + w_K = 1$. However, the mapping from a weight space to an objective space is not always linear. The nonlinear mapping potentially decreases the diversity in the objective space. A natural approach to this issue is adaptively adjusting weights by the information from the objective space [94]. Inspired by the idea, we import weight adjustment to the subproblem trees in MORRF*.

We use the definition of *sparsity level*, SL, from [94] to measure the "diversity by crowdedness" of solutions. The sparsity level of the i-th path $\sigma_i^* \in \Sigma^{\text{MORRF}^*}$ is calculated as

$$d(\sigma_i^*) = \textsc{Sum}_{j=1}^{m} L_2^{NN_i^j}, \tag{6.17}$$

in which $L_2^{NN_i^j}$ means the Euclidean distance from the i-th path to its j-th nearest neighbor in the objective space. A larger sparsity level for a solution means that there aren't other solutions "crowding" around it. The diversity of a set of solutions, called the *spacing*

metric [102, 132], is thus calculated by averaging the sparsity levels of solutions, as follows:

$$S(\Sigma^{\text{MORRF}^*}) = \sqrt{\frac{\text{Sum}_{\sigma_i^* \in \Sigma \text{MORRF}^*}(d(\sigma_i^*) - \bar{d})}{|\Sigma^{\text{MORRF}^*}| - 1}}, \tag{6.18}$$

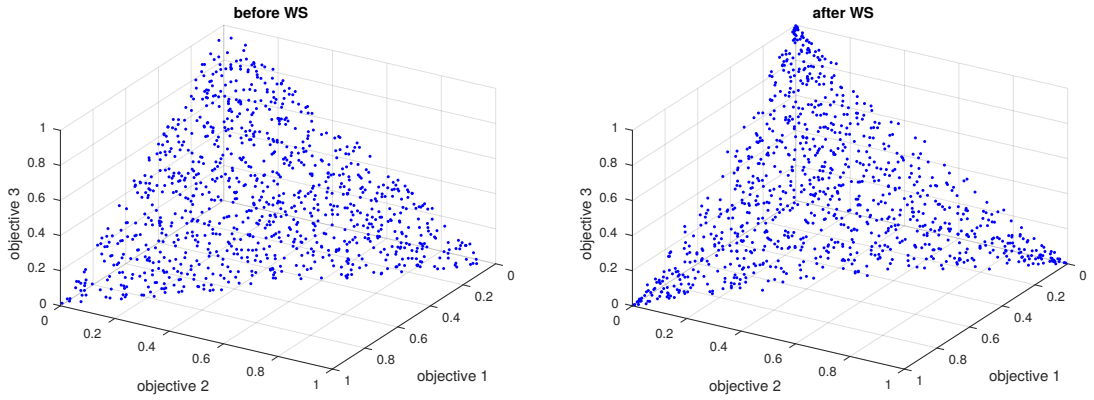
in which $\bar{d} = \frac{\text{Sum}_{\sigma_i^* \in \Sigma \text{MORRF}^* d(\sigma_i^*) - \bar{d})}}{|\Sigma \text{MORRF}^*|}$. Observe that we used functional notation for Sum since we are using Σ's to represent sets of paths. If all the solutions are evenly distributed in the objective space, the value of spacing metric is close to zero.

Our goal is to minimize the value of spacing metric or, equivalently, to maximize the diversity of paths in the objective space, by adaptively adjusting weights. We will need to use the information in the objective space to find new weights. For the Tchebycheff approach, a so-called *WS-transformation* is introduced in [94] to represent the mapping from a weight space to an objective space. This transformation is defined as

$$\lambda' = WS(\lambda) = \left[\frac{\frac{1}{\lambda_1}}{\sum_{k=1}^{K} \frac{1}{\lambda_k}}, \frac{\frac{1}{\lambda_2}}{\sum_{k=1}^{K} \frac{1}{\lambda_k}}, \cdots, \frac{\frac{1}{\lambda_K}}{\sum_{k=1}^{K} \frac{1}{\lambda_k}} \right]. \tag{6.19}$$

Figure 6.5 shows one example of WS transformation. Figure 6.5a shows 3D weights that are uniformly sampled in a simplex. Figure 6.5b shows transformed results. We can see that WS-transformation moves points toward end points of the simplex. The WS-transformation is its own inverse, that is $\lambda = WS(WS(\lambda))$ [94]. In order to have solutions uniformly distributed in objective spaces, we can apply the WS-transformation to uniformly sampled weights.

In most cases, the subproblem trees converge to a sub-optimal structure very quickly. It takes more iterations to refine the structure to the optimal. Because a tree structure determines only one best path, we can use the sparsity level of the best path to represent the sparsity level of the tree. If we could identify that the assigned weight leads to a Pareto-optimal solution with low sparsity level, we could stop the exploration earlier, and start a new subproblem with a new weight that is likely to lead to a solution with higher sparsity level. This forms the basis of the *adaptive weight adjustment* (MORRF*-AWA) algorithm. This



(a) Before (b) After

Figure 6.5: WS Transformation of 3D weights.

algorithm uses reference trees in the samw way as MORRF*, weights of subproblem trees are continously adjusted by the sparsity levels. MORRF*-AWA is presented in Algorithm 4.

Algorithm 4 MORRF*-AWA

for each $V_r \in \boldsymbol{V}_r$ **do**
  $V_r \leftarrow \{x_{init}\}$; $E_r \leftarrow \emptyset$; $i \leftarrow 0$
for each $V_s \in \boldsymbol{V}_s$ **do**
  $V_s \leftarrow \{x_{init}\}$; $E_s \leftarrow \emptyset$; $i \leftarrow 0$
while $i < N$ **do**
  $x_{rand} \leftarrow$ SAMPLE (i) ; $i \leftarrow i + 1$
  $\boldsymbol{G}$ is arbitrary graph from $\boldsymbol{G}_r \cup \boldsymbol{G}_s$.
  $x_{nearest} \leftarrow$ NEAREST(G, x_{rand})
  $x_{new} \leftarrow$ STEER$(x_{nearest}, x_{rand}, \eta)$
  if OBSTACLEFREE$(x_{nearest}, x_{new})$ **then**
    for each $G_r \in \boldsymbol{G}_r$ **do**
      $G_r \leftarrow$ EXTEND$_{Ref}$ $(G_r, x_{new}, x_{nearest}, r)$
    for each $G_s \in \boldsymbol{G}_s$ **do**
      $G_s \leftarrow$ EXTEND$_{Sub}$ $(G_s, x_{new}, x_{nearest}, s)$
  UPDATEBESTPATHS $(\boldsymbol{G}_s)$
  UPDATESPARSITYLEVEL $(\boldsymbol{G}_s)$
  $\boldsymbol{G}_s \leftarrow$ ADAPTIVEWEIGHTADJUSTMENT $(\boldsymbol{G}_s)$

UPDATEBESTPATHS $(\boldsymbol{G})$ updates the best path that is found by the current tree structure. UPDATESPARSITYLEVEL $(\boldsymbol{G}_s)$ updates the sparsity level of each subproblem tree. ADAPTIVEWEIGHTADJUSTMENT($\boldsymbol{G}_s$) is given in Algorithm 5.

Algorithm 5 AdaptiveWeightAdjustment ($\boldsymbol{G}_s$)

1: $H \leftarrow$ UpdateHistoryPair($\boldsymbol{G}_s$)
2: DeleteTreesBySparistiyLevel($\boldsymbol{G}_s, \tau$)
3: $P \leftarrow$ FindHistoryPairsBySparsityLevel(H, τ)
4: $W \leftarrow$ CreateNewWeights(P)
5: $\boldsymbol{G}_s \leftarrow$ CreateNewSubproblemTree(()W)

- UpdateHistoryPair($\boldsymbol{G}_s$): Update the history pairs by the new best paths of trees in $\boldsymbol{G}_s$.
- DeleteTreesBySparistiyLevel($\boldsymbol{G}_s, \tau$) : Delete τ subproblem trees in $\boldsymbol{G}_s$ that have the lowest sparsity levels.
- FindHistoryPairsBySparsityLevel(H, τ): Return τ history pairs that have the highest sparsity levels.
- CreateNewWeights(P): Return $|P|$ number of weights by history pairs P. Given a history pair $(\boldsymbol{c}, SP) \in P$, we can create a new weight λ by Equation (6.20) [94].

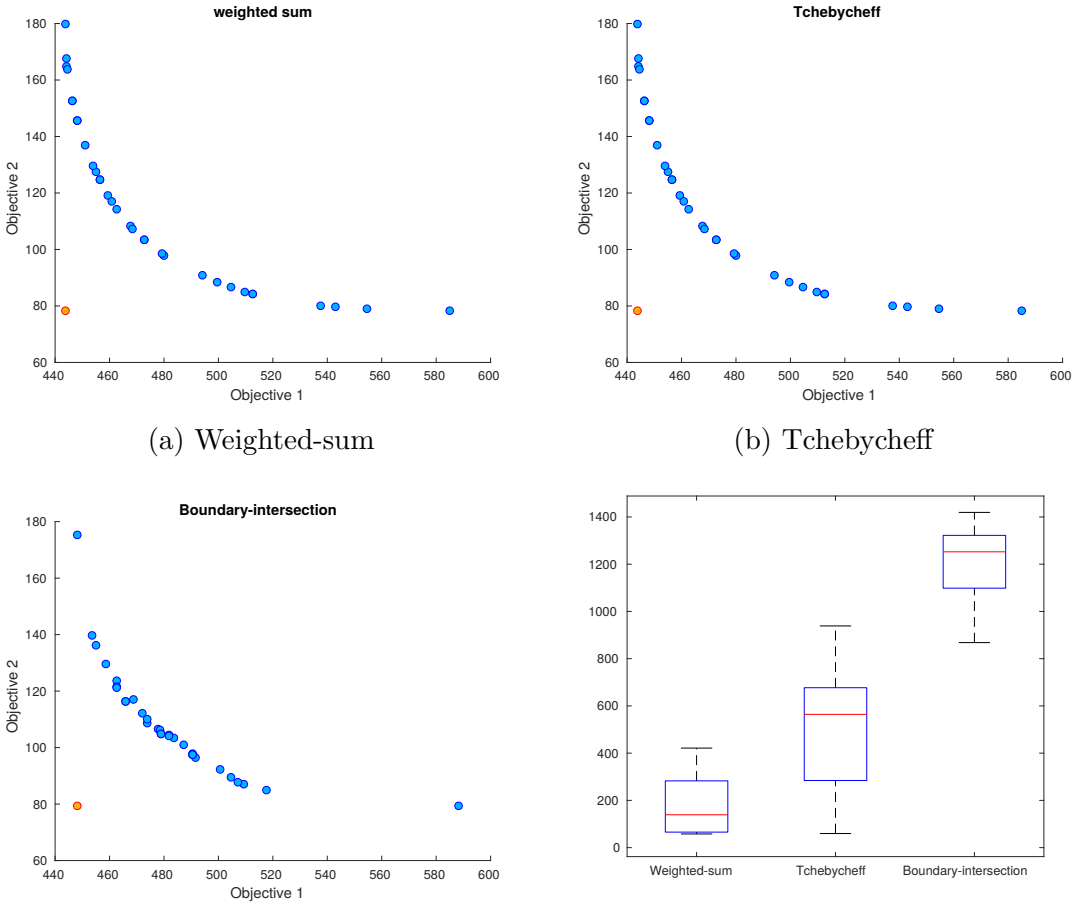
$$\lambda = \left[\frac{\frac{1}{\boldsymbol{c}_K - \boldsymbol{z}_K^{\text{utop}}}}{\sum_{k=1}^{K} \frac{1}{\boldsymbol{c}_k - \boldsymbol{z}_k^{\text{utop}}}}, \cdots, \frac{\frac{1}{\boldsymbol{c}_K - \boldsymbol{z}_K^{\text{utop}}}}{\sum_{k=1}^{K} \frac{1}{\boldsymbol{c}_k - \boldsymbol{z}_k^{\text{utop}}}} \right] \tag{6.20}$$

- CreateNewSubproblemTree(W): Return $|W|$ number of subproblem trees by a set of weights W.

The adaptive weight adjustment process will refine the weights of subproblem trees by removing trees with low sparsity levels and adding new trees that are likely to have high sparsity levels.

6.6 Experiments

In this section, we use four cases to evaluate the performance of the MORRF* and MORRF*-AWA. Because MORRF*-AWA only includes adaptive weight adjustment for the Tchebycheff



(a) Weighted-sum

(b) Tchebycheff

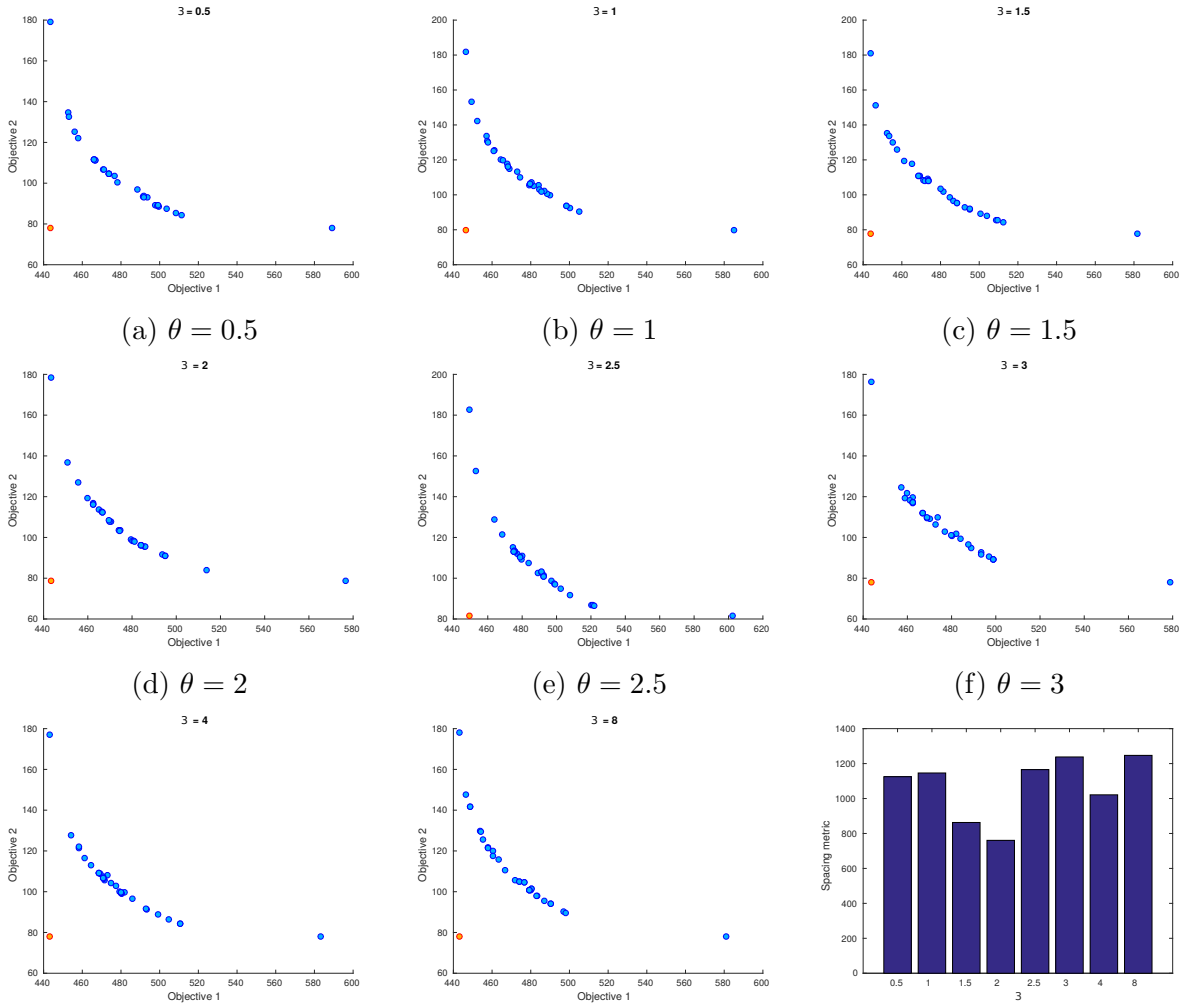
(c) Boundary-intersection, $\theta = 4$

(d) Spacing metric

Figure 6.6: Pareto front of a two-objective path-planning problem.

approach, we call it "Tchebycheff-AWA" in the comparison of the Tchebycheff approach in MORRF*.

Case 1 is a two-objective planning problem in a workspace without obstacles. Figure 6.6a-6.6c show the solutions found by the weighted-sum, Tchebycheff and Boundary-intersection approaches, respectively, usingMORRF* in the objective space. 30 solutions are collected by each approach and the algorithms are run for 8000 iterations. Figure 6.6d shows the spacing metric of the solutions found by all three approaches. A lower value means a better diversity. We can see that the solutions found by all the three approaches converge to Pareto-optimal by the shape of "Pareto-front". We can also see that the weighted-sum approach superior diversity, while the boundary-intersection approach has inferior diversity.



(a) $\theta = 0.5$ (b) $\theta = 1$ (c) $\theta = 1.5$

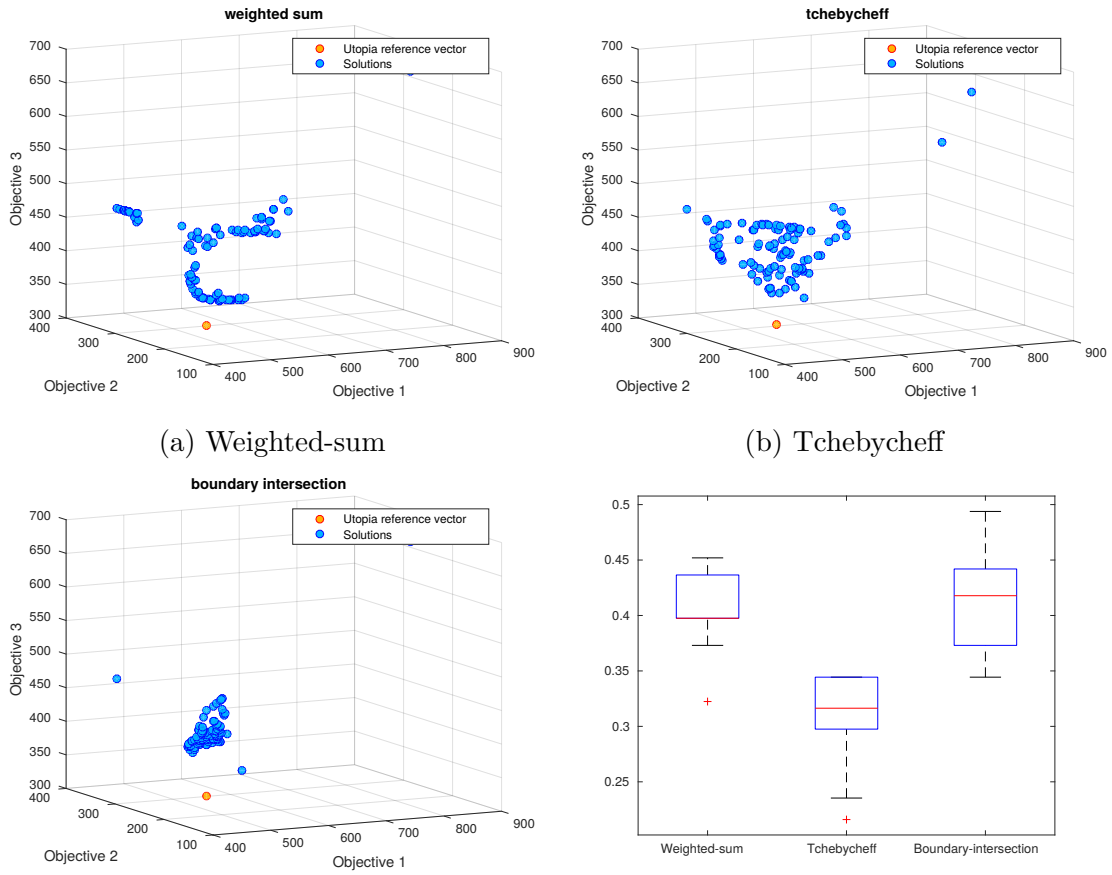
(d) $\theta = 2$ (e) $\theta = 2.5$ (f) $\theta = 3$

(g) $\theta = 4$ (h) $\theta = 8$ (i) Spacing metric

Figure 6.7: Different θ of the boundary-intersection approach in a two-objective optimization problem.

Our implementation of boundary-intersection approach uses the penalty-based boundary intersection in Equation (6.4). There is a parameter θ in balancing the penalties. Zhang and Li [130] claim that tuning θ can impact the diversity of solutions. Figure 6.7b shows the solutions in the objective space found by choosing different θ. The spacing metrics are compared in Figure 6.7i. We can see that $\theta = 2$ shows the best diversity.

Case 2 tests a three-objective planning problem in a workspace without obstacles. Figure 6.8a - 6.8a show the solutions found by three approaches in the objective space.



(a) Weighted-sum

(b) Tchebycheff

(c) Boundary-intersection

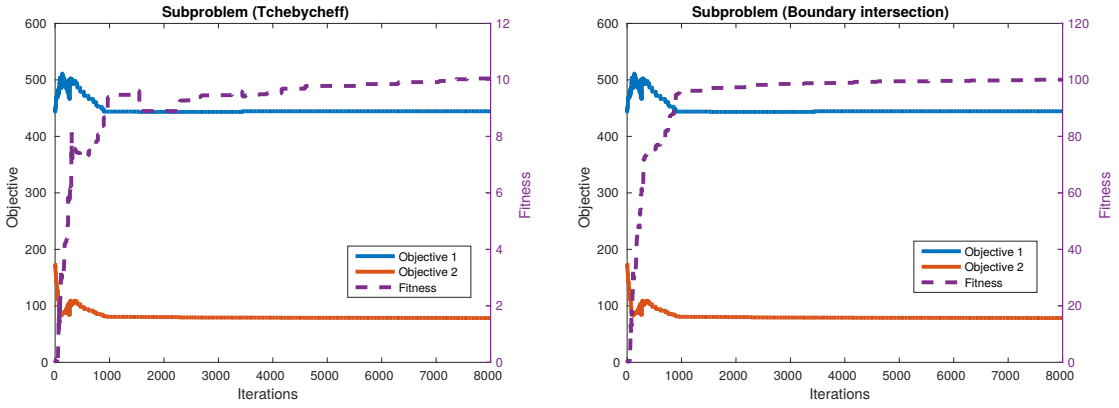
(d) Spacing metric

Figure 6.8: Pareto front of a three-objective path-planning problem.

Figure 6.8d shows the comparison by spacing metric. We can see that the Tchebycheff approach shows the best diversity among all three approaches.

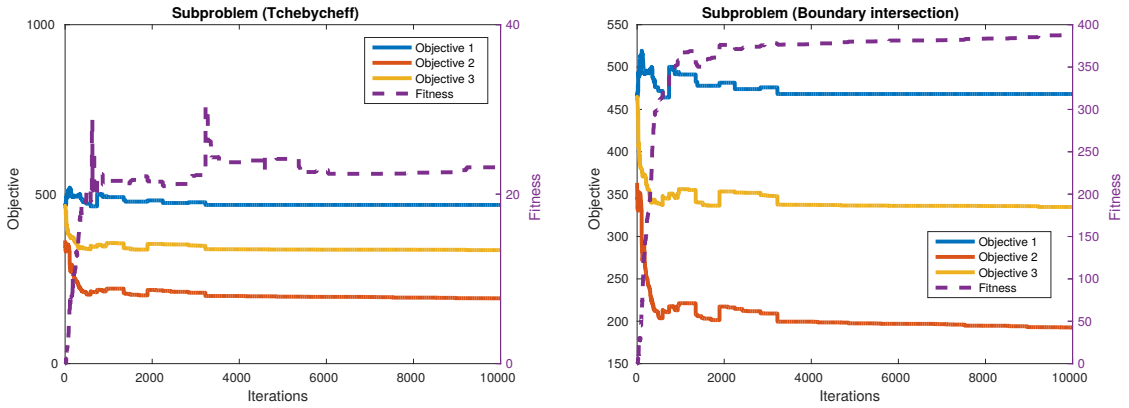
Figure 6.9 and Figure 6.10 show the convergence of a subproblem tree in a two-objective path-planning problem and in a three-objective path-planning problem. The fitness means the objective value of each subproblem. Here the convergence of a tree is represented by the convergence of the best path it finds. As expected, the subproblem trees converge after reference trees converge.

Case 3 tests a two-objective planning problem in a world with obstacles. Figure 6.11a - 6.11c show the solutions found by three approaches in the objective space. Figure 6.11d compares the spacing metrics of solutions found by three approaches. The Tchebycheff approach shows the best diversity.



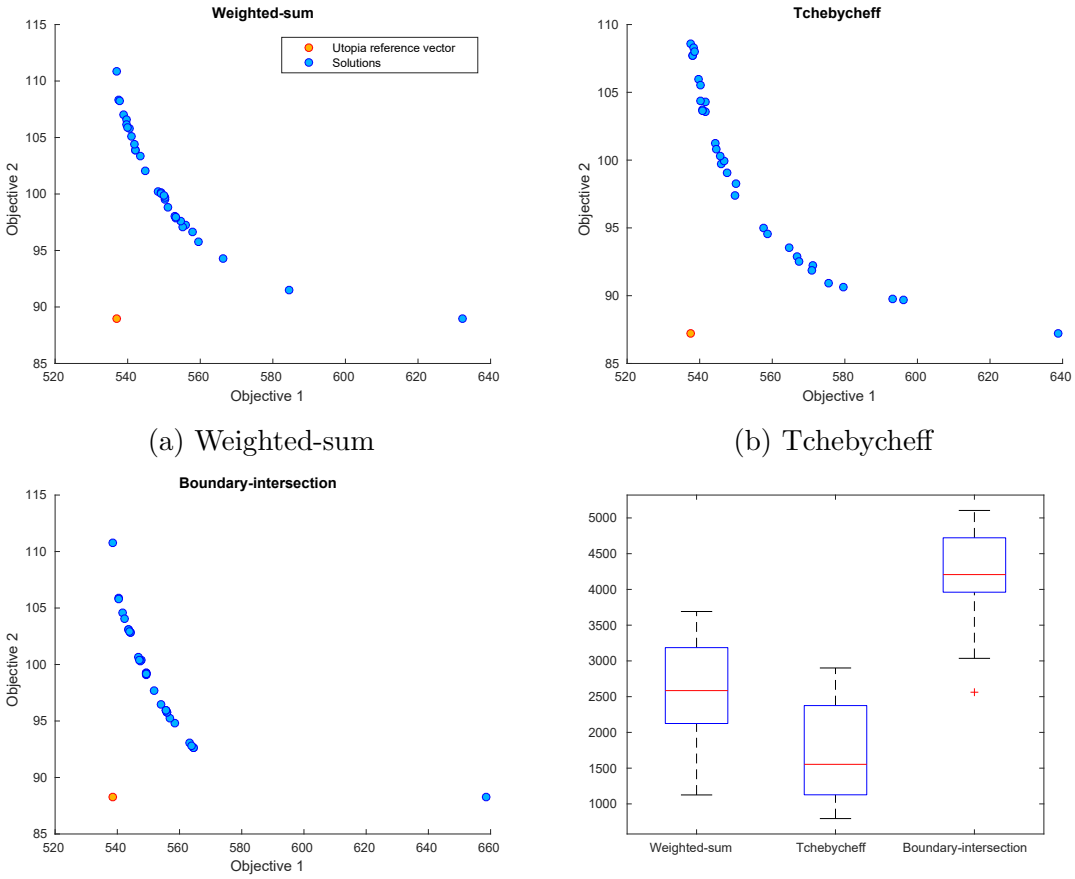
(a) Tchebycheff (b) Boundary-intersection

Figure 6.9: Convergence of a subproblem tree in a two-objective path-planning problem.



(a) Tchebycheff (b) Boundary-intersection

Figure 6.10: Convergence of a subproblem tree in a three-objective path-planning problem.



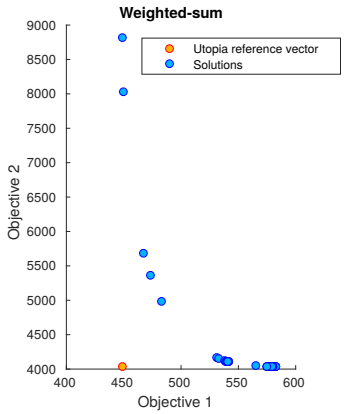
(a) Weighted-sum

(b) Tchebycheff

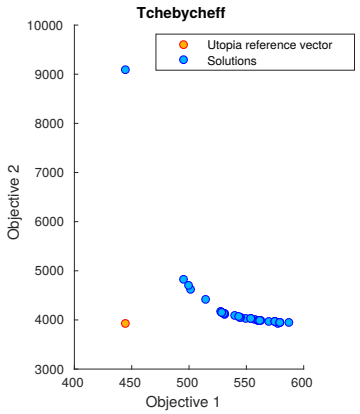
(c) Boundary-intersection

(d) Sparsity level

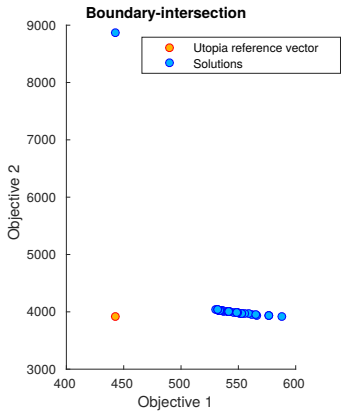
Figure 6.11: Pareto front of a two-objective path-planning problem in a workspace with obstacles.



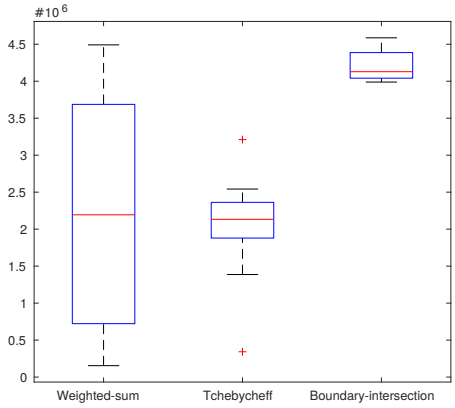
(a) Weighted-sum



(b) Tchebycheff



(c) Boundary-intersection

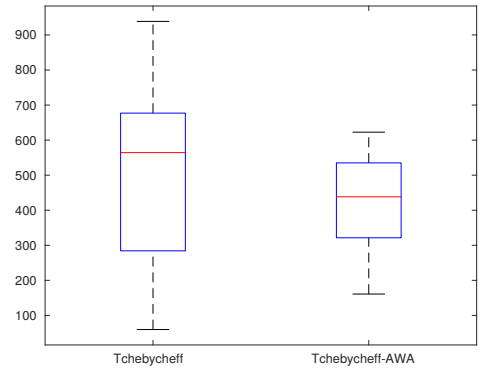


(d) Sparsity level

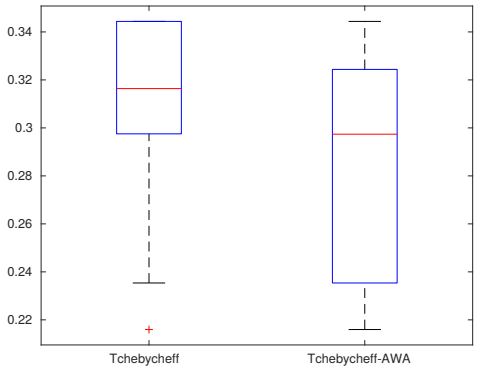
Figure 6.12: Pareto front of a two-objective path-planning problem.

In all above cases, the range of the objectives over the set of solutions is small (meaning that there is little variance in the scores of different solutions). In case 4, we tested a two-objective path-planing problem in a workspace without obstacles but the values in the objective 2 vary in a wider range than those in the objective 1. Figure 6.12d shows the solutions from the weighted-sum approach and the Tchebycheff approach. The Tchebycheff approach shows more consistent performance in 20 different runs.

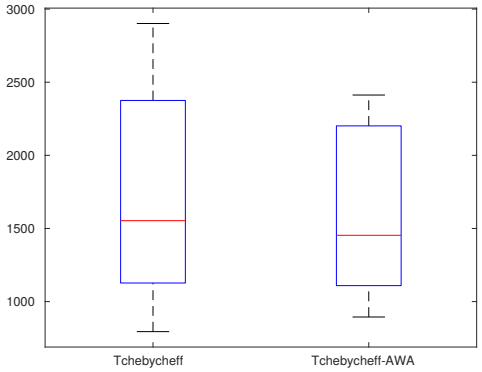
We now evaluate whether MORRF*-AWA improves the diversity of solution by comparing with the Tchebycheff approach in MORRF*. Figure 6.13 shows that it improves the diversities in all above four cases.



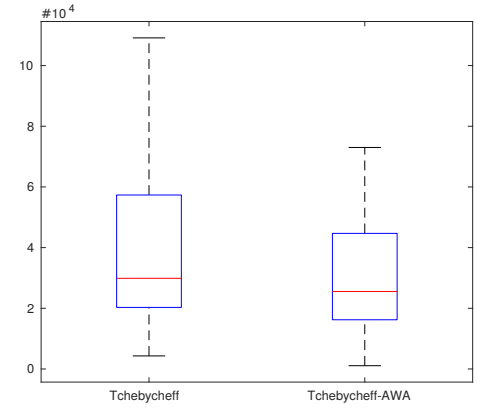
(a) case 1



(b) case 2



(c) case 3



(d) case 4

Figure 6.13: Spacing metric comparison between Tchebycheff approach and Tchebycheff-AWA.

Because the cost of a path is an integral along the points in the path and we assume additivity of the objective functions, it is very hard to create a test case that the mulit-objective function is non-convex. By the cases we tested, we can see all three approaches can provide Pareto-optimal solutions if there are enough iterations to run. The Tchebycheff approach shows the best performance in diversity in different cases. The diversity of solutions can be improved by introducing adaptive weight adjustment a la MORRF-AWA*.

6.7 Summary and Future Work

In this paper, we proposed a sampling-based multi-objective path-planning algorithm, MORRF*. It consists of reference trees and subproblem trees. The reference trees explore a workspace to provide the estimation of Utopia reference vectors. The subproblem trees use the estimated Utopia reference vectors to connect sampled points for solving individual subproblems. Three different approaches of creating subproblems are provided. The solutions returned by subproblem trees are the solutions of the algorithm. We have applied theoretical analyses on the optimality of solutions in different approaches. MORRF*-AWA is then introduced, and applied adaptive weight adjustment to subproblem trees for enhancing diversity of solutions. We have conducted experiments to show the performances of different approaches in MORRF* and MORRF*-AWA.

The weighted sum and Tchebycheff methods for giving weights for the subproblem trees are guaranteed to produce Pareto optimal solutions, but the boundary intersection method is not. In the empirical studies, the Tchebycheff method produced the greatest diversity, even when the θ parameter of the boundary intersection technique was optimized. Adjusting weights improved diversity for the Tchebycheff method, but did so at the cost of time to find solutions.

Future work should mix subproblems that different approaches in one MORRF* algorithm by, for example, including weighted sum, Tchebycheff, adjusted-weight Tchebycheff,

and boundary intersection methods. This can potentially use the advantages of the different approaches to maximize performance of the complete algorithm.

Future work should also seek to exploit similarities among subproblems to accelerate the tree extending process. Similarity among subproblems can potentially allow the algorithm to reuse existing tree structures in creating new subproblem trees in the process of adaptive weight adjustment.

Chapter 7

Homotopy-Aware RRT* : Toward Human-Robot Topological Path-Planning [1]

Abstract

An important problem in human-robot interaction is for a human to be able to tell the robot go to a particular location with instructions on how to get there or what to avoid on the way. This paper provides a solution to problems where the human wants the robot not only to optimize some objective but also to honor "soft" or "hard" topological constraints, i.e. "go quickly from A to B while avoiding C". The paper presents the HARRT* (homotopy-aware RRT*) algorithm, which is a computationally scalable algorithm that a robot can use to plan optimal paths subject to the information provided by the human. The paper provides a theoretic justification for the key property of the algorithm, proposes a heuristic for RRT*, and uses a set of simulation case studies of the resulting algorithm to make a case for why these properties are compatible with the requirements of human-robot interactive path-planning.

[1] Published in The Eleventh ACM/IEEE International Conference on Human Robot Interaction(HRI '16). Authors are Daqing Yi, Michael A. Goodrich and Kevin D. Seppi.

7.1 Introduction

In search and rescue, police, and military applications of human-robot teaming, a human may want to tell a robot to go to a particular location while giving information that the robot should use to decide what path to take. This is a form of interaction that requires a robot to plan a path that honors the human's intent. In this paper, we focus on two useful elements of intent: the shape and quality of the planned path. Although algorithms exist to implement path properties like continuity and smoothness, there appears to be no algorithm that is computationally tractable while being powerful enough to honor both shape and quality aspects of a path. Such an algorithm would support human instructions like "go quickly around building A and then between the two trees while avoiding region C." In this paper, we present an algorithm that optimizes path quality while allowing a human to specify regions to avoid, preferences for directions of traveling around obstacles, via-point/waypoint constraints, and reference path constraints [124].

The contribution of this paper (a) is a computationally efficient algorithm for detecting when two paths are homotopic that (b) can be used as a heuristic for an RRT* planner to restrict search to a given homotopy class, where (c) the planning is done by the robot and (d) the optimization criteria and shape constraint is specified by the human. The specific contribution to human-robot interaction is that these properties, supported by simulation results, create a set of path affordances that allow a human to specify a wide range of shape constraints and objective preferences that the robot honors in path-planning.

The key to this algorithm is the topological concept of *homotopy*, which is a mathematical formalism of the inherent similarity or dissimilarity of two paths. Given two paths σ_1 and σ_2 with the same endpoints, if one can be continuously deformed into the other without encroaching any obstacle and without moving the endpoints then they are said to be *homotopic* [12, 46]. We write this as $\sigma_1 \simeq \sigma_2$. In a slight abuse of notation, we say that two sets of paths are homotopic $\Gamma_1 \simeq \Gamma_2$ if all paths in the sets are homotopic.

We restrict attention to paths that start at a given initial position x_{init} and end at a given terminal position x_{goal}, and group the set of all such possible paths into classes according to their shape properties. Formally, the set of paths that are homotopic to each other form a *homotopy class*, and the set of homotopy classes partition the set of all possible paths between any two positions x_{init} and x_{goal}. In an environment containing obstacles, we argue that this homotopy partition, provides a mapping between a human-based or colloquial use of the term "shape" and the corresponding topological notion.

Theoretically, there exist infinite homotopy classes, because an obstacle can be encircled an arbitrary number of times. With some loss of generality, we impose the following:

Restriction 1. *We consider only "simple paths", that is, paths that do not form complete loops around obstacles.*

We can now define the problem the algorithm must solve.

Definition 1. ***Homotopy-Based Optimal Path-Plan-ning*** *Let $X \subset \mathbb{R}^d$ denote a bounded connected open set, $X_{obs} \subset X$ an obstacle space, $X_{free} = X \setminus X_{obs}$ the obstacle-free space, x_{init} an initial state , and x_{goal} a goal state. Define a* path *in X as a continuous curve parameterized by s as $\sigma(s) : [0,1] \to X$. Denote the monotonic increasing cost of the path as $\textsc{Cost}(\sigma)$. Let $\boldsymbol{H}(x_{init}, x_{goal})$ denote the set of homotopy classes defined by $x_{init} \in X_{free}$ and $x_{goal} \in X_{free}$, $H = h_1, \cdots, h_N \subseteq \boldsymbol{H}$ a particular subset of homotopy classes, and $h(\sigma)$ the homotopy class of σ. The goal is to find paths $\sigma^*_{h_i} \in \Sigma^*, h_i \in H$ such that (a) $\forall s \in [0,1], \sigma^*(s)_{h_i} \in X_{free}$; (b) $\sigma^*_{h_i}(0) = x_{init}$ and $\sigma^*_{h_i}(1) = x_{goal}$; and (c) $\forall h_i \in H, \sigma^*_{h_i} = \arg\min_{\sigma \in X_{free} \wedge h(\sigma) = h_i} \textsc{Cost}(\sigma)$.*

Definition 1 says that, given a particular set of homotopy classes and a cost function, the planner should find paths that minimize the cost in the given homotopy classes.

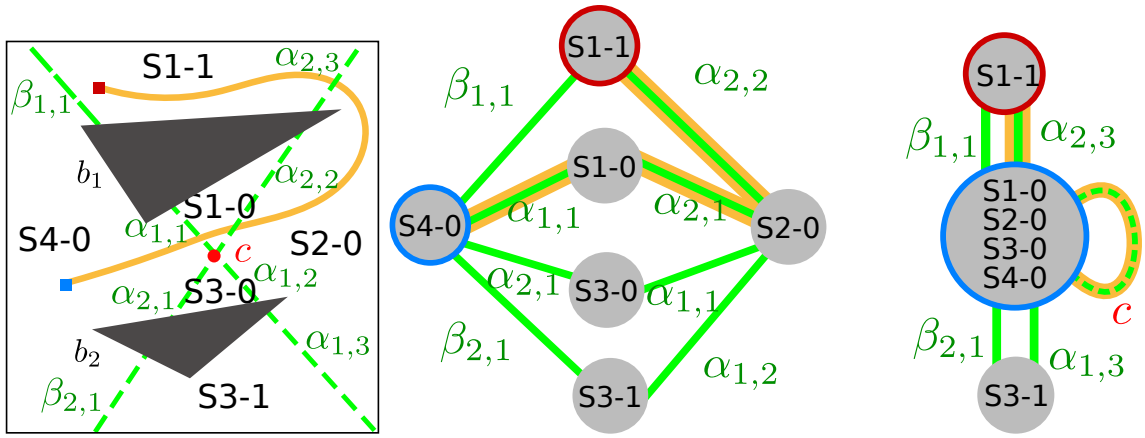
7.2 Related Work

From the human side of the HRI problem, many researchers have noted that humans often represent the world using topological rather than metric-based mental models [66]. Various methods have been used to create topological representations that can be used by path-planners for robots [40, 74, 104, 114]. Because these planners represent the relationships between landmarks as a graph and then plan paths using a graph-search algorithm, they satisfy strict topological constraints but do not minimize $\textsc{Cost}(\sigma)$.

Homotopy-based path-planning should enable a combination of constraints and continuous objectives, but determining the homotopic equivalence of two paths is usually computationally expensive or not general. For example, the Voronoi diagram is used to identify a path from any homotopy class in [8], but this algorithm has limitations when there are certain kinds of complex obstacles in the world. A so-called funnel algorithm in the universal covering space yields an improvement [49], but complexity does not scale well when obstacles are not smooth and convex.

Other algorithms for finding homotopic equivalence include (a) using semi-algebraic cuts to convert a candidate path into a "word" [44] and (b) converting a plane into a complex plane and then finding invariant properties of the paths in this plane [12]. Unfortunately, their performance depends on how the map is discretized; computation cost expands greatly if the obstacles are reasonably approximated by a high resolution discretization.

Homotopies have been used in sampling-based algorithms. In a probabilistic road map structure, the paths can be categorized into homotopy classes using a method called homotopic redundancy [101]. Another approach is to divide the space using a set of reference frames crossing each obstacle [48]. This method is particularly relevant because, as we shall show, how a path crosses the reference frames can be represented as a canonical sequence and comparing the sequences allows homotopic equivalence to be determined. We extend the ideas in these algorithms to enable optimal path-planning within a more complete set of



(a) Ref. frames (b) State trans. (ST) (c) Homotopic ST

Figure 7.1: Map with obstacles.

homotopically equivalent paths. To accomplish this, we use a variation of the bidirectional RRT* algorithm [110], which is more efficient than the original [58].

7.3 Homotopic String Classes

This section shows how to create a string-based representation of any simple path, and then use the easy-to-compute strings to efficiently identify homotopic equivalence.

7.3.1 Generating String Representations

Strings are generated using an improved method of Jenkins' approach [53] to detecting homotopic equivalence of two paths by separating a map into disjoint subregions [47]. A reference frame segment, which Jenkins called a *reference frame*, is a line segment constructed from a center point and a point in a obstacle, extended to the map boundaries. The collection of reference frames created from a set of obstacles partition the map into disjoint subregions. Figure 7.1a shows an example of a map with two obstacles and two reference frames (blue and green dashed lines) – one for each obstacle. Strings will be constructed based on the simple idea that if two paths cross the same sequences of reference frames, then they belong to the same homotopy class.

In Algorithm 1, the reference frames $\boldsymbol{R}$ are created from a set of points that are generated as follows: In a map with a set of obstacle regions $\boldsymbol{B}$, an obstacle point b_k is randomly sampled from each obstacle region $B_k \in \boldsymbol{B}$. A center point c is then randomly sampled in the non-obstacle region $\boldsymbol{X}_{free} = \boldsymbol{X} \setminus \boldsymbol{B}$ subject to the constraint that it is not in any line that connects two different b_k. Connecting each b_k with c creates a radial structure of reference frames that partition the map.

Algorithm 1 InitRefFrames ($\boldsymbol{X}_{free}, \boldsymbol{B}$)

$\boldsymbol{R} = \emptyset, \boldsymbol{b} = \emptyset$
for each $B_k \in \boldsymbol{B}$ **do**
    $\boldsymbol{b} \leftarrow \boldsymbol{b} \cup b_k$ randomly sampled from B_k
$c \leftarrow$ Randomly sampled from $\boldsymbol{X}_{free}$
while $\exists b_k, b_{k'}, c \in$ Line($b_k, b_{k'}$) **do**
    $c \leftarrow$ Randomly sampled from $\boldsymbol{X}_{free}$
for each $b_k \in \boldsymbol{b}$ **do**
    $l_k \leftarrow$ Line(b_k, c)
    $\{l_{k_m}\} \leftarrow$ Intersect$(l_k, \boldsymbol{B}, c)$
    $\boldsymbol{R} \leftarrow \boldsymbol{R} \cup \{l_{k_m}\}$
return $\boldsymbol{R}$

The method Line(p_1, p_2) returns the line defined by p_1 and p_2, and the method Intersect$(r, \boldsymbol{B}, c)$ returns all segments of line r that don't intersect with an obstacle in $\boldsymbol{B}$ or the center point c.

If we assign an ID character to each reference frame, then how a path sequentially crosses the reference frames can be converted into a string of ID characters. For example, in Figure 7.1a, the path that starts in subregion S_{4-0} and ends in subregion S_{1-1} sequentially visits the reference frames $\alpha_{1,1}, \alpha_{2,2}, \alpha_{2,3}$. Concatenating these characters yields the path string $\alpha_{1,1}\alpha_{2,2}\alpha_{2,3}$. A deterministic finite automata (DFA) formalizes this process; see Figure 7.1b.

Definition 2. *Let $\boldsymbol{M} = (\boldsymbol{S}, \boldsymbol{R}, \delta, S_0, S_T)$ be a DFA that represents the string generation process from a path, where $\boldsymbol{S}$ is a set of subregions, $\boldsymbol{R}$ is a set of reference frames, $S_0 \in \boldsymbol{S}$ is the start subregion, $S_T \in \boldsymbol{S}$ is the end subregion, and $\delta : \boldsymbol{S} \times \boldsymbol{R} \rightarrow \boldsymbol{S}$ is the transition function that defines how one subregion transitions to another subregion by crossing one reference frame in $\boldsymbol{R}$. A string v is created as follows:*

- *v is initialized as an empty string ε.*
- *The path starts at $x_{init} \in S_0$ and ends at $x_{goal} \in S_T$.*
- *When there is a transition across a reference frame $r \in \boldsymbol{R}$, $v \leftarrow vr$.*

Thus, v is the string generated by a path through the map using $\boldsymbol{M}$. A *string block* Γ_v is the set of all paths that generate string v. We now develop conditions under which a set of string blocks partition the set of all simple paths into a set of disjoint homotopy classes. We present these as a series of properties, lemmas, and theorems.

Recall that we are restricting attention to simple paths, and let Γ denote the set of all simple paths. We begin with properties of paths and the strings that they generate through $\boldsymbol{M}$. Property 1 states that there are a finite number of unique strings generated by $\boldsymbol{M}$ that induce a partition over the Γ.

Property 1. $\Gamma = \bigcup_{i=1}^{m} \Gamma_{v_i}$ *and* $v_i \neq v_j \Rightarrow \Gamma_{v_i} \cap \Gamma_{v_j} = \emptyset$.

The second property is that $\simeq$ is an equivalence relation.

Property 2. $\simeq$ *in* $\Gamma_{v_i} \simeq \Gamma_{v_j}$ *is an equivalence relation.*

Property 3 states that two paths are homotopic when they belong to the same string block Γ_v. In other words, if σ_i and σ_j generate the same string v, σ_i and σ_j are homotopic.

Property 3. $\forall \sigma_i, \sigma_j \in \Gamma_v, \sigma_i \simeq \sigma_j$.

Property 4 states that if two touching paths σ_i and σ_j are concatenated together to form $\sigma_i \circ \sigma_j$ then $\boldsymbol{M}$ generates a string $v_i v_j$ that is the concatenation of the strings generated by the two individual paths v_i and v_j.

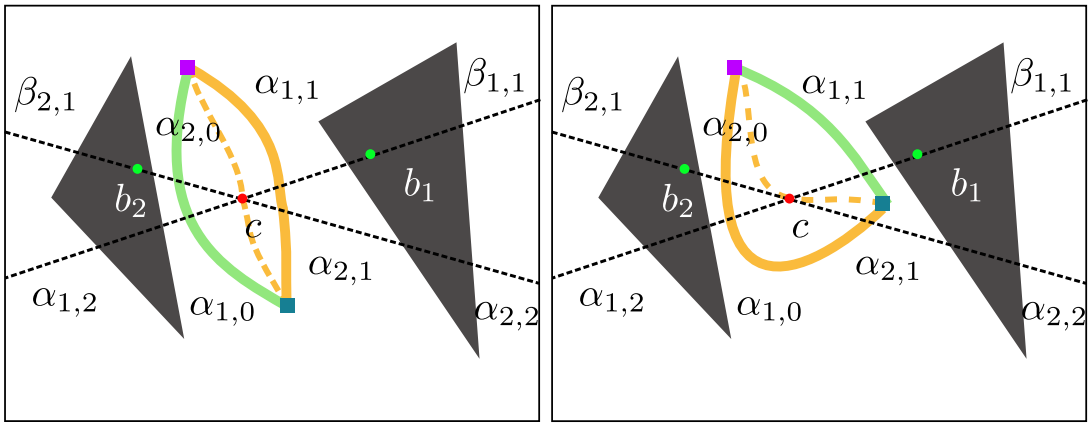
Property 4. *If* $\sigma_i \in \Gamma_{v_i}, \sigma_j \in \Gamma_{v_j}$ *and* $\sigma_i(1) = \sigma_j(0)$ *then* $\sigma_i \circ \sigma_j \in \Gamma_{v_i v_j}$.

To determine the homotopic equivalence of two paths that belong to different string blocks, we remove an ambiguity from $\boldsymbol{M}$. Observe that the reference frames form a radial structure emanating from the center point c. We denote the set of all reference frames segments

between the center point c and an obstacle boundary by $\boldsymbol{R}_c$ and the set of subregions that connect with the center point c by $\boldsymbol{S}_c$.

Property 5. *A path segment that sequentially crosses several reference frames in $\boldsymbol{R}_c$ between two different subregions in $\boldsymbol{S}_c$ is homotopic to a path segment that crosses only the center point c.*

For example, in Figure 7.2a, two paths with different strings $\alpha_{2,0}\alpha_{1,0}$ and $\alpha_{1,1}\alpha_{2,1}$ indicate two paths in the same homotopy class. By Property 5, we have $\Gamma_{\alpha_{2,0}\alpha_{1,0}} \simeq \Gamma_c \simeq \Gamma_{\alpha_{1,1}\alpha_{2,1}}$. Figure 7.2b illustrates a second example.



(a) Example A (b) Example B

Figure 7.2: Equivalence in Homotopy.

An important consequence of Property 5 is that paths that only go through regions $\boldsymbol{S}_c$ are homotopic to a simple path segment that starts and ends at the same position within $\boldsymbol{S}_c$. Furthermore, all of these paths are homotopic to a path that generates the empty string. This means that we can merge all the subregions in $\boldsymbol{S}_c$ into a new subregion $\widehat{S}_c$. We can now create a new DFA, illustrated in Figure 7.1c, that removes the ambiguity associated with the center region.

Definition 3. *Let $\boldsymbol{M}^h = (\boldsymbol{S}^h, \boldsymbol{R}^h, \delta^h, S_0^h, S_T^h)$ be a* homotopic DFA *that represents the string generation process from a path, where $\boldsymbol{S}^h = (\boldsymbol{S} \setminus \boldsymbol{S}_c) \cup \{\widehat{S}_c\}$ is a set of subregions, $\boldsymbol{R}^h = \boldsymbol{R} \setminus \boldsymbol{R}_c$ is a set of reference frames, $S_0^h \in \boldsymbol{S}^h$ is the start subregion, $S_T^h \in \boldsymbol{S}^h$ is the end subregion, and*

$\delta^h : \boldsymbol{S}^h \times \boldsymbol{R}^h \to \boldsymbol{S}^h$ is the transition function that defines how one subregion transitions to another subregion along the path by reference frames in $\boldsymbol{R}^h$. Strings are generated as follows:

- *v is initialized as an empty string ε.*
- *The path starts at $x_{init} \in S_0^h$ and ends at $x_{goal} \in S_T^h$.*
- *When there is a transition across a reference frame $r \in \boldsymbol{R}^h$, $v \leftarrow vr$.*
- *When there is transition $\boldsymbol{R}_c$, $v \leftarrow v\varepsilon = v$.*

Now that we have removed this ambiguity, we observe an important relationship between a simple path and the string that $\boldsymbol{M}^h$ generates from this path. Let $v = \boldsymbol{M}^h(\sigma)$ denote the string generated from a path σ.

Property 6. *A duplicate ID character in a string $\boldsymbol{M}^h(\sigma)$ indicates that σ has visited a subregion at least twice.*

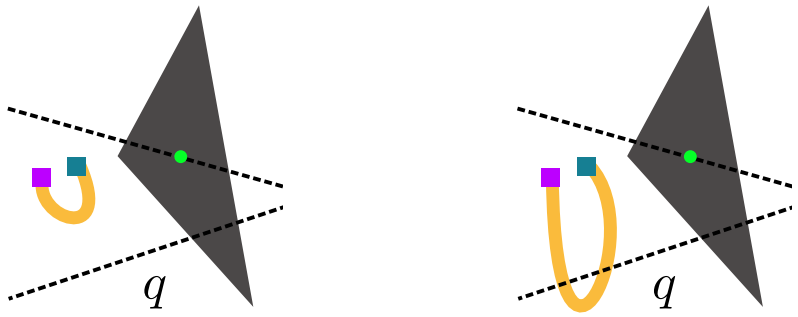
We call strings that don't have duplicate ID characters *non-repeating strings* v^*. Non-repeating strings can only be generated by simple paths that never leave a subregion by crossing a reference frame and then returning by recrossing that same reference frame. This implies the next property.

Property 7. *In every simple homotopy class there exists a path σ such that $\boldsymbol{M}^h(\sigma)$ has no duplicate characters.*

Consider a string constructed in the following manner: begin with the empty string ε and recursively insert a palindromic substring ww^R, where the R operator reverses the characters in the string, into any position of a string. We denote a string made up of recursively embedded palindromic substrings an *REP string*. Note that ε and strings of the form ww^R are REP strings.

We now present the culminating lemma of this subsection.

Lemma 1. *If σ is a simple path segment that begins and ends in the same subregion and encloses no obstacle, then $\boldsymbol{M}^h(\sigma)$ is a REP string.*



(a) Before deformation (b) After deformation

Figure 7.3: Path deformation.

Proof. The proof is by induction on the number of subregions visited by a path.

- **Base case:** If a simple path segment σ never leaves a subregion then $\boldsymbol{M}^h(\sigma) = \varepsilon$.
- **Induction step:** Assume a simple path segment σ that begins and ends in the same subregion, and $\boldsymbol{M}^h(\sigma)$ is a REP string. Deform the path segment σ into a different simple segment σ' by crossing only one more reference frame with ID q, as illustrated in Figure 7.3. $\boldsymbol{M}^h(\sigma')$ is $\boldsymbol{M}^h(\sigma)$ embedded with qq, where $qq = qq^R$ is a palindromic substring. Thus, $\boldsymbol{M}^h(\sigma')$ is also a REP string.
- **Conclusion:** Any simple path segment σ that begins and ends in the same subregion can be obtained by recursively applying deformation to a simple path that never leaves a subregion in the inductive step.

□

A useful consequence of this Lemma is the following.

Corollary 1. *All simple paths that begin and end in the same subregion and enclose no obstacle are homotopic to each other and to a path σ such that $\boldsymbol{M}^h(\sigma) = \varepsilon$.*

7.3.2 Identifying the Equivalence

Having characterized several relationships between a path σ and its corresponding string $\boldsymbol{M}^h(\sigma)$, we now want to identify string properties that tell us when two paths are homotopic.

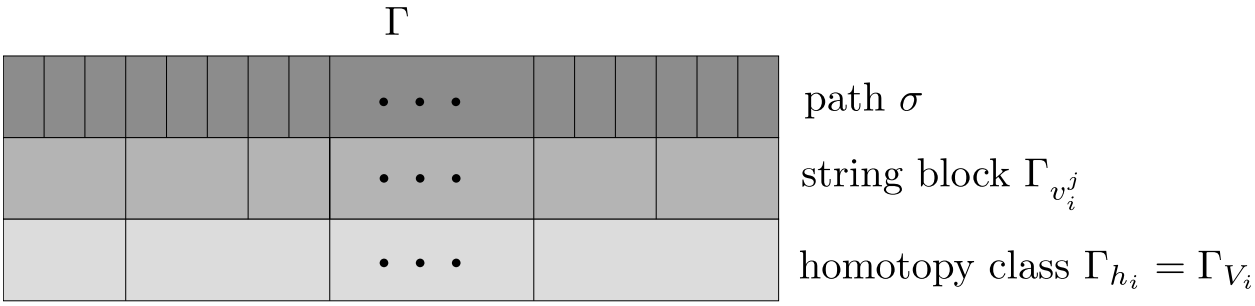


Figure 7.4: A hierarchy of path partitions.

Although Property 3 tells us that two paths are homotopic when they are in the same string block, we need more. Specifically, we also need to know when two paths from different string blocks are homotopic.

Write the set of all the simple paths as the union of several homotopy classes $\Gamma = \Gamma_{h_1} \cup \Gamma_{h_2} \cdots \cup \Gamma_{h_g}$, in which Γ_{h_i} is the set of all the paths in homotopy class h_i. By Property 1 and Property 3, we know that each homotopy class is a union of several string blocks, that is $\Gamma_{h_i} = \bigcup \Gamma_{v_i^j}$. Let V_i denote the set of strings v_i^j associated with the homotopy class h_i, and define $\Gamma_{V_i} = \bigcup \Gamma_{v_i^j}$.

Property 8 tells us that given a homotopy class h_i and its corresponding V_i, if $\boldsymbol{M}^h(\sigma) \in V_i$, then $\sigma \in \Gamma_{h_i}$, and vice versa.

Property 8. $\forall \Gamma_{h_i}, \exists V_i = \cup v_i^j, \Gamma_{h_i} = \Gamma_{V_i} = \cup \Gamma_{v_i^j}$.

This induces a hierarchy of path partitions and their associated string patterns, as illustrated in Figure 7.4.

This hierarchy tells us that we can identify the homotopy class of a path, $\sigma \in \Gamma_{h_i}$, by finding $\boldsymbol{M}^h(\sigma) \in V_i$.

We now show that we can use a non-repeating string v^* to determine whether $\boldsymbol{M}^h(\sigma_1)$ and $\boldsymbol{M}^h(\sigma_2)$ belong to the same V_i. We begin with Lemma 2, which shows that every path is homotopic to a path that generates a non-repeating string.

Lemma 2. $\forall v, \exists v^*, \Gamma_v \simeq \Gamma_{v^*}$

Proof. Every path in a homotopy class is homotopic to the shortest path in that same homotopy class. By Property 7, the shortest path generates a non-repeating string v^*, which means $\Gamma_v \simeq \Gamma_{v^*}$. □

The next lemma states that any two different non-repeating string blocks are not homotopic.

Lemma 3. *If* $v_i^* \neq v_j^*$ *then* $\Gamma_{v_i^*} \not\simeq \Gamma_{v_j^*}$.

Proof. Assume that $\exists v_i^* \neq v_j^*$ such that $\Gamma_{v_i^*} \simeq \Gamma_{v_j^*}$. Let $\sigma_i \in \Gamma_{v_i^*}$ and $\sigma_j \in \Gamma_{v_j^*}$. Because σ_i and σ_j are homotopic to each other, we can assume without loss of generality that σ_j and σ_i end at the same point, $\sigma_i(1) = \sigma_j(1)$. Again because $\sigma_i \simeq \sigma_j$, the continuous path formed when we connect σ_i to $\sigma_j^R(s) = \sigma_j(1-s)$, a reversed version of σ_j, encloses no obstacle. By Lemma 1, $\boldsymbol{M}^h(\sigma_i \circ \sigma_j^R)$ is a REP string. Observe that $\boldsymbol{M}^h(\sigma_i \circ \sigma_j^R) = v_i^* v_j^{*R}$ by construction. The only way to cut a REP string v into two non-repeating strings is that the number of any character in the REP string v is two and the REP string v is palindromic. Cutting the palindromic string v in the middle gets v_1 and v_2, in which $v_1 = v_2^R$. But since $v_i^* \neq v_j^*$, then $v_i^* v_j^{*R}$ cannot be a REP string. □

This implies that each V_i associated with each homotopy class Γ_{h_i} contains only one non-repeating string.

Theorem 1 characterizes the relationship between V_i and its one and only non-repeating string v_i^*.

Theorem 1. *For all* V_i *there exists a unique non-repeating string* $v_i^* \in V_i$*, such that* $\forall v_i^j \in V_i, \Gamma_{v_i^j} \simeq \Gamma_{v_i^*}$.

Proof. Lemma 2 states that such a string much exist and Lemma 3 states that this string is unique. □

The next theorem gives us a construction by which we can identify the non-repeating string representative from for each V_i.

Theorem 2. *Removing all the REP substrings of* $\boldsymbol{M}^h(\sigma)$ *yields the* v_i^* *for which* $\Gamma_{\boldsymbol{M}^h(\sigma)} \simeq \Gamma_{v_i^*}$.

Proof. The differences between $\boldsymbol{M}^h(\sigma)$ and v_i^* are the REP substrings $\boldsymbol{M}^h(\sigma)$. The REP substrings are generated by adding a path segment that leaves a subregion by crossing one or more reference frames and then returning across the same reference frames in reverse order. By Lemma 1, this path segment is homotopic to a path $\sigma \in \Gamma_\varepsilon$ (empty string). By Property 4, we have $\Gamma_{\boldsymbol{M}^h(\sigma)} \simeq \Gamma_{v_i^*}$. □

This gives us a powerful tool for finding when two strings represent homotopic or non-homotopic paths. Consider Algorithm 2, which removes REP substrings using the simple principle that if a character is on the top of the stack when you encounter the next one, you've found a REP substring and should eliminate it from the string.

Algorithm 2 REPTrim(v)

```
stack T = ∅
for char ∈ v do
    if Top(T) == char then
        Pop(T)
    else
        T ← char
return T
```

When combined with Theorem 2, this has the marvelous effect of finding a non-repeating string called REPTrim(v) such that $\Gamma_v \simeq \Gamma_{\text{REPTrim}(v)}$. This yields the following corollary.

Corollary 2. REPTrim($\boldsymbol{M}^h(\sigma_i)$) = REPTrim($\boldsymbol{M}^h(\sigma_j)$) *iff* $\sigma_i \simeq \sigma_j$.

7.4 Homotopy-aware RRT*

Corollary 2 gives us a useful way to determine whether two paths belong to the same homotopy class, but it doesn't tell us anything about how to construct the path that minimizes the cost objective within that homotopy class. This section uses a heuristic based on the REP trim algorithm to restrict paths generated by RRT* to a desired homotopy class. Future work

will explore improved heuristics and better variations of RRT* using the explicit REP Trim algorithm.

RRT* explores the map to generate an optimal tree structure based on the cost distribution on the map. While the tree structure explores the planning space, the DFA $\boldsymbol{M}^h$ can be used to generate the strings of the branches. The string of each branch indicates the homotopic information of the corresponding subpath. The resulting algorithm, Algorithm 3, is called Homotopy-aware RRT* (HARRT*).

Algorithm 3 HARRT* (x_{init}, x_{goal})

$i \leftarrow 0$
$N_s \leftarrow \{x_{init}\}$; $E_s \leftarrow \emptyset$; $T_s \leftarrow (N_s, E_s)$
$N_g \leftarrow \{x_{goal}\}$; $E_g \leftarrow \emptyset$; $T_g \leftarrow (N_g, E_g)$
while $i < N$ **do**
    $T_s, x_s^{new} \leftarrow$ EXPLORE(T_s, i)
    $T_g, x_g^{new} \leftarrow$ EXPLORE(T_g, i)
    $p_s \leftarrow$ CONNECT(x_s^{new}, T_g)
    $p_g \leftarrow$ CONNECT(x_g^{new}, T_s)
    $P \leftarrow$ UPDATEBESTPATHBYCLASS(p_s, P)
    $P \leftarrow$ UPDATEBESTPATHBYCLASS(p_g, P)
    $i \leftarrow i + 1$
$P \leftarrow$ MERGEPATHS(P) **return** P

The algorithm uses a bi-directional structure. There is a *start tree* $T_s = (N_s, E_s)$, which is an RRT* structure from the start position for the optimal cost-to-arrive. N_s is the set of vertices in T_s, and E_s is the set of edges in T_s. Similarly, there is a *goal tree* $T_g = (N_g, E_g)$, which is an RRT* structure from the goal position for the optimal cost-to-go.

In each iteration, a new vertex is created and added to each tree using EXPLORE(). CONNECT() is then called to create a path with a vertex in the other tree. In order to guarantee optimality, a set of near vertices in T_g is provided to find the best vertex to be connected with the new vertex x_s^{new} in T_s, and vice versa. The created path will be compared with the current best path that belongs to the same string block. If it is a better one, the best path in this string block will be updated, which is implemented in UPDATEBESTPATHBYCLASS().

Algorithm 4 EXPLORE(T, i)

$x_{rand} \leftarrow$ SAMPLE(i) ;
$x_{nearest} \leftarrow$ NEAREST(T, x_{rand})
$x_{new} \leftarrow$ STEER$(x_{nearest}, x_{rand}, \eta)$
if OBSTACLEFREE$(x_{nearest}, x_{new})$ **then**
    $s \leftarrow \mathrm{STR}(x_{nearest}) \circ \mathrm{CRF}((x_{nearest}, x_{new}))$
    if STRINGCHECK(s) **then**
        $x_{min} \leftarrow x_{nearest}$
        $X_{near} \leftarrow$ NEAR$(T, x_{new}, |N|)$
        for each $x_{near} \in X_{near}$ **do**
            if OBSTACLEFREE(x_{new}, x_{near}) **then**
                $s \leftarrow \mathrm{STR}(x_{near}) \circ \mathrm{CRF}((x_{near}, x_{new}))$
                if STRINGCHECK(s) **then**
                    if COST(x_{near}) $+c($ LINE(x_{near}, x_{new}) $) <$ COST(x_{new}) **then**
                        $x_{min} \leftarrow x_{near}$
        $E' \leftarrow E' \cup \{(x_{min}, x_{new})\}$
        for each $x_{near} \in X_{near} \setminus \{x_{min}\}$ **do**
            if OBSTACLEFREE(x_{new}, x_{near}) **then**
                $s \leftarrow \mathrm{STR}(x_{new}) \circ \mathrm{CRF}((x_{new}, x_{near}))$
                if STRINGCHECK(s) **then**
                    if COST$(x_{near}) >$ COST(x_{new}) + c(LINE(x_{new}, x_{near})) **then**
                        $x_{parent} \leftarrow$ PARENT(x_{near})
                        $E' \leftarrow E' \setminus \{(x_{parent}, x_{near})\}$
                        $E' \leftarrow E' \cup \{(x_{new}, x_{near})\}$
return T, x_{new}

Algorithm 4 gives the exploration process of a tree structure and is similar with that used in RRT* [59]. The first difference is that the string associated with each branch is updated (implementing $\boldsymbol{M}^h$ on that branch) and the second difference is the use of the StringCheck() method (implementing a heuristic version of the REP Trim algorithm) to check whether the string of a branch satisfies the string constraint. The methods in Algorithm 4 are defined as follows:

- CRF(l): Return the ID characters that represent the crossed reference frames of a line segment l if any.
- STR(x): Return the string that represents the crossed reference frames of the subpath from the root to the node x sequentially. This implements $\boldsymbol{M}^h$.

We assume that a human has specified one or more homotopy classes, and therefore string blocks, as the constraint of the planned paths. StringCheck() compares whether a string of a subpath is a substring of the strings generated by the human. In effect, this eliminates branches that deviate from the non-repeating string representation of the human constraint. It is a heuristic because it does not test whether a branch has a REP substring but rather prevents RRT* from exploring branches that might have such substrings. Notice that the goal tree T_g compares the strings in a reversed order.

Because RRT* maintains a tree structure, each vertex has only one path to arrive from the root. This path, which starts from the root to the vertex, can be converted into a string of ID characters by $\boldsymbol{M}^h$. The StringCheck() guarantees that a new node is added or rewired so that all the branches of the tree structure are in the constraint of strings. For example, suppose we have a string constraint "ab". A branch of the start tree T_s with string "a" satisfies the constraint, because "a" can be extended into "ab" by concatenating a "b". However, a branch beginning with string "b" cannot be extended into "ab", and therefore does not satisfy the string constraint. It is similar for the goal tree T_g but with reversed string order. Note that this is an early check of the homotopy class constraint and may

eliminate some paths that would explore areas outside of the current subregion; we will say more about this in the results section.

Algorithm 5 CONNECT(x_{new}, T)

```
1: p_min = ∅
2: X_near ← NEAR(T, x_new, |N|)
3: for each x_near ∈ X_near do
4:     if ObstacleFree(x_new, x_near) then
5:         if x_new ∈ T_s then
6:             p ← Concatenate(x_new, x_near)
7:         else
8:             p ← Concatenate(x_near, x_new)
9:         if StringCheck(p) and c(p) < c(p_min) then
10:            p_min = p
   return p_min
```

The methods in Algorithm 5 are defined as follows:

- PATH(v, T): Return the path from the root of the tree T to the vertex v.
- CONCATENATE(p_a, p_b): Return a concatenated path of p_a and p_b. If p_a and p_b are from different directions, one of them will be reversed for the concatenation.

When the exploration process is finished, there is a set of the best paths of all the string blocks. By the REP Trim algorithm we can merge the optimal paths in the string blocks that belong to the same homotopy class. The MERGEPATHS() merges the equivalent string blocks into homotopy classes. Thus, the set of paths P will be updated.

7.5 Experiments

Recall the following claim from the introduction: "The contribution of this paper is an algorithm that has guaranteed properties ... [that] create a set of path affordances that allow a human to specify a wide range of hard and soft constraints that the robot is guaranteed to honor when it plans its path." To this point in the paper, all the text has focused on either the theoretic analysis of Palindrome Trim algorithm or the heuristic implementation of this

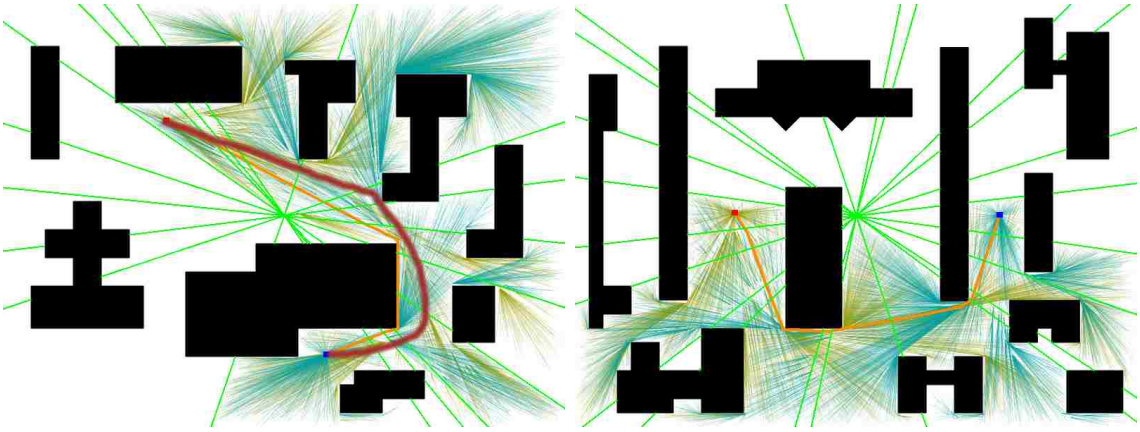
algorithm to help RRT* restrict exploration to a given homotopy class. This section provides evidence that the algorithm can support an important need in HRI.

Consider a path-planning problem where a human supervisor defines the task for a robot. Consider further different ways in which the human can express hard and soft constraints as well as performance objectives. The first two examples of human intent translate into homotopic path constraints over an optimization problem. Note that we use the word "quickly" to represent the optimization criterion; in practice, many possible criteria exist.

1. *Quickly go from point A to point B through a sequence of specific regions.* Topologically, such a path is constrained to one homotopy class, so this homotopy class becomes the constraint of the optimization problem and "quickly" becomes the objective to optimize [49].
2. *Quickly go from point A to point B making sure to visit some regions and avoid other regions.* Topologically, such a path is constrained to be among the set of homotopy classes that include the desired regions and avoid the undesired regions. The corresponding homotopic constraint restricts the optimal path to the set of homotopy classes that satisfy the requirements.

The next example of human intent allow a human to express preference among different path shapes, but also allow the human to trade off between following a desired path shape and optimizing another performance objective.

3. *In quickly going from point A to point B, I prefer some types of paths over others, but I recognize that tradeoffs may be needed.* This indicates that the human has preferences over different homotopy classes, and also acknowledges that certain homotopy classes may not allow an acceptable optimization of another task-based objective. If the preference on the homotopy classes can be modeled using an objective function, then non-dominant solutions over the task-based objective (e.g., "quickly") and homotopic



(a) Example A (b) Example B

Figure 7.5: Optimal paths with hard constraints.

objective (e.g., "north of building A") can be found, and the human can select one of these solutions by balancing tradeoffs.

Clearly, these examples do not cover the set of all possible ways a human can express intent, but they do represent an important (and we would argue a natural) subset of ways that a human can express intent.

In the results in this section, we use the Euclidean distance as the objective to minimize because optimality can easily be verified. The objective can be replaced with any other type.

7.5.1 Single Homotopy Class

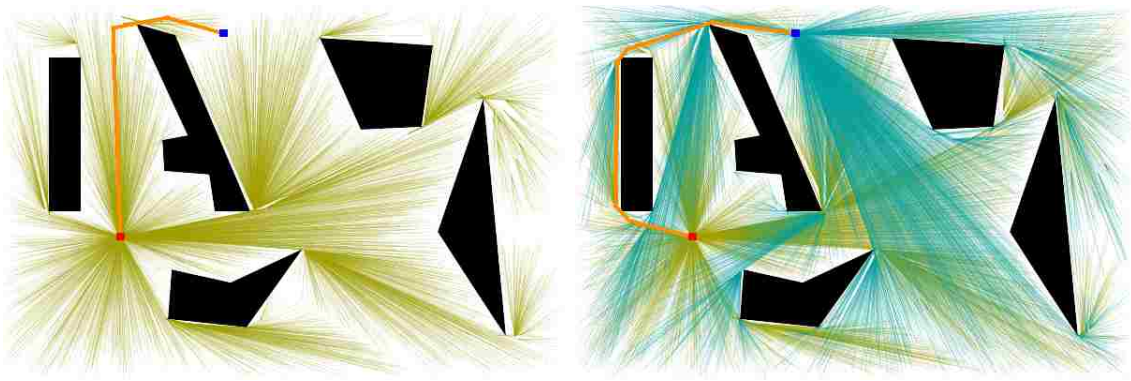
This subsection considers the first way of expressing intent: *Quickly go from point A to point B through a sequence of specific regions.* In this case, the algorithm simply seeks to find the path that minimizes the Euclidean distance between two points subject to the path belonging to a homotopy class.

Figure 7.5 shows results for two different worlds. In Example A, the authors sketched the fuzzy red path, turned that path into a single homotopy constraint, and then used the HARRT* algorithm to find the shortest path that connected the points subject to the constraint. The green radial lines indicate the reference frames, the yellowish lines indicate the branches of the forward tree of the RRT, the turqoise lines indicate the backward tree of

the RRT, and the orange line represents the path found by the algorithm. Inspection shows that that the path is indeed the shortest path within the homotopy class. Further inspection shows that the algorithm concentrated its search effort in the homotopy class, abandoning branches of the tree either when costs became high or when the path crossed a reference frame that deviated from the non-repeating string pattern. Example B shows a similar result but with the human input suppressed to help improve clarity.

Figure 7.5 also illustrates that the current implementation of HARRT* is an approximation. The theoretic results show that we can determine when any two paths are in the same homotopy class by removing REP substrings, but where we check for homotopic consistency in the RRT implementation has a big impact on how big the trees grow in the algorithm. At one extreme, future work should explore whether it is possible (a) to check the homotopy class of the path returned by the RRT-based exploration once a complete path from start to finish has been found and (b) to reject paths that do not satisfy the homotopy constraint. In the current version of HARRT*, the StringCheck() method prevents the extension of the branches into a string that does match the non-repeating string. This is an "early check" on the path that can reject potentially optimal paths within a homotopy class if the optimal path would naturally generate some REP substrings.

As illustrated by the simple examples in this section, this "early check" avoids exploring a lot of the state space, resulting in search efficiency and producing acceptable paths, but it is possible to construct examples where this early check would prevent the discovery of the optimal path. Future work should explore variations of the StringCheck() method that allow a budgeted amount of deviation from a path that generates a non-repeating string. For example, the algorithm could allow a two character palindrome to be part of the string, allowing exploration of paths that leave a subregion to avoid an area of high cost and then return to the subregion once they have circumvented the high cost area. Certainly, this future work would need to explore tradeoffs in the deviation budget, the spacing of sample points that generate the radial structure, and the structure of the cost function.



(a) Single directional tree (b) Bidirectional tree

Figure 7.6: Search results from different structures.

We conclude this subsection with an example world that exposes a problem that is avoided by the bi-directional approach we have taken but which a single direction RRT encounters.

Figure 7.6 gives the results from the single directional and bidirectional tree structures for a problem where multiple homotopy classes are allowed. Each tree structure finds an optimal path in each of several homotopy classes (though the optimal path is shown for only one homotopy class) optimal paths in several homotopy classes but only one of them is illustrated, but the single directional tree structure could not find an optimal path that swings by the left side of the left-most obstacle; see Figure 7.6a. By contrast, the via-point constraints in the bidirectional tree structure enforce the exploration of all the possible homotopy classes; see Figure 7.6b.

Although this result is encouraging, it exposes a challenge in combining RRT* with the Palindrome Trim algorithm or a related heuristic. Stated simply, it is possible for RRT* to "rewire" nodes in such a way that homotopy constraints are violated. Using bidirectional search fixed the problem for this world, but it is easy to construct other worlds for which bidirectional search won't work. Future work will address this challenge.

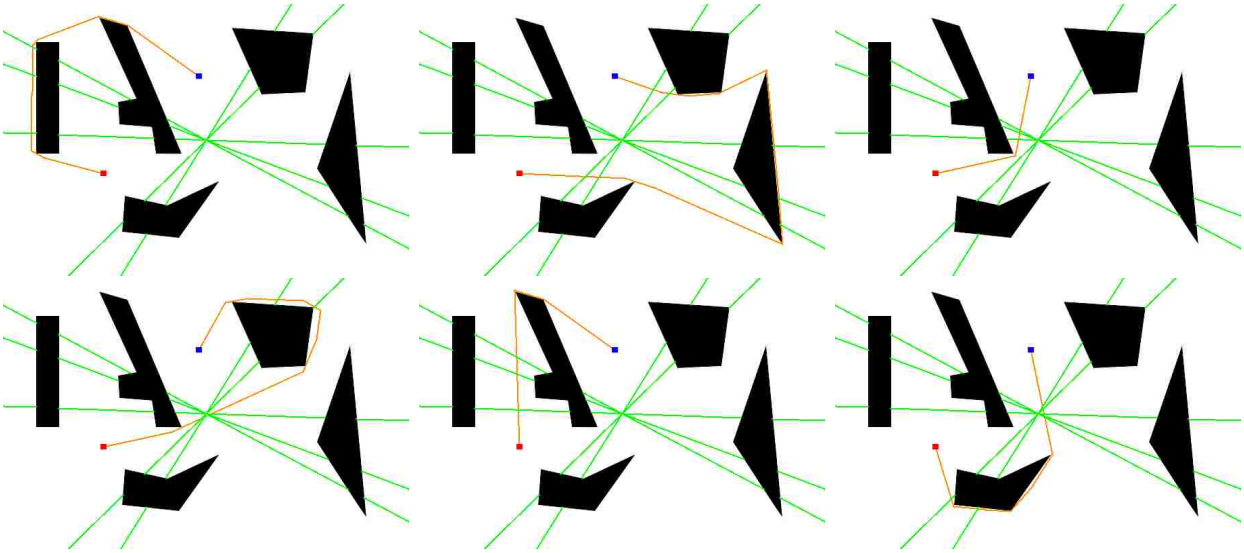


Figure 7.7: Optimal paths in six homotopy classes.

7.5.2 Multiple Homotopy Classes

This section considers the second way of expressing intent: *Quickly go from point A to point B making sure to visit some regions and avoid other regions.* In this case, the set of regions to visit and regions to avoid create a set of possible homotopy classes.

This section gives an example of the kinds of possible solutions that can be generated when multiple homotopy classes are explored simultaneously. Figure 7.7, which will be referenced again in the next section, shows the optimal solutions returned by HARRT* for six different homotopy classes. Observe that each path is optimal for the homotopy class given that the cost function is Euclidean distance. (As an aside, when the cost function is Euclidean distance the heuristic implementation of the REP trim algorithm won't cause problems since the shortest path is also a path that generates a non-repeating string.) For this second type of human intent expression, the path with the lowest cost would be returned.

7.5.3 Soft Constraint and Tradeoffs

This section considers the third way of expressing intent: *In going from point A to point B, I prefer some types of paths over others, but I recognize that tradeoffs may be needed.* This

method is actually quite different than the two previous methods because it treats path shape not as a hard constraint but rather as an objective to be optimized. Fortunately, HARRT* can support a human trying to find tradeoffs between preferences among homotopy classes and the costs associated with choosing a path from a specific homotopy class.

The algorithm for finding these tradeoffs would use HARRT* to find the optimal path from the set of all relevant homotopy classes, as was done in the previous subsection. Recall that HARRT* can simultaneously search in different homotopy classes and returns the solutions in one run. The output of the algorithm would then change its what it presents to the human; rather than returning the best path across homotopy classes as in the previous subsection, the algorithm would find the minimum cost path for each homotopy class and then output both the path and the cost of the path for each homotopy class. This generates a tradeoff space that can be evaluated by the human.

For example, Figure 7.7 shows the optimal paths in six different homotopy classes found by HARRT*. Associated with each class/path pair is the cost of the optimal path. If the cost is displayed for each class/path pair, a human could determine which shape/cost pair provides the best tradeoff.

7.6 Conclusion and Future Work

It is possible to create a computationally efficient algorithm that uses strings to determine when two continuous paths are homotopic. This algorithm can be used as a heuristic in a bi-directional RRT* algorithm to prune paths from the search that are not compatible with an intended homotopy class. Furthermore, the algorithm seems compatible with various degrees of approximation, allowing for tradeoffs between computation speed and quality of the path found by the algorithm, but future work must confirm this.

The paper did not explore the usability, workload, or naturalness of the interactions between the human and the robot afforded by the algorithm, but the empirical results suggest that efficient interactions are plausible given the properties of the algorithm. Future work

should explore how natural language or a graphical user interface can be combined with the algorithm, and whether resulting interactions are compatible with human workload bounds and situation awareness needs.

Finally, future work should explore how the homotopy-aware RRT* algorithm can be extended to problems with multiple performance objectives, enabling either hard shape constraints or tradeoffs between a set of objectives and the shape of the resulting path.

Chapter 8

Topology-Aware RRT* for Parallel Optimal Sampling in Topologies [1]

Abstract

In interactive human-robot path-planning, a capability for expressing the path topology provides a natural mechanism for describing task requirements. We propose a topology-aware RRT* algorithm that can explore in parallel any given set of topologies. The topological information used by the algorithm can either be assigned by the human prior to the planning or be selected from the human in posterior path selection. Theoretical analyses and experimental results are given to show that the optimal path of any topology can be found, including a winding topological constraint wherein the robot must circle one or more objects of interest.

[1] To appear in 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC). Authors are Daqing Yi, Michael A. Goodrich, Thomas M. Howard and Kevin D. Seppi.

8.1 Introduction

Robot path-planning is commonly modeled as path optimization. In path optimization, the topology of the path is often ignored because it can be difficult to quantify, but the topology can be critical to real-world performance. There are many scenarios that require considering the topologies of paths: in collaborative search, the navigation of a robot can be constrained by a human's trajectory [124]; in surveillance, the movement of robot can be constrained to circle regions of interest [83]. In following a demonstration, a robot needs to follow a coarse path topology obtained from a human [105]. Topology can also be used to help find similar paths in re-planning.

In human-robot interaction, specifying path topology is a straightforward way for a human to describe a task. Because a human is better at high-level reasoning, directly expressing a topological requirement simplifies the path planning for a robot. For example, both avoiding risky regions or visiting regions in some temporal sequence define or at least constrain the eligible topological shapes. This kind of topological information often indicates that only part of region needs to be considered in planning, which reduces the planning cost.

We propose a path-planning algorithm that supports topological constraints, including multi-class topological constraints. This algorithm explores in parallel all paths within each of a set of topological constraints and does so using a shared structure for the identical section of topologies. The topological information can either be assigned prior to the planning or be queried during posterior path selection.

8.2 Related Work

In path-planning problems, we are interested in comparing the topologies of two paths σ_1 and σ_2 that share the same start position and the same end position. σ_1 and σ_2 are *homotopic* iff one can be continuously deformed into the other without intersecting any obstacle [11]. A *homotopy class* is defined as a set of paths that are homotopic.

An approximation of homotopy is homology, which can be identified a complex analysis [12]. In this approach the 2D plane is modeled as a complex plane, and there is a point marked as undefined in each obstacle. By comparing the complex integral values along pairs of paths, homologous paths can be identified. In a similar way, paths are classified by homological equivalence by Delaunay-Čech complex values [90].

By approximating obstacles by polygons, a triangulation can be created to generate lines that divide the map [36], but when obstacles are not polygons, the complexity grows quickly. Another approach is to create parallel, non-intersecting rays from representative points in obstacles [62]. These rays are independent of obstacle shape. In order to support winding paths, which are self-intersecting paths (paths with loops), virtual sensor beams created from obstacles have been used to identify homotopies [105, 119]. In a similar way, a radial structure can be used that generates reference frames connecting obstacle [47].

Sampling methods have been widely used to perform efficient path-planning. RRT* [60] exploits sampling efficiency and guarantees that it will find the optimal path in the limit as samples grow. An RRT* approach has also been combined with the ability to identify completely the homotopic equivalence of two paths in 2D using a homotopic Deterministic Finite Automata(DFA) [128]. In that work the homotopic equivalence of two arbitrary paths could be determined using properties of strings recognized by the DFA, but the RRT* implementation did not fully exploit this capability.

Most of these algorithms lack completeness analyses, that is, there is no guarantee that the homotopy class can be identified for every path. Importantly, many of the homotopy- and homology-based algorithms only support finding the shortest path [39] or a feasible path [47] rather than an optimal path with respect to general objective functions. There is still a need for a complete path-planning algorithm that is capable of exploring any topology class, including winding topologies, and that guarantees that the optimal path within the topology class will be found.

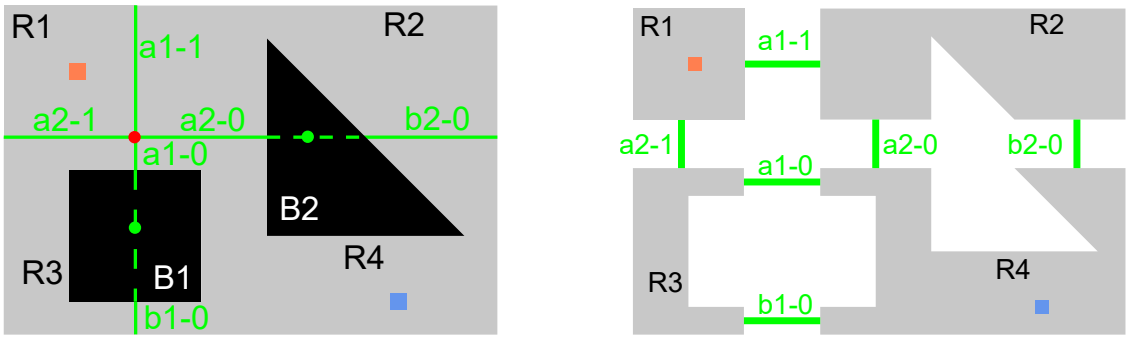
8.3 Path Homotopy Identification

A *simple path* is defined as a path that does not enclose any obstacle and a *simple homotopy class* is a set of simple paths that are homotopic to each other. Prior work showed that an equivalent definition of a simple homotopy class is a homotopy class that contains at least a path that has no duplicate character [128]. This paper extends prior work to the identification to all kinds of paths, including non-simple paths. This section briefly reviews the decomposition method in [128] and proves that homotopic equivalence can be identified by comparing strings.

8.3.1 Homotopic DFA Strings

The following decomposition method, introduced in [128], uses a homotopic DFA, $\boldsymbol{M}^h$, that converts a path σ into a string v. First, a point b_k is randomly sampled in each obstacle $B_k \in \boldsymbol{B}$ as a representative point. A representative point is not allowed to lie on any line that connects any two other representative points. Second, a center point c is randomly sampled in the non-obstacle region which does not lie on any line that can be created from any two representative points from different obstacles. Third, starting at the center point a ray can be created to each representative point. Fourth, the radial structure is cut into a set of line segments by the obstacles. The set of line segments are used as a set of reference frames $\boldsymbol{R}$, which separate the map into a set of subregions $\boldsymbol{S}$. Figure 8.1a shows an example decomposition. In this example, the map is cut into four subregions, $R1$, $R2$,$R3$ and $R4$. The green line segments are the reference frames that define how the subregions are connected. A topology about how the subregions are connected is shown in Figure 8.1b. The start is an orange point and the goal is a blue point.

An *ID character* is assigned to each reference frame so that a sequence of crossed reference frames can be represented by a *string* of ID characters. The resulting strings can be used to identify homotopic equivalence. The homotopic DFA defined in [128] performs the conversion of a path into a string representation $v = \boldsymbol{M}^h(\sigma)$. Homotopic DFA terminology is



(a) Decomposition (b) Topology

Figure 8.1: Map decomposition and its topology.

as follows: the set of reference frames that connect with the center point c is denoted $\boldsymbol{R}_c$ and the set of subregions that connect with the center point c is denoted $\boldsymbol{S}_c$. A new subregion $\widehat{S}_c$ is created by combining all the subregions in $\boldsymbol{S}_c$. A *string block* Γ_v is the set of all paths that yield the same string v.

We now review a sequence of important properties of strings produced by the homotopic DFA [128]. Two paths in the same string block are homotopic — Property 1.

Property 1. $\forall \sigma_i, \sigma_j \in \Gamma_v, \sigma_i \simeq \sigma_j$.

Because all the paths in a string block are homotopic, each homotopy class Γ consists of infinitely many string blocks — Property 2.

Property 2. $\Gamma = \bigcup_{i=1}^{\infty} \Gamma_{v_i}$ *and* $v_i \neq v_j \Rightarrow \Gamma_{v_i} \cap \Gamma_{v_j} = \emptyset$.

The union in Property 2 is over an infinite number of subsets because a sub-path can visit arbitrarily many subregions and backtrack without enclosing an obstacle, which results in strings of arbitrary lengths. Consider a string constructed as follows: begin with the empty string ε and recursively insert a palindromic substring ww^R, where the R operator reverses the characters in the string, into any position of a string. A string made up of *R*ecursively *E*mbedded *P*alindromic substrings is denoted an *REP string* [128]. Note that ε and strings of the form ww^R are REP strings.

The shortest string in a homotopy class, denoted v^*, is not an REP string. Each homotopy class can be represented using this shortest string — Property 3.

Property 3. $\forall v, \exists v^*, \Gamma_v \simeq \Gamma_{v^*}$

Since the shortest path in a homotopy class does not contain a subpath that visits several subregions and backtracks, the Homotopic DFA represents the shortest path in a homotopy class by the shortest string v^*. Algorithm 2 in [128] removes REP substrings by sequentially pushing characters from the string onto a stack *unless* the character at the top of the stack matches the next character in the string [128]. If the top of the stack and the next character match, the stack is popped to eliminate the palindromic structure.

Consider two paths in the same homotopy class, σ and σ^*. Suppose that σ^* is the shortest path. REPTrim() converts string $v = \boldsymbol{M}^h(\sigma)$ into $v^* = \boldsymbol{M}^h(\sigma^*)$ — Property 4.

Property 4. $v^* = \text{REPTrim}(v)$.

The homotopic DFA converts two concatenated paths into two concatenated strings. Because of the recursive nature of Algorithm 2 in [128], these strings decompose into shortest strings — Property 5.

Property 5. $\text{REPTrim}(v_1 v_2) = \text{REPTrim}(v_1^* v_2^*)$
$= \text{REPTrim}(\text{REPTrim}(v_1)\text{REPTrim}(v_2))$.

Algorithm 2 in [128] will never have two consecutive characters the same in the stack. This implies that there are no two consecutive characters that are the same in the output v^*.

Property 6. $\forall i \in \{1, \ldots, |v^*| - 1\} v^*[i] \neq v^*[i+1]$.

8.3.2 Identifying Homotopic Equivalence

We now extend results beyond prior work. We show that we can use the shortest string v^* to determine when two paths are homotopic. This requires two lemmas.

Lemma 1. $\text{REPTrim}(v_i^* v_j^{*R}) = \varepsilon \Leftrightarrow v_i^* = v_j^*$.

Proof. Suppose $\text{REPTrim}(v_i^* v_j^{*R}) = \varepsilon$. When $v_i^* v_j^{*R}$ is input to REPTrim(), v_i^* is pushed onto the stack first. The no duplicate character property of shortest strings, Property 6, means

that the entire string v_i^* will be on the stack. Thus, the length of the stack is $|v_i^*|$. In order to get an empty string ε as the output, the stack needs to be cleared. This requires that $|v_i^*| = |v_j^{*R}|$ and $v_i^{*R} = v_j^{*R}$, which implies that $v_i^* = v_j^*$.

Conversely, when $v_i^* = v_j^*$, simulating stack pushes/pops shows that $\mathrm{REPTrim}(v_i^* v_j^{*R}) = \mathrm{REPTrim}(v_i^* v_i^{*R}) = \varepsilon$. □

We also have Lemma 2 for an equivalence in the path space.

Lemma 2. $\sigma_1 \simeq \sigma_2 \Leftrightarrow \mathrm{REPTrim}(\boldsymbol{M}^h(\sigma_1 \circ \sigma_2^R)) = \varepsilon$.

Proof. Suppose $\sigma_1 \simeq \sigma_2$. Concatenating σ_1 with a reversed σ_2 creates the path $\sigma_1 \circ \sigma_2^R$, which encloses no obstacle by the definition of homotopy. $\sigma_1 \circ \sigma_2^R$ is homotopic to the (closed) path that starts and ends at the same position, $\sigma_1(0)$. Since this closed path encloses no obstacle, $\sigma_1 \simeq \sigma_2 \Rightarrow \sigma_1 \circ \sigma_2^R \simeq \sigma_1(0)$. This means that the closed path $\sigma_1 \circ \sigma_2^R$ recursively visits several subregions and backtracks, returning to the starting point. Applying REPTrim() to such a path yields the empty string.

Conversely, suppose $\mathrm{REPTrim}(\boldsymbol{M}^h(\sigma_1 \circ \sigma_2^R)) = \varepsilon$. We use a proof by contradiction. Assume that $\sigma_1 \not\simeq \sigma_2$. By Property 5, $\mathrm{REPTrim}(\boldsymbol{M}^h(\sigma_1 \circ \sigma_2^R)) = \mathrm{REPTrim}(v_1^* v_2^{*R})$. By Lemma 1, we have $v_1^* = v_2^*$. Because $\sigma_1 \not\simeq \sigma_2$, let $\sigma_1 \in \Gamma_1$ and $\sigma_2 \in \Gamma_2$, we have $\Gamma_1 \not\simeq \Gamma_2$. Let $\mathrm{min\_len}(\Gamma_i) = \arg\min_{\sigma \in \Gamma_i} |\sigma|$ be the shortest path in Γ_i. We have $\mathrm{min\_len}(\Gamma_1) \not\simeq \mathrm{min\_len}(\Gamma_2)$. However, by the definition of the shortest string, $\boldsymbol{M}^h(\mathrm{min\_len}(\Gamma_1)) = \boldsymbol{M}^h(\mathrm{min\_len}(\Gamma_2)) = v_1^* = v_2^*$. By Property 1, it means that $\mathrm{min\_len}(\Gamma_1) \simeq \mathrm{min\_len}(\Gamma_2)$. This is a contradiction. □

By Lemma 2 and Lemma 1, we can derive Theorem 1.

Theorem 1. $\mathrm{REPTrim}(\boldsymbol{M}^h(\sigma_1)) = \mathrm{REPTrim}(\boldsymbol{M}^h(\sigma_2))$ *iff* $\sigma_1 \simeq \sigma_2$.

Proof. Let $v_1^* = \mathrm{REPTrim}(\boldsymbol{M}^h(\sigma_1))$ and $v_2^* = \mathrm{REPTrim}(\boldsymbol{M}^h(\sigma_2))$.

When $\mathrm{REPTrim}(\boldsymbol{M}^h(\sigma_1)) = \mathrm{REPTrim}(\boldsymbol{M}^h(\sigma_2))$, we have $v_1^* = v_2^*$. By Lemma 1, we know $\mathrm{REPTrim}(v_1^* v_2^{*R}) = \varepsilon$. Thus, $\mathrm{REPTrim}(\boldsymbol{M}^h(\sigma_1)\boldsymbol{M}^h(\sigma_2)^R) = \varepsilon$. By Lemma 2, we have $\sigma_1 \simeq \sigma_2$.

When $\sigma_1 \simeq \sigma_2$, we can create a path $\sigma_1 {\sigma_2}^R$ by concatenating σ_1 and a reversed σ_2. Without loss of generality, we can assume that this path starts and ends at the same position. By Lemma 2 and Property 5, we have $\varepsilon = \text{REPTrim}(\boldsymbol{M}^h(\sigma_1 {\sigma_2}^R)) = \text{REPTrim}(v_i^* {v_j^*}^R)$. By Lemma 1, we know $v_i^* = v_j^*$, which implies $\text{REPTrim}(\boldsymbol{M}^h(\sigma_1)) = \text{REPTrim}(\boldsymbol{M}^h(\sigma_2))$. □

Theorem 1 tells us that we can identify the homotopy of paths by comparing the strings generated by the homotopic DFA after passing them through REPTrim Algorithm [128]. We use this to create an RRT*-based planner that only generates solutions constrained to one or more homotopy classes.

8.4 Topology-Aware RRT*

In this section, we propose a topology-aware random sampling algorithm that is derived from the standard RRT* [60]. Assumptions from [60] about RRT* are inherited here. We are interested in both non-simple paths and non-simple homotopy classes, but Property 2 implies that there exists an infinite number of possible string blocks in a homotopy class. If we assume the optimal path is not an infinitely long path (which will be true for cost functions that monotonically increase with path length), the string produced by homotopic DFA for optimal path will not be infinitely long. We state this assumption and use it to limit the number of possible subtrees produced by our algorithm.

Assumption 1. $\forall \sigma^* = \arg\min_{\sigma \in \Gamma} \textsc{Cost}(\sigma), \exists \tau \geq 1, |v| \leq \tau |v^*|$ *and* $\sigma^* \in \Gamma_v$.

In this assumption, the topological constraint is represented by the set Γ that contains all homotopy classes consistent with this constraint. We assume that we can restrict search to paths that (a) are in a homotopy class consistent with an optimal path and (b) do not produce really long strings.

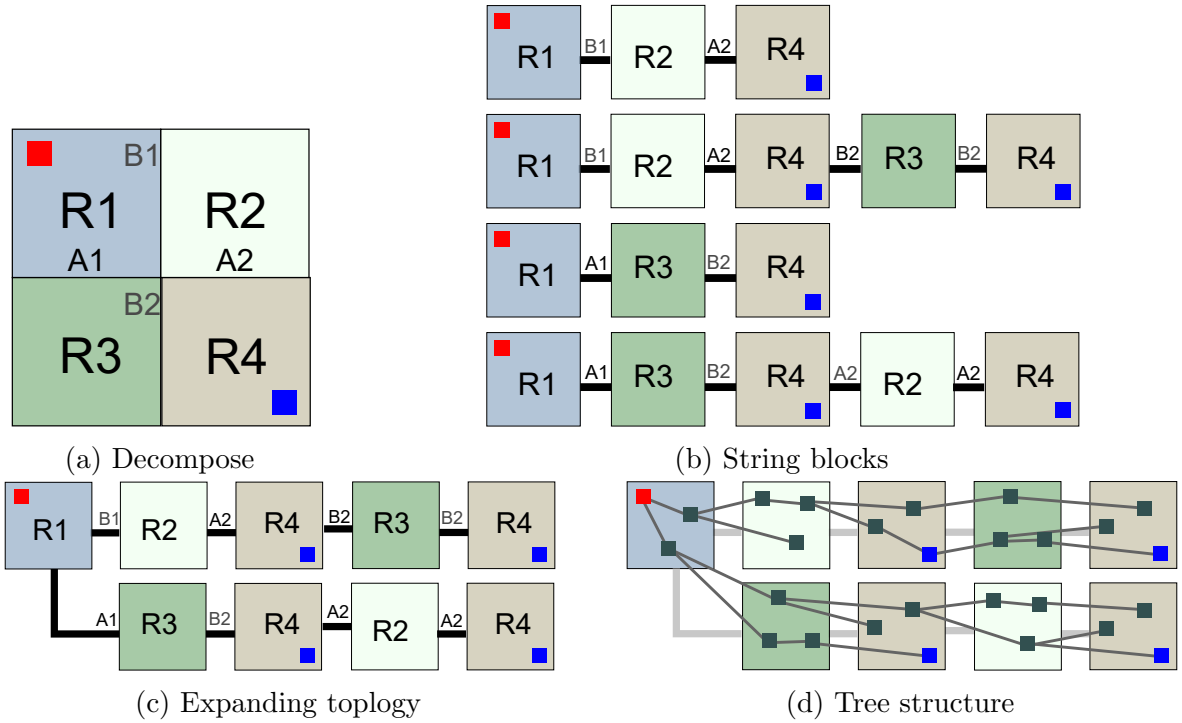
8.4.1 Expanding Topology

The basic idea of the Topology Aware RRT* (TARRT*) algorithm is that the tree produced by RRT* is "chunked" into subtrees when the paths in those subtrees produce different string blocks; extension of the trees is restricted to only those string blocks that are consistent with the desired topological constraint(s). The RRT* subtrees find the optimal path within each string block, and the optimal path for the homotopy class is the best of the paths for the string blocks.

We will be talking about two different trees: the tree produced by RRT* and the tree of subtrees produced when we group the RRT* branches into string blocks. To help distinguish between the elements of the tree of subtrees and the elements of the tree, we will refer to the subtrees that belong to the same string block as a *TARRT* node* and the individual elements of the complete tree as *RRT* vertices*.

Figure 8.2a illustrates a very simple world with four regions. These four regions are separated by four reference frames, "A1,B1,A2,B2", and the homotopic DFA adds the label for these references frames to the string whenever the path crosses the reference frame. The small red square indicates the start position and the small blue square indicates the goal position. Thus, they represent different string blocks and different sequences of TARRT* nodes that can be created.

Each TARRT* node is associated with a subregion, and each edge in TARRT* is associated with a reference frame. There exist similarities between string blocks. For example, all the string blocks start in the same initial subregion "R1". The string block "B1,A2,B2,B2" contains a substructure that is identical with the string block "B1,A2". We can thus use an expanding topology to efficiently express these string blocks, as shown in Figure 8.2c. The *root TARRT* node* is always associated with the start subregion. Denote a TARRT* node associated with the goal subregion as a *terminal TARRT* node*. Any path from the root expanding node to a terminal TARRT* node defines a string block, which is called a



(a) Decompose (b) String blocks

(c) Expanding toplogy (d) Tree structure

Figure 8.2: Expanding Topology.

string-block branch of the TARRT* tree. In Figure 8.2c, each path from the TARRT* node "R1" to a TARRT* node "R4" is within one of the string blocks in Figure 8.2b.

RRT* uses directed random sampling to create new possible nodes in the RRT* subtrees. Since each one of the new possible RRT* vertices is located in a subregion, it is possible that the location of the new node can be part of multiple string blocks and their corresponding TARRT* nodes. If we can generate an optimal structure like RRT* but sorted by string blocks, backtracking from a goal position in a terminal expanding node to the root obtains the optimal path of the corresponding string block.

8.4.2 Topology-Aware Space Sampling

The TARRT* algorithm is given as Algorithm 1. It inherits optimal spatial sampling from RRT* but the tree generation process is guided by an expanding topology of TARRT* nodes. The branches of the tree are sorted by string-block branches of the expanding topology, like in Figure 8.2d.

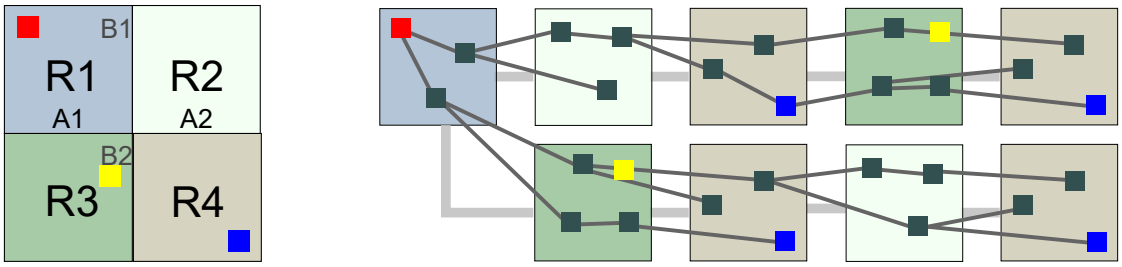
The algorithm enforces a tree-of-subtrees structure by ensuring that the parent RRT* vertex of any RRT* vertex can only be located (a) within the same TARRT* node as the RRT* vertex or (b) in the parent node of that TARRT* node. Moreover, if an existing RRT* vertex is linked with a new RRT* vertex, the edge between those vertices visit reference frames as defined in the expanding topology of TARRT* nodes.

For example, consider the string block "B1, A2" at the top of Figure 8.2b and an RRT* vertex in the TARRT* node "R4". If the RRT* vertex has a parent in TARRT* node "R2", the edge between child and parent should cross the reference frame "A2". If the RRT* vertex has an edge connecting to a grandparent RRT* vertex in TARRT* node "R1", the edge should cross the reference frames "B1","A2" sequentially. If an RRT* vertex has an edge towards a node in the TARRT* node "R3", the edge violates the requirement of this string block and should not exist.

Algorithm 1 Topology-Aware Rapidly-exploring Random Tree* $G(V, E)$

```
1: V ← {x_init}; E ← ∅; i ← 0
2: while i < N do
3:     x_rand ← SAMPLE (i) ; i ← i + 1
4:     x_nrst ← NEAREST (G, x_rand)
5:     x_new ← STEER (x_nrst, x_rand, η)
6:     if OBSTACLEFREE(x_nrst, x_new) then
7:         R = REGION ( x_new )
8:         for each tarrt_node in TARRTNODES ( R ) do
9:             tarrt_node ← x_new
10:            G ← EXTEND (G, x_new, x_nrst)
```

Because a subregion is associated with multiple TARRT* nodes, for each new position obtained via directed sampling in RRT* a new RRT* vertex will be created in each relevant TARRT* node; relevant TARRT* nodes are those associated with the subregion in which the new position lies. This means that when a new position is sampled, there are new RRT* vertices created in several associated TARRT* nodes. For example, a new position (the yellow square) is sampled in the subregion "R3", as illustrated in Figure 8.3a. Two new



(a) Sampling (b) Create new nodes

Figure 8.3: Sampling and adding new nodes.

RRT* vertices of the position are added to the two TARRT* nodes, one for the top string block topology and another for the bottom string block topology as shown in Figure 8.3b.

We now define several functions, using appropriately modified definitions from the RRT* algorithm in [60].

- SAMPLE(): Returns independent uniformly distributed samples from X_{free}.
- NEAREST(): Returns a position of the vertex whose position is closest to point x. NEAREST$(G = (V, E), x) = \arg\min_{v \in V} \|x - v\|$.
- STEER(): Given two points x and y, returns a point z on the line segment from x to y that is no greater than η from y. STEER$(\ x, y, \eta\ ) = \arg\min_{z \in \boldsymbol{R}^d, \|z-x\| \leq \eta} \|z - y\|$.
- OBSTACLEFREE(x, x'): Returns *True* if $[x, x'] \subset X_{free}$, which is the line segment between x and x' lies in X_{free}.
- REGION(x): Returns the subregion that position x is in.
- TARRTNODES(R): Returns all TARRT* nodes from the expanding topology that are associated with subregion R.

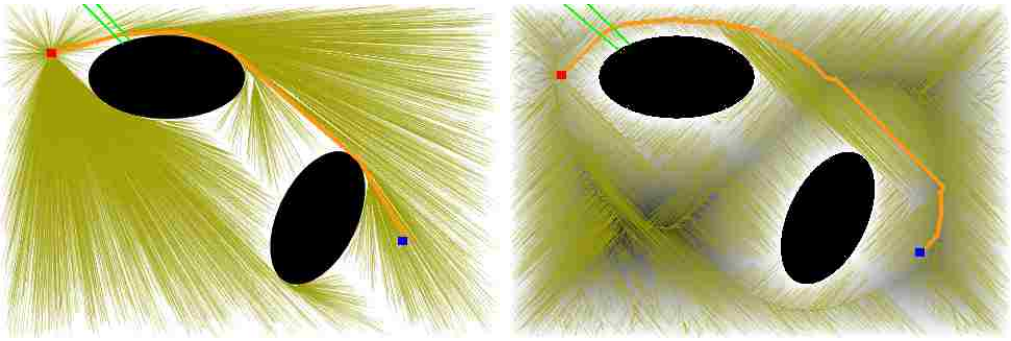
The RRT* vertices of the TARRT* tree are created and stored in TARRT* nodes. This provides information for how to add connections between new positions to potential parent RRT* vertices and also how to rewire RRT* vertices so that rewiring honors string block constraints. Thus, the EXTEND procedure of TARRT* is slightly different with that in RRT*. It is stated in Algorithm 2.

The precise definitions of the methods used in the Algorithm 2 are given below.

Algorithm 2 EXTEND$(G, x_{new}, x_{nearest})$

if $x_{new} = x_{nrst}$ **then return** $G = (V, E)$
$V' \leftarrow V \cup \{x_{new}\}$
$x_{min} \leftarrow x_{nrst}$
$X_{near} \leftarrow \text{NEAR}(G, x_{new}, |v|)$
for each $x_{near} \in X_{near}$ **do**
    if OBSTACLEFREE(x_{new}, x_{near}) **and** HOMOTOPYELIGIBLE(x_{new}, x_{near}) **then**
        $c'_k \leftarrow \text{COST}_k(x_{near}) + c_k(\text{ LINE}(x_{near}, x_{new})\ )$
        if $c'_k < \text{COST}_k(x_{new})$ **then**
            $x_{min} \leftarrow x_{near}$
$E' \leftarrow E' \cup \{(x_{min}, x_{new})\}$
for each $x_{near} \in X_{near} \setminus \{x_{min}\}$ **do**
    if OBSTACLEFREE(x_{near}, x_{new}) **and** HOMOTOPYELIGIBLE(x_{near}, x_{new}) **then**
        $c'_k \leftarrow \text{COST}_k(x_{new}) + c_k(\text{ LINE}(x_{new}, x_{near})\ )$
        if $c'_k < \text{COST}_k(x_{near})$ **then**
            $x_{parent} \leftarrow \text{PARENT}(x_{near})$
            $E' \leftarrow E' \setminus \{(x_{parent}, x_{near})\}$
            $E' \leftarrow E' \cup \{(x_{new}, x_{near})\}$
return $G' = (V', E')$

- NEAR$(G, x, card)$: Returns all vertices within the closed ball of radius $\gamma = \min\{\gamma_{RRT*}(\log(card)/card)^{1/d}, \eta\}$ centered at x, in which $\gamma > (2(1 + 1/d))^{1/d}(\frac{\mu(X_{\text{free}})}{\zeta_d})^{1/d}$ [60].
- HOMOTOPYELIGIBLE(x_{from}, x_{to}): Uses REPTRIM() to return *True* if the sequence of reference frames a line visits is consistent with a required sequence of reference frames. The line is from x_{from} to x_{to}. The required sequence of reference frame is obtained by the sequence of edges from the TARRT* node that x_{from} is in to the TARRT* node that x_{to} is in.
- LINE$(x, x') : [0, s] \leftarrow X_{free}$ denotes the path defined by line segment from x to x'.
- COST(x): Returns cost of the unique path (because G is a tree) from x_{init} to the vertex $x \in V$. COST$(x_{\text{init}}) = 0$.



(a) Minimum Distance (b) Maximum Safety

Figure 8.4: Optimality. $\tau = 1$.

8.5 Experiments

This section presents a series of informative examples that illustrate how the TARRT* algorithm works, that provide empirical support for the claims in the paper, and that illustrate some useful properties of the algorithm. The examples include some use cases. The first use case is when a human specifies a single homotopy class as the topological constraint and the algorithm returns the optimal path subject to that constraint. The second use case is when the human specifies some things to avoid; these are translated into a topological constraint that includes multiple homotopy classes, and then the human selects from multiple possible paths returned by the algorithm.

8.5.1 Optimality and Practicality

Figure 8.4 illustrates TARRT* for a homotopy class where the path is required to go above the obstacle at the top of the world. Black blobs indicate obstacles. The orange line is a found path from the start (red point) to the goal (blue point). The olive lines visualize the tree structure generated by TARRT*. There are also green line segments that show the reference frames associated with a string block in a homotopy class.

The algorithm behavior when path length is the objective is given in Figure 8.4a and when minimizing distance to any obstacle is the objective is given Figure 8.4b. The gray background shows the cost map distribution; darker means lower cost and the lighter

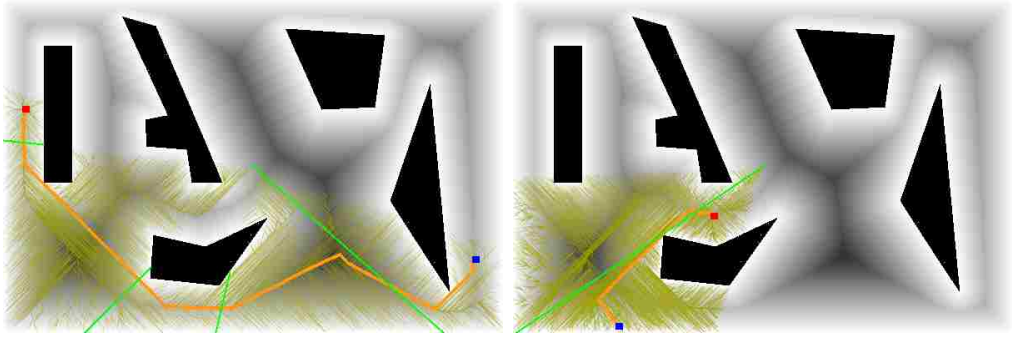


Figure 8.5: Non-winding topology.

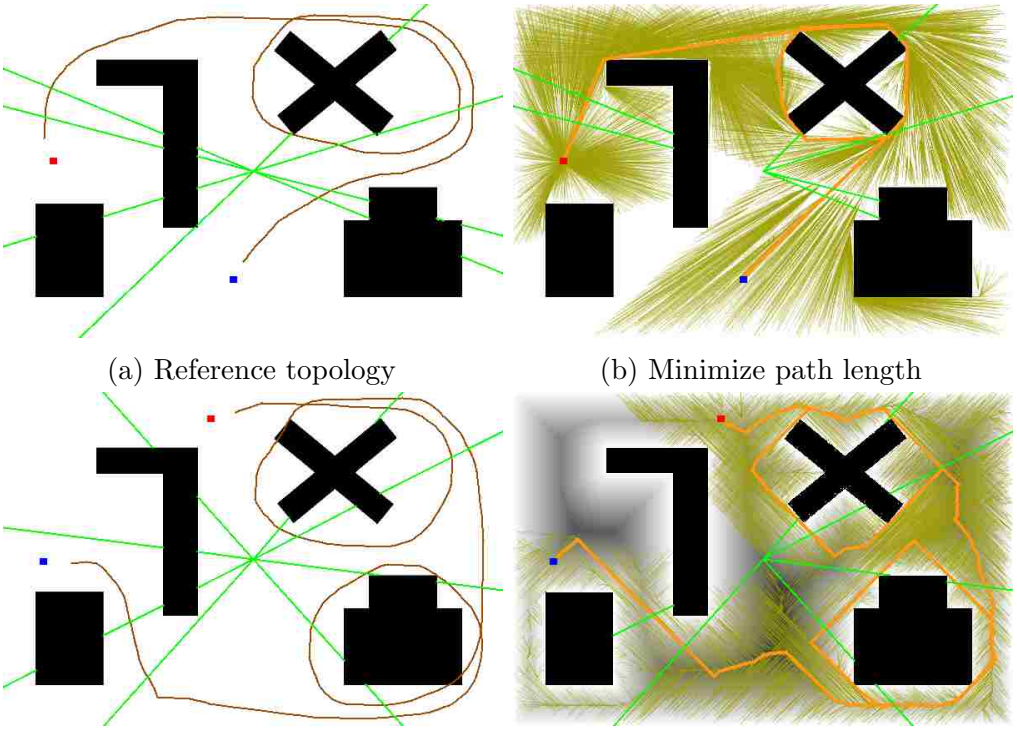
means higher cost. Because the two reference frames happened to be very close together, the algorithm wastes a lot of time exploring to the lower left of the obstacles. Nevertheless, the algorithm returns the shortest path and the minimum cost path, respectively, in the world.

8.5.2 Use Case 1

In a human-to-robot approach, a homotopy class is provided a priori by a human. Consider the world illustrated in Figure 8.5 and two constraints provided by the human. In the first constraint, the human draws a path that goes south of every obstacle. In the second constraint, the human draws a path that goes to the west of the obstacle in the lower center of the world. The human-drawn paths were translated into shortest strings using the REPTime algorithm, and then string blocks that were consistent with these shortest strings were passed to TARRT*.

Because a homotopy class from a human is given, subregions that are not associated with the homotopy class need not be explored. As a result, the tree extends only to part of the map and consists of fewer subregions. Figure 8.5 illustrates two examples where the optimal paths for a cost of avoiding near approaches to obstacles can be found even when only part of the map is explored. A tight constraint enhances efficiency.

So far, the examples have considered only simple homotopy classes. TARRT* is able to find the optimal path of a winding topology. Figure 8.6b shows the optimal path of a homotopy class that contains single windings of two different obstacles using minimum path length as the objective. The reference topology was drawn by a human. Figure 8.6d uses a



(a) Reference topology (b) Minimize path length

(c) Reference topology (d) Maximize safety

Figure 8.6: Wwinding topology.

topology constraint that winds around two obstacles for a cost function related to obstacle proximity (darker is better).

8.5.3 Use Case 2

Suppose that the human only specifies a region to avoid, leaving many different homotopy classes that might satisfy this constraint. In the extreme, suppose that the human wants to see the lowest cost path for every possible topology. Finally, suppose that the human specifies an upper bound on how long the path can be, which turns the problem from one of searching through an infinite number of possible homotopy classes to searching the finite number of classes that satisfy the stretching assumption (Assumption 1). Figure 8.7 illustrates TARRT* exploring different homotopies in parallel. Each subfigure shows the portion of the TARRT* tree relevant for one of the homotopy classes along with the best path from that homotopy class.

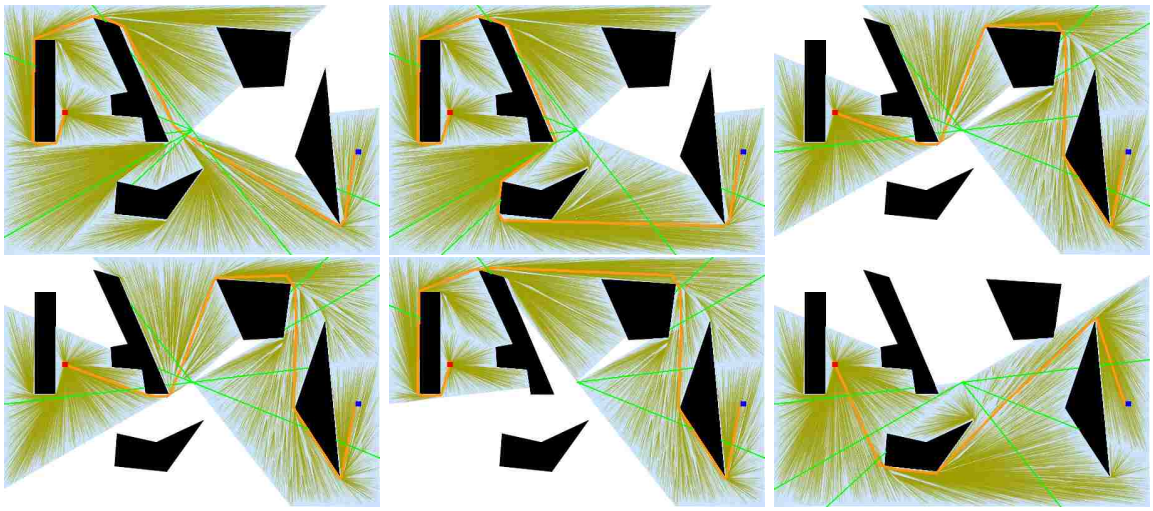


Figure 8.7: Multiple string blocks.

8.6 Summary

In this paper, we proved that the decomposition method and homotopic DFA can be applied not only to simple paths but also to non-simple paths. We also proposed the TARRT* algorithm, which provides an efficient sampling structure for exploring a topological constraint over multiple homotopy classes. TARRT* enforces samplings that honor a set of possible homotopy classes and rewires the RRT* tree so that it explores multiple homotopy classes in parallel. TARRT* finds a solution for any homotopy class with finite length paths including winding topologies.

Future work should apply TARRT* to different decomposition methods and in higher dimension spaces such as in a robotic manipulation problem using a 3D decomposition method. Practical complexity of the algorithm will likely become an issue for these problems.

Chapter 9

Sampling-based Optimal Path Planning with Homotopy Class Constraints[1]

Abstract

Supporting homotopy class constraints enables a hierarchical framework of cooperative path-planning consisting of high-level topological reasoning and low-level workspace planning. Two phases of human interaction support this cooperative path-planning: (i) a human provides one or more path topologies to guide the path-planning of a robot, and (ii) a human select one path from the set of "best" paths associated with each path topology. This paper enables these two phases by introducing a homotopic DFA that converts a path into a string representation for identifying the homotopic equivalence of two paths. Embedding the homotopic DFA in an RRT* structure allows the homotopic class of each branch to be identified.

We propose two RRT*-based path-planning algorithms that use homotopic class information, and show how these algorithms can be used to find optimal paths from certain types of homotopy classes. The first algorithm uses a bidirectional structure that generates paths from within a certain kind of homotopy subclass by via-point constraints so that different homotopy subclasses can be efficiently explored. The second algorithm follows a sequence-guided RRT* structure that guides the direction of branching, and finds optimal paths from with general types of homotopy classes. We present a formal analysis of when (i) solutions converge to optimal paths of a given homotopy classes or subclasses and (ii) all

[1]In preparation to be submitted to *IEEE Transactions on Robotics*. Authors are Daqing Yi, Michael A. Goodrich, Kevin D. Seppi and Thomas M. Howard.

homotopy classes can be explored, including winding homotopy classes. Empirical evaluations are presented that support the theoretical analyses.

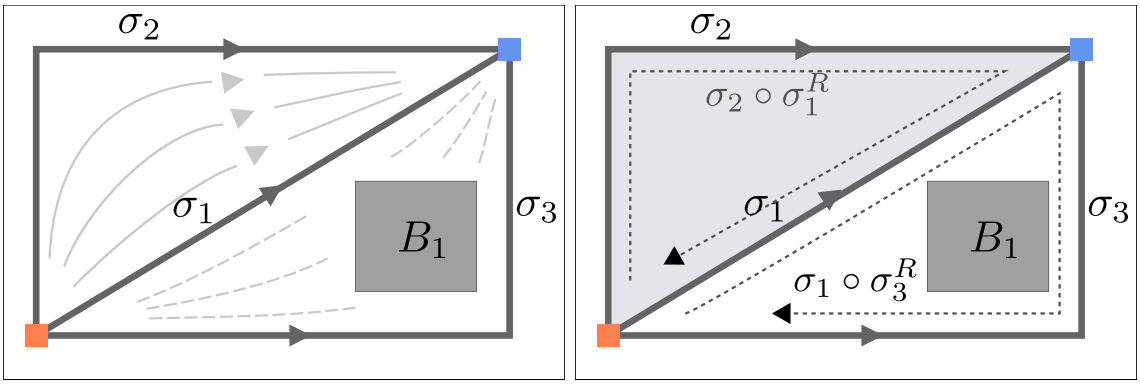
9.1 Introduction

In many applications, a human will care about the shape of a path, but path shape is often neglected in optimal path-planning, partly because path shape is difficult to encode as an objective function to be optimized. This paper presents two algorithms for path-planning that find optimal paths subject to one or more homotopy constraints, plus a small number of technical path constraints, under the assumption that a human provides these constraints. These path topologies indicate that only part of workspace needs to be considered in planning if the constraint can be used to reduce the size of the search space that must be explored.

Consider how path shape, more precisely called topological information, is an important human consideration in the following different scenarios: in collaborative search, the trajectory of a human constrains the navigation of a robot who executes a subtask [123]; in surveillance, a team supervisor gives a rough execution plan in the form of path topology that requires the robot to maximize the performance subject to a topological constraint [83]; in following a demonstration, a human sketches a coarse path topology to guide a robot's navigation [105]. Moreover, real-time re-planning can theoretically use path topology to efficiently find similar paths [20].

We are interested in path-planning problems where path shape is expressed as a topological constraint. Such constraints can define single or multiple path topologies. For a single-topology constraint, a planner only explores a partial workspace that is determined by the topological information. For a multi-topology constraint, a planner parallel explores multiple path topologies and may return multiple paths, one each for each topology.

This paper is extended from prior work by the authors in [128] and [127]. The paper is outlined as follows: we introduce homotopic DFA that converts any path into a string. The string that represents the topological information can be used to compare topological similarity [128]. Two different path-planners were proposed in the prior work:, HARRT* [128] and TARRT*[127]. These two planners have many things in common, so it is useful to study them together. This paper adds a complete theoretical analyses of HARRT* and TARRT*,



(a) Homotopy (b) Homology

Figure 9.1: Homotopy and homology of paths.

which includes optimality and completeness in homotopy-class exploration. Additionally, more experiments are provided that help explain similarities, strengths and limitations of the two algorithms.

9.2 Related Work

Consider path-planning problems where all feasible paths share the same start position and end position. For such problems, the mathematical notions of *homotopy* and *homology* are often used to compare the topologies of two paths. Two paths are *homotopic* iff one can be continuously deformed into the other without intersecting any obstacle [11]. An example is given in Figure 9.1a, which illustrates how path σ_1 can be deformed into σ_2 but not into σ_3 because of obstacle B_1. σ_1 and σ_2 are *homologous* iff concatenating σ_1 with a reversed σ_2 forms the complete boundary of a 2-dimensional manifold embedded in the space not containing or intersecting any of the obstacles [11]. An example is given in Figure 9.1b, which illustrates how the manifold created from σ_1 and σ_2 intersects no obstacle but the manifold created from σ_1 and σ_3 would intersect an obstacle.

Paths can be categorized into classes by the equivalence in homotopy or homology. A *homotopy class* (respectively *homology class*) is defined as a set of paths that are homotopic (respectively, homologic). When two paths are homotopic, they are also homologous [83], but

the converse is not always true [11]. This means that a homology class is a close analog of homotopy class but they are not equivalent.

There have been algorithms created that use homologies to support path-planning. [12] proposed a method of homology class identification by a complex analysis approach. In this approach the 2D plane is modeled as a complex plane, and there is a point marked as undefined in each obstacle. By comparing the complex integral values along pairs of paths, homologous paths can be identified. In a similar way, paths are classified by homological equivalence by Delaunay-Čech complex values [90].

Homotopy defines a stricter equivalence comparison, thus its identification is harder. One approach to this problem is to use one of several methods (see below) that convert paths into string representations. In these approaches, the map is decomposed into subregions by line segments. The topology of a path is then associated with a sequence of visited subregions or, equivalently, a sequence of crossed line segments. Each subregion or line segment is identified by a token, and a topology is identified by a string of tokens. In such an approach homotopic equivalence is determined by comparing strings.

By approximating obstacles by polygons, a triangulation can be created to generate lines that divide the map [36, 49], but when obstacles are not polygons, the complexity grows quickly. Another approach is to create parallel, non-intersecting rays from representative points in obstacles [62]. These rays are independent of obstacle shape.

In order to support winding paths, which are self-intersecting paths (paths with loops), virtual sensor beams created from obstacles have been used to identify homotopies [83, 119]. In a similar way, a radial structure can be used that generates reference frames connecting obstacle [47].

Most of these algorithms lack completeness analyses, that is, there is no guarantee that the homotopy class can be identified for every path. For example, when the strings generated by two paths are different, it may be unknown whether the two paths are in the same homotopy class. Importantly, many of the homotopy- and homology-based algorithms

only support finding the shortest path [39, 44] or a feasible path [47] rather than an optimal path with respect to general objective functions.

Sampling methods have been widely used to perform efficient path-planning. RRT* [60] exploits sampling efficiency and guarantees that it will find the optimal path in the limit as samples grow. An RRT* approach has also been combined with the ability to identify completely the homotopic equivalence of two paths in 2D using a homotopic Deterministic Finite Automata (DFA) [128]. In that work the homotopic equivalence of two arbitrary paths could be determined using properties of strings recognized by the DFA, but the RRT* implementation did not fully exploit this capability.

There is still a need for a complete path-planning algorithm that is capable of exploring any topology class, including winding topologies, and that guarantees that the optimal path within the topology class will be found. We propose a sampling-based algorithm that should work with many different decomposition techniques [39, 119], but we demonstrate results using the homotopic DFA method from.

9.3 Homotopy Identification

Given the information from the previous section, we can define the problem the algorithm must solve.

Definition 1. ***Homotopy-based Optimal Path-Planning*** *Let $X \subset \mathbb{R}^d$ denote a bounded connected open set, $X_{obs} \subset X$ an obstacle space, $X_{free} = X \setminus X_{obs}$ the obstacle-free space, x_{init} an initial state , and x_{goal} a goal state. Define a* path *in X as a continuous curve parameterized by s as $\sigma(s) : [0,1] \to X$. Denote a strictly monotonically increasing cost of the path as $\textsc{Cost}(\sigma)$. Let $\boldsymbol{H}(x_{init}, x_{goal})$ denote the set of homotopy classes defined by $x_{init} \in X_{free}$ and $x_{goal} \in X_{free}$, let $H = \{h_1, \cdots, h_N\} \subseteq \boldsymbol{H}$ denote a particular subset of homotopy classes, and let $h(\sigma)$ denote the homotopy class of σ. For each $h_i \in H$ the goal is to find paths $\sigma^*_{h_i}$ such that (a) $\forall s \in [0,1], \sigma^*_{h_i}(s) \in X_{free}$; (b) $\sigma^*_{h_i}(0) = x_{init}$ and $\sigma^*_{h_i}(1) = x_{goal}$;*

and (c) $\sigma^*_{h_i} = \arg\min_{\sigma \in X_{free} \wedge h(\sigma)=h_i} \textsc{Cost}(\sigma)$. $\Sigma^*(H) = \{\sigma^*_{h_i} : h_i \in H\}$ *denotes the set of optimal paths subject to the set of allowed homotopy classes.*

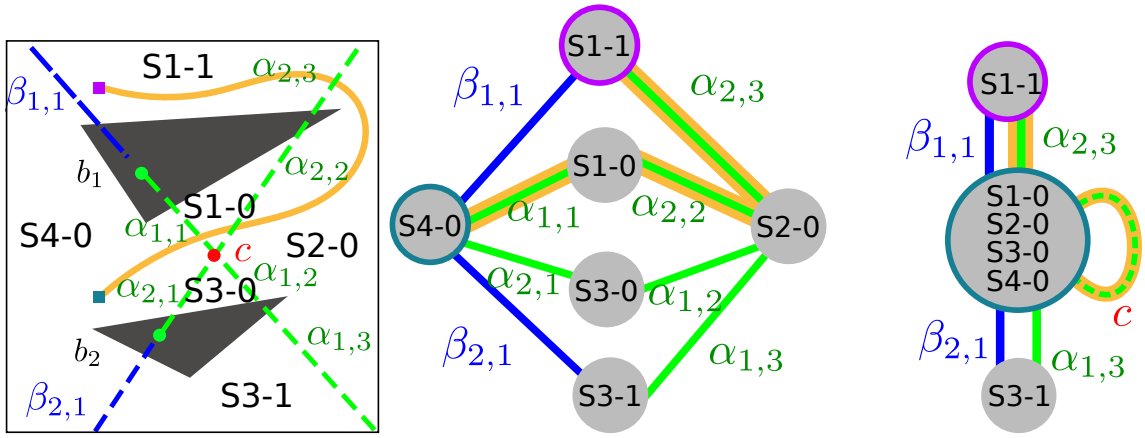
Definition 1 says that, given a particular set of homotopy classes and a cost function, the planner should find paths that minimize the cost in the given set of homotopy classes. Because we assume that the cost function is monotonically increasing, we can have Property 1.

Property 1. $\forall h_i,\ \sigma^*_{h_i}$ *has finite length.*

In order to find the optimal path in a homotopy class, we need to be able to represent and identify the homotopy class of all paths. Our homotopy identification process converts any path into a string so that identifying homotopy equivalence is mapped into a problem of identifying string equivalence. Strings are generated using an improved method of Jenkins' approach [53] to detect homotopic equivalence of two paths by separating a map into disjoint subregions [47]. A reference frame segment, which Jenkins called a *reference frame*, is a line segment constructed from a center point and a point in an obstacle, extended to the map boundaries. The collection of reference frames created from a set of obstacles partitions the map into disjoint subregions. Figure 9.2a shows an example of a map with two obstacles and two reference frames (blue and green dashed lines) – one for each obstacle. Strings will be constructed based on the simple idea that if two paths cross the same sequences of reference frames, then they belong to the same homotopy class.

9.3.1 String representation

Algorithm 1, given below, creates reference frames $\boldsymbol{R}$ from a set of points that are generated as follows: In a map with a set of obstacle regions X_{obs} clustered in a finite set of obstacle regions B_k such that $X_{obs} = \cup_k B_k$, an obstacle point b_k is randomly sampled from each obstacle region $B_k \in X_{obs}$. A center point c is then randomly sampled in the non-obstacle region $X_{free} = X \setminus X_{obs}$ subject to the constraint that it is not in any line that connects two different b_k. Connecting each b_k with c creates a radial structure of reference frames that partition the map.



(a) Ref. frames (b) State trans. (ST) (c) Homotopic ST

Figure 9.2: Map with obstacles.

Algorithm 1 InitRefFrames (X_{free}, X_{obs})

$\boldsymbol{R} = \emptyset, \boldsymbol{b} = \emptyset$
for each $B_k \in X_{obs}$ **do**
    $\boldsymbol{b} \leftarrow \boldsymbol{b} \cup \{b_k\}$ randomly sampled from B_k
$c \leftarrow$ Randomly sampled from X_{free}
while $\exists b_k, b_{k'}, c \in$ Line$(\ b_k, b_{k'}\ )$ **do**
    $c \leftarrow$ Randomly sampled from X_{free}
for each $b_k \in \boldsymbol{b}$ **do**
    $l_k \leftarrow$ Line$(\ b_k, c\ )$
    $\{l_{k_m}\} \leftarrow$ Intersect(l_k, X_{obs}, c)
    $\boldsymbol{R} \leftarrow \boldsymbol{R} \cup \{l_{k_m}\}$
return $\boldsymbol{R}$

The method $\textsc{Line}(p_1, p_2)$ returns the line defined by p_1 and p_2, and the method $\textsc{Intersect}(r, X_{obs}, c)$ returns all segments of line r that don't intersect with an obstacle in X_{obs} or the center point c.

If we assign an ID character to each reference frame, then how a path sequentially crosses the reference frames can be converted into a string of ID characters. For example, in Figure 9.2a, the path that starts in subregion S_{4-0} and ends in subregion S_{1-1} sequentially visits the reference frames $\alpha_{1,1}, \alpha_{2,2}, \alpha_{2,3}$. Concatenating these characters yields the path string $\alpha_{1,1}\alpha_{2,2}\alpha_{2,3}$. A deterministic finite automata (DFA) formalizes the process of translating a path into a path string; see Figure 9.2b.

Definition 2. *Let $\boldsymbol{M} = (\boldsymbol{S}, \boldsymbol{R}, \delta, S_0, S_T)$ be a DFA that represents the string generation process from a path, where $\boldsymbol{S}$ is a set of subregions, $\boldsymbol{R}$ is a set of reference frames, $S_0 \in \boldsymbol{S}$ is the start subregion, $S_T \in \boldsymbol{S}$ is the end subregion, and $\delta : \boldsymbol{S} \times \boldsymbol{R} \to \boldsymbol{S}$ is the transition function that defines how one subregion transitions to another subregion by crossing one reference frame in $\boldsymbol{R}$. A string v is created as follows:*

- *v is initialized as an empty string ε.*
- *The path starts at $x_{init} \in S_0$ and ends at $x_{goal} \in S_T$.*
- *When there is a transition across a reference frame $r \in \boldsymbol{R}$, $v \leftarrow vr$.*

Thus, v is the string generated by a path through the map using $\boldsymbol{M}$. Because we restrict attention to finite length paths σ, the strings generated by those paths have a finite number of characters.

We now develop conditions under which a set of string blocks partition the set of all paths into a set of disjoint homotopy classes. We present these as a series of properties, lemmas, and theorems.

Let Σ denote the set of all (finite length) paths $\sigma \subset X_{free}$, and let Σ_v denote the set of all paths that generate string v. We call Σ_v a *string block* because it partitions Σ into blocks

that share common string properties. Notice that the lower case σ denotes a path and the upper case Σ denotes a set of paths; we use this convention throughout the paper.

We now present properties of paths and the strings generated by those paths via $\boldsymbol{M}$. Let v_i denote a string of a path that is generated by $\boldsymbol{M}$. Property 3 states that Σ is partitioned by a set disjoint string blocks. Note that the properties apply to both simple (non-looping) and non-simple (self-intersecting) paths. By Property 1, we can have Property 2.

Property 2. $\forall h_i,\ \boldsymbol{M}(\sigma^*_{h_i})$ *is a finite-length string.*

Property 2 says that we need only consider finite length strings even though there an uncountable number of paths through X_{free}. This means that the union in Property 3 is over a countable set in applications.

Property 3. $\Sigma = \bigcup_{v_i} \Sigma_{v_i}$ *and* $v_i \neq v_j \Rightarrow \Sigma_{v_i} \cap \Sigma_{v_j} = \emptyset$.

We use $\simeq$ to denote the homotopy of two paths, e.g. $\sigma_1 \simeq \sigma_2$. Property 4 states that two paths are homotopic when they belong to the same string block Σ_v. In other words, if σ_i and σ_j generate the same string v, σ_i and σ_j are homotopic.

Property 4. $\forall \sigma_i, \sigma_j \in \Sigma_v, \sigma_i \simeq \sigma_j$.

We extend the meaning of $\simeq$ to string blocks. Thus, $\Sigma_{v_i} \simeq \Sigma_{v_j}$ means that all the paths in string block Σ_{v_i} are homotopic to all the paths in string block Σ_{v_j}. Property 5 that $\simeq$ over string blocks is an equivalence relation.

Property 5. $\simeq$ *in* $\Sigma_{v_i} \simeq \Sigma_{v_j}$ *is an equivalence relation.*

Property 6 is related to Property 3 and tells us that the set of all paths in a single homotopy class h_i is partitioned into the string blocks associated with that homotopy class. Let $V_i = \{\boldsymbol{M}(\sigma) : \sigma \in h_i\}$, and let Σ_{V_i} denote the corresponding set of paths that generate strings in V_i.

Property 6. $\forall \Sigma_{h_i}, \exists V_i = \cup v_i^j, \Sigma_{h_i} = \Sigma_{V_i} = \cup \Sigma_{v_i^j}$.

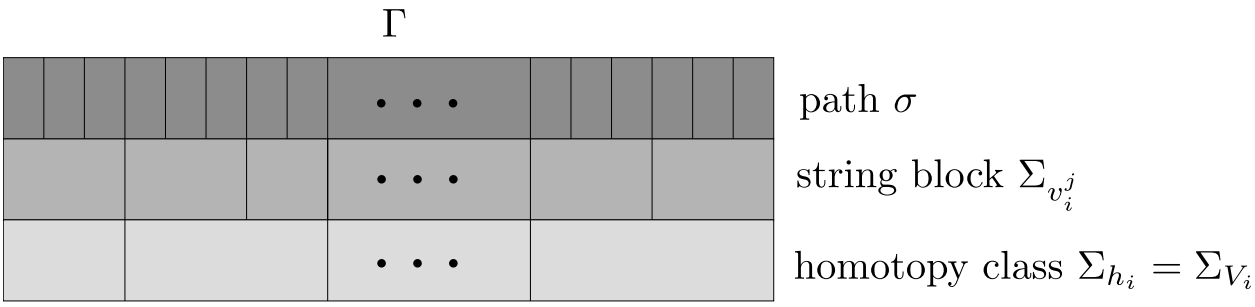


Figure 9.3: A hierarchy of path partitions.

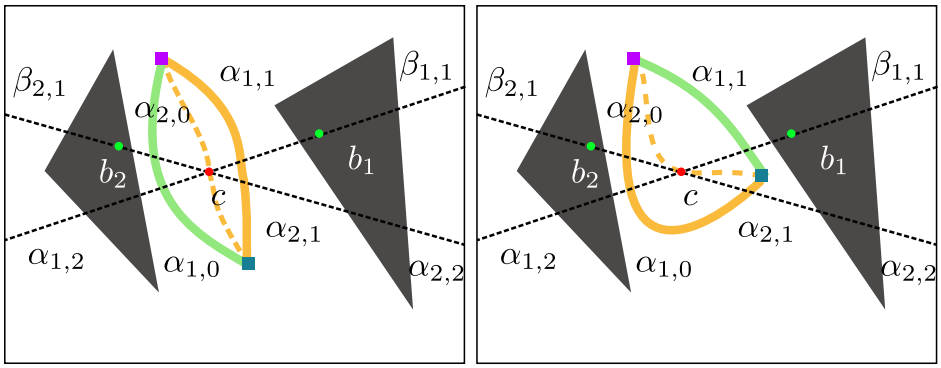
The set of all the paths can be partitioned into a set of homotopy classes and each homotopy class can be partitioned into a set of string blocks. This results in a hierarchical decomposition of paths, as illustrated in Figure 9.3.

Because of this partition, determining whether two paths in different string blocks are homotopic is equivalent to determining whether the two string blocks that the paths belong to are homotopic. We now establish some properties that allow us to turn this claim into an algorithm that determines whether two paths are homotopic by comparing the strings produced by those paths.

Let $\sigma_i \circ \sigma_j$ indicate the concatenation of σ_i and σ_j, and $v_i v_j$ indicate the concatenation of v_i and v_j. Property 7 states that if two touching paths σ_i and σ_j are concatenated together to form $\sigma_i \circ \sigma_j$ then $\boldsymbol{M}$ generates a string $v_i v_j$ that is the concatenation of the strings generated by the two individual paths v_i and v_j. We have Property 7.

Property 7. *If* $\sigma_i \in \Sigma_{v_i}, \sigma_j \in \Sigma_{v_j}$ *and* $\sigma_i(1) = \sigma_j(0)$ *then* $\sigma_i \circ \sigma_j \in \Sigma_{v_i v_j}$.

To determine the homotopic equivalence of two paths that belong to different string blocks, we remove an ambiguity from $\boldsymbol{M}$. Observe that the reference frames form a radial structure emanating from the center point c. We denote the set of all reference frames segments between the center point c and an obstacle boundary by $\boldsymbol{R}_c$ and the set of subregions that connect with the center point c by $\boldsymbol{S}_c$.



(a) Example A (b) Example B

Figure 9.4: Equivalence in Homotopy.

Property 8. *A path segment that sequentially crosses several reference frames in $\boldsymbol{R}_c$ between two different subregions in $\boldsymbol{S}_c$ is homotopic to a path segment that crosses only the center point c.*

For example, in Figure 9.4a, two paths with different strings $\alpha_{2,0}\alpha_{1,0}$ and $\alpha_{1,1}\alpha_{2,1}$ indicate two paths in the same homotopy class. By Property 8, we have $\Sigma_{\alpha_{2,0}\alpha_{1,0}} \simeq \Sigma_c \simeq \Sigma_{\alpha_{1,1}\alpha_{2,1}}$. Figure 9.4b illustrates a second example.

An important consequence of Property 8 is that paths that only go through regions $\boldsymbol{S}_c$ are homotopic to a simple path segment that starts and ends at the same position within $\boldsymbol{S}_c$. Furthermore, all of these paths are homotopic to a path that generates the empty string. This means that we can merge all the subregions in $\boldsymbol{S}_c$ into a new subregion $\widehat{S}_c$. We can now create a new DFAthat removes the ambiguity associated with the center region.

Definition 3. *Let $\boldsymbol{M}^h = (\boldsymbol{S}^h, \boldsymbol{R}^h, \delta^h, S_0^h, S_T^h)$ be a* homotopic DFA *that represents the string generation process from a path, where $\boldsymbol{S}^h = (\boldsymbol{S} \setminus \boldsymbol{S}_c) \cup \{\widehat{S}_c\}$ is a set of subregions, $\boldsymbol{R}^h = \boldsymbol{R} \setminus \boldsymbol{R}_c$ is a set of reference frames, $S_0^h \in \boldsymbol{S}^h$ is the start subregion, $S_T^h \in \boldsymbol{S}^h$ is the end subregion, and $\delta^h : \boldsymbol{S}^h \times \boldsymbol{R}^h \to \boldsymbol{S}^h$ is the transition function that defines how one subregion transitions to another subregion along the path by reference frames in $\boldsymbol{R}^h$. Strings are generated as follows:*

- *v is initialized as an empty string ε.*
- *The path starts at $x_{init} \in S_0^h$ and ends at $x_{goal} \in S_T^h$.*

- *When there is a transition across a reference frame* $r \in \boldsymbol{R}^h$, $v \leftarrow vr$.
- *When there is transition* $\boldsymbol{R}_c$, $v \leftarrow v\varepsilon = v$.

Figure 9.2c gives an example of the homotopic DFA that is created from the DFA in Figure 9.2b.

Now that we have removed this ambiguity, we observe an important relationship between a simple path and the string that $\boldsymbol{M}^h$ generates from this path. We separate this relationship into its own subsection to emphasize a shift from the properties of relevant partitions to useful algorithms.

9.3.2 Non-REP Strings

A duplicate ID character in a string reveals that a path crosses a reference frame more than once. Property 9 formalizes this.

Property 9. *A duplicate ID character in a string* $\boldsymbol{M}^h(\sigma)$ *indicates that* σ *has visited a subregion at least twice.*

Strings with no duplicate ID character can only be generated by simple paths that never leave a subregion by crossing a reference frame and then returning by recrossing that same reference frame. This implies the next property.

Property 10. *In every simple homotopy class there exists a path* σ *such that* $\boldsymbol{M}^h(\sigma)$ *has no duplicate characters.*

Consider a string constructed in the following manner: begin with the empty string ε and recursively insert a palindromic substring ww^R, where the R operator reverses the characters in the string, into any position of a string. We denote a string made up of recursively embedded palindromic substrings an *REP structure*. Note that ε and strings of the form ww^R are REP structures. A *non-REP string* is defined as a string that does not contain an REP structure or substring. Removing REP substrings of a REP string produces a non-REP string.

(a) Before deformation ε (b) After deformation qq

Figure 9.5: Path deformation.

We now present a lemma that yields the first useful algorithm of the paper.

Lemma 1. *If σ is a simple path segment that begins and ends in the same subregion and encloses no obstacle, then $\boldsymbol{M}^h(\sigma)$ is a REP string.*

Proof. The proof is by induction on the number of subregions visited by a path.

- **Base case:** If a simple path segment σ never leaves a subregion then $\boldsymbol{M}^h(\sigma) = \varepsilon$.
- **Induction step:** Assume a simple path segment σ that begins and ends in the same subregion, and $\boldsymbol{M}^h(\sigma)$ is a REP string. Deform the path segment σ into a different simple segment σ' by crossing only one more reference frame with ID q and then returning to the original subregion, as illustrated in Figure 9.5. $\boldsymbol{M}^h(\sigma')$ is $\boldsymbol{M}^h(\sigma)$ embedded with qq, where $qq = qq^R$ is a palindromic substring. Thus, $\boldsymbol{M}^h(\sigma')$ is also a REP string.
- **Conclusion:** Any simple path segment σ that begins and ends in the same subregion can be obtained by recursively applying deformation to a simple path that never leaves a subregion in the inductive step.

□

A useful consequence of this lemma is the following.

Corollary 1. *All simple paths that begin and end in the same subregion and enclose no obstacle are homotopic to each other and to a path σ such that $\boldsymbol{M}^h(\sigma) = \varepsilon$.*

Algorithm 2 removes REP substrings using the simple principle that if a character is on the top of the stack when you encounter the next one, you've found a REP substring and should eliminate it from the string. Algorithm 2 removes REP substrings by sequentially

Algorithm 2 REPTrim(v)

```
1: stack T = ∅
2: for char ∈ v do
3:     if Top(T) == char then
4:         Pop(T)
5:     else
6:         T ← char
   return T
```

pushing characters from the string onto a stack unless the character at the top of the stack matches the next character in the string. If the top of the stack and the next character match, the stack is popped to eliminate the palindromic structure.

Consider two paths in the same homotopy class, σ and σ^*. Suppose that σ^* is the shortest path in that homotopy class. REPTrim() converts string $v = \boldsymbol{M}^h(\sigma)$ into $v^* = \boldsymbol{M}^h(\sigma^*)$ yielding Property 11.

Property 11. $v^* = \text{REPTrim}(v)$.

The homotopic DFA converts two concatenated paths into two concatenated strings. Because of the recursive nature of Algorithm 2, these strings decompose into shortest strings yielding Property 12.

Property 12. $\text{REPTrim}(v_1 v_2) = \text{REPTrim}(v_1^* v_2^*)$
$= \text{REPTrim}(\text{REPTrim}(v_1)\text{REPTrim}(v_2))$.

Algorithm 2 will never have two consecutive characters the same in the stack. This implies that there are no two consecutive characters that are the same in the output v^*.

Property 13. $\forall i \in \{1, \ldots, |v^*| - 1\} v^*[i] \neq v^*[i+1]$.

Having characterized several relationships between a path σ and its corresponding string $\boldsymbol{M}^h(\sigma)$, we now want to identify string properties that tell us when two paths are homotopic. Although Property 4 tells us that two paths are homotopic when they are in the same string block, we need more. Specifically, we also need to know when two paths from different string blocks are homotopic.

9.3.3 Homotopic Equivalence

We now show that we can use a non-REP string v^* to determine whether $\boldsymbol{M}^h(\sigma_1)$ and $\boldsymbol{M}^h(\sigma_2)$ belong to the set of strings V_i generated by paths in the same partition. Lemma 2 shows that every path is homotopic to a path that generates a non-REP string.

Lemma 2. $\forall v, \exists v^*, \Sigma_v \simeq \Sigma_{v^*}$

Proof. Every path in a homotopy class is homotopic to the shortest path in that same homotopy class. By Property 10, the shortest path generates a non-REP string v^*, which means $\Sigma_v \simeq \Sigma_{v^*}$. □

Non-REP string v^* can be used to determine whether two paths are homotopic. This requires two lemmas. Lemma 3 applies when we have two identical shortest strings.

Lemma 3. $\text{REPTrim}(v_i^* v_j^{*R}) = \varepsilon \Leftrightarrow v_i^* = v_j^*$.

Proof. Suppose $\text{REPTrim}(v_i^* v_j^{*R}) = \varepsilon$. When $v_i^* v_j^{*R}$ is input to REPTrim(), v_i^* is pushed onto the stack first. The "no duplicate character" property of shortest strings, Property 13, means that the entire string v_i^* will be on the stack. Thus, the length of the stack is $|v_i^*|$. In order to get an empty string ε as the output, the stack needs to be cleared. This requires that $|v_i^*| = |v_j^{*R}|$ and $v_i^{*R} = v_j^{*R}$, which implies that $v_i^* = v_j^*$.

Conversely, when $v_i^* = v_j^*$, simulating stack pushes/pops shows that $\text{REPTrim}(v_i^* v_j^{*R}) = \text{REPTrim}(v_i^* v_i^{*R}) = \varepsilon$. □

Lemma 3 is a lemma about strings, and Lemma 4 is a corresponding lemma about paths.

Lemma 4. $\sigma_1 \simeq \sigma_2 \Leftrightarrow \mathrm{REPTrim}(\boldsymbol{M}^h(\sigma_1 \circ \sigma_2^R)) = \varepsilon$.

Proof. Suppose $\sigma_1 \simeq \sigma_2$. Concatenating σ_1 with a reversed σ_2 creates the path $\sigma_1 \circ \sigma_2^R$, which encloses no obstacle by the definition of homotopy. $\sigma_1 \circ \sigma_2^R$ is homotopic to the (closed) path that starts and ends at the same position, $\sigma_1(0)$. Since this closed path encloses no obstacle, $\sigma_1 \simeq \sigma_2 \Rightarrow \sigma_1 \circ \sigma_2^R \simeq \sigma_1(0)$. This means that the closed path $\sigma_1 \circ \sigma_2^R$ recursively visits several subregions and backtracks, returning to the starting point. Applying REPTrim() to the string generated by such a path yields the empty string. Thus, we have $\mathrm{REPTrim}(\boldsymbol{M}^h(\sigma_1 \circ \sigma_2^R)) = \varepsilon$.

Conversely, suppose $\mathrm{REPTrim}(\boldsymbol{M}^h(\sigma_1 \circ \sigma_2^R)) = \varepsilon$. We use a proof by contradiction. Assume that $\sigma_1 \not\simeq \sigma_2$. By Property 12, $\mathrm{REPTrim}(\boldsymbol{M}^h(\sigma_1 \circ \sigma_2^R)) = \mathrm{REPTrim}(v_1^* v_2^{*R})$. By Lemma 3, we have $v_1^* = v_2^*$. Because $\sigma_1 \not\simeq \sigma_2$, let $\sigma_1 \in \Sigma_1$ and $\sigma_2 \in \Sigma_2$, we have $\Sigma_1 \not\simeq \Sigma_2$. Let $\mathrm{min\_len}(\Sigma_i) = \arg\min_{\sigma \in \Sigma_i} |\sigma|$ be the shortest path in Σ_i. We have $\mathrm{min\_len}(\Sigma_1) \not\simeq \mathrm{min\_len}(\Sigma_2)$. However, by the definition of non-REP string, $\boldsymbol{M}^h(\mathrm{min\_len}(\Sigma_1)) = \boldsymbol{M}^h(\mathrm{min\_len}(\Sigma_2)) = v_1^* = v_2^*$. Property 4 implies that $\mathrm{min\_len}(\Sigma_1) \simeq \mathrm{min\_len}(\Sigma_2)$. This is a contradiction, so $\sigma_1 \simeq \sigma_2$. □

By Lemma 4 and Lemma 3, we can derive Theorem 1.

Theorem 1. $\mathrm{REPTrim}(\boldsymbol{M}^h(\sigma_1)) = \mathrm{REPTrim}(\boldsymbol{M}^h(\sigma_2))$ *iff* $\sigma_1 \simeq \sigma_2$.

Proof. Let $v_1^* = \mathrm{REPTrim}(\boldsymbol{M}^h(\sigma_1))$ and $v_2^* = \mathrm{REPTrim}(\boldsymbol{M}^h(\sigma_2))$.

When $\mathrm{REPTrim}(\boldsymbol{M}^h(\sigma_1)) = \mathrm{REPTrim}(\boldsymbol{M}^h(\sigma_2))$, we have $v_1^* = v_2^*$. By Lemma 3, we know $\mathrm{REPTrim}(v_1^* v_2^{*R}) = \varepsilon$. Thus, $\mathrm{REPTrim}(\boldsymbol{M}^h(\sigma_1)\boldsymbol{M}^h(\sigma_2)^R) = \varepsilon$. By Lemma 4, we have $\sigma_1 \simeq \sigma_2$.

When $\sigma_1 \simeq \sigma_2$, we can create a path $\sigma_1 {\sigma_2}^R$ by concatenating σ_1 and a reversed σ_2. Without loss of generality, we can assume that this path starts and ends at the same position. By Lemma 4 and Property 12, we have $\varepsilon = \mathrm{REPTrim}(\boldsymbol{M}^h(\sigma_1 {\sigma_2}^R)) = \mathrm{REPTrim}(v_i^* v_j^{*R})$. By Lemma 3, we know $v_i^* = v_j^*$, which implies $\mathrm{REPTrim}(\boldsymbol{M}^h(\sigma_1)) = \mathrm{REPTrim}(\boldsymbol{M}^h(\sigma_2))$. □

Theorem 1 tells us that we can identify the homotopy of two paths by comparing the strings generated by the homotopic DFA after passing them through Algorithm 2. We now

use this property to create RRT*-based planners that only generate solutions constrained to one or more homotopy classes.

We first present HARRT* in Section 9.4, which efficiently explores multiple string blocks. Although HARRT* does not allow optimal paths to be found subject to a homotopy class given a single string block, it is still useful from a human-robot interaction stand point because it enables an optimal path to be found if and when the human specifies a desired string block. If a human wanted to specify an entire homotopy class, we introduce TARRT* in Section 9.5.1, which enforces the sequences of subregions exploration for required string blocks subject to a technical condition on the length of the optimal path.

9.4 HARRT*: Bidirectional Homotopy-Aware Sampling

Before discussing the HARRT* algorithm, we give a brief discussion of how homotopy constraints can be generated. In the most simple case, suppose that we are given a desired homotopy class as an example path σ_{ex}. The REPTrim algorithm can then be used to generate the corresponding representative string $v^* = \text{REPTrim}(\boldsymbol{M}^h(\sigma_{ex}))$ for the homotopy class. Strings from other paths can then be compared to this string to see if the two strings match; the paths are in the same homotopy class if and only if the strings agree. In the more general case, multiple paths can represent multiple homotopy constraints, and a set of representative strings can be formed and used to determine if a path satisfies any of the homotopy constraints. HARRT* extends RRT* to include the string comparisons that eliminate branches of the tree that are not consistent with a desired homotopy class. More precisely, HARRT* takes as input a string block, presumably specified by a human, and finds optimal paths within this string block (subject to a technical condition explained shortly). Recall that a string block is a subclass of a homotopic class, so finding an optimal path subject to a string block is an important contribution to homotopy-based path-planning.

RRT* explores a map to generate an optimal tree structure based on the cost distribution on the map. While the tree structure explores the planning space, the DFA $\boldsymbol{M}^h$ can be

used to generate the strings of the branches. Branches of the tree terminate if they generate a string that differ from string block constraint specified by the human. The string of each branch indicates the homotopic subclass of the corresponding subpath given by the string block. The resulting algorithm, Algorithm 3, is called Homotopy-aware RRT* (HARRT*).

Algorithm 3 HARRT* (x_{init}, x_{goal})

$i \leftarrow 0$
$N_s \leftarrow \{x_{init}\}$; $E_s \leftarrow \emptyset$; $T_s \leftarrow (N_s, E_s)$
$N_g \leftarrow \{x_{goal}\}$; $E_g \leftarrow \emptyset$; $T_g \leftarrow (N_g, E_g)$
while $i < N$ **do**
    $T_s, x_s^{new} \leftarrow$ EXPLORE(T_s, i)
    $T_g, x_g^{new} \leftarrow$ EXPLORE(T_g, i)
    $p_s \leftarrow$ CONNECT(x_s^{new}, T_g)
    $p_g \leftarrow$ CONNECT(x_g^{new}, T_s)
    $P \leftarrow$ UPDATEBESTPATHBYCLASS(p_s, P)
    $P \leftarrow$ UPDATEBESTPATHBYCLASS(p_g, P)
    $i \leftarrow i + 1$
$P \leftarrow$ MERGEPATHS(P) **return** P

The algorithm uses a bi-directional structure. There is a *start tree* $T_s = (N_s, E_s)$, which is an RRT* structure from the start position for the optimal cost-to-arrive. N_s is the set of vertices in T_s, and E_s is the set of edges in T_s. Similarly, there is a *goal tree* $T_g = (N_g, E_g)$, which is an RRT* structure from the goal position for the optimal cost-to-go.

In each iteration, a new vertex is created and added to each tree using EXPLORE(). CONNECT() is then called to create a path with a vertex in the other tree. In order to guarantee optimality, a set of near vertices in T_g is provided to find the best vertex to be connected with the new vertex x_s^{new} in T_s, and vice versa. The created path will be compared with the current best path that belongs to the same string block. If it is a better one, the best path in this string block will be updated, which is implemented in UPDATEBESTPATHBYCLASS().

Algorithm 4 gives the exploration process of a tree structure and is similar with that used in RRT* [59]. The first difference is that the string associated with each branch is updated (implementing $\boldsymbol{M}^h$ on that branch). The second difference is that the STRINGCHECK()

Algorithm 4 EXPLORE(T, i)

$x_{rand} \leftarrow$ SAMPLE(i) ;
$x_{nearest} \leftarrow$ NEAREST(T, x_{rand})
$x_{new} \leftarrow$ STEER$(x_{nearest}, x_{rand}, \eta)$
if OBSTACLEFREE$(x_{nearest}, x_{new})$ **then**
    $s \leftarrow \mathrm{STR}(x_{nearest}) \circ \mathrm{CRF}((x_{nearest}, x_{new}))$
    if STRINGCHECK(s) **then**
        $x_{min} \leftarrow x_{nearest}$
        $X_{near} \leftarrow$ NEAR$(T, x_{new}, |N|)$
        for each $x_{near} \in X_{near}$ **do**
            if OBSTACLEFREE(x_{new}, x_{near}) **then**
                $s \leftarrow \mathrm{STR}(x_{near}) \circ \mathrm{CRF}((x_{near}, x_{new}))$
                if STRINGCHECK(s) **then**
                    if COST(x_{near}) $+c($ LINE(x_{near}, x_{new}) $) <$ COST(x_{new}) **then**
                        $x_{min} \leftarrow x_{near}$
        $E' \leftarrow E' \cup \{(x_{min}, x_{new})\}$
        for each $x_{near} \in X_{near} \setminus \{x_{min}\}$ **do**
            if OBSTACLEFREE(x_{new}, x_{near}) **then**
                $s \leftarrow \mathrm{STR}(x_{new}) \circ \mathrm{CRF}((x_{new}, x_{near}))$
                if STRINGCHECK(s) **then**
                    if COST$(x_{near}) >$ COST(x_{new}) + c(LINE(x_{new}, x_{near})) **then**
                        $x_{parent} \leftarrow$ PARENT(x_{near})
                        $E' \leftarrow E' \setminus \{(x_{parent}, x_{near})\}$
                        $E' \leftarrow E' \cup \{(x_{new}, x_{near})\}$
return T, x_{new}

method is used to check whether the string of a branch satisfies the string constraint specified by the given string block constraint. Strings that satsify the string constraint correspond to a branch of the tree that is in an allowed string block. The methods in Algorithm 4 are defined as follows:

- CRF(l): Return the ID characters that represent the crossed reference frames of a line segment l if any.
- STR(x): Return the string that represents the crossed reference frames of the subpath from the root to the node x sequentially. This implements $\boldsymbol{M}^h$.

Because RRT* maintains a tree structure, each vertex has only one path to arrive from the root. This path can be converted into a string of ID characters by $\boldsymbol{M}^h$. StringCheck() guarantees that a new node is added or rewired so that all the branches of the tree structure are in the set of representative strings that correspond to the set of permissible string blocks. We are emphasizing that HARRT* checks to see if the path is within a string block rather than checking if the path is in a permissible homotopy class (recall from the hierarchy that there can be multiple string blocks in each homotopy class). The TARRT* algorithm discussed later enforces a more precise version of this homotopic, allowing for exploration of branches that might have REP substrings.

For example, suppose we have a string constraint "ab". A branch of the start tree T_s with string "a" satisfies the constraint, because "a" can be extended into "ab" by concatenating a "b". However, a branch beginning with string "b" cannot be extended into "ab", and therefore does not satisfy the string constraint. It is similar for the goal tree T_g but with reversed string order. Note that this is an early check of the homotopy class constraint and may eliminate some paths that would explore areas outside of the current subregion; we will say more about this in the results section.

The methods in Algorithm 5 are defined as follows:

- Path(v, T): Return the path from the root of the tree T to the vertex v.

Algorithm 5 CONNECT(x_{new}, T)

```
1: p_min = ∅
2: X_near ← NEAR(T, x_new, |N|)
3: for each x_near ∈ X_near do
4:     if ObstacleFree(x_new, x_near) then
5:         if x_new ∈ T_s then
6:             p ← Concatenate(x_new, x_near)
7:         else
8:             p ← Concatenate(x_near, x_new)
9:         if StringCheck(p) and Cost(p) < Cost(p_min) then
10:            p_min = p
   return p_min
```

- CONCATENATE(p_a, p_b): Return a concatenated path of p_a and p_b. If p_a and p_b are from different directions, one of them will be reversed for the concatenation.

When the exploration process is finished, the algorithm returns a set of the best paths corresponding to legal paths for the different permissible string blocks. Once the algorithm completes, it is possible to use the REPTrim algorithm to cluster paths from different string blocks into the same homotopy class. This means that a human can specify one or more string blocks within one or more given homotopy classes that might be of interest to the human, and then let the algorithm find the best path for each homotopy class generated by exploring the relevant string blocks. We assume that this is the human's intent, so the MERGEPATHS() method clusters the equivalent string blocks into their relevant homotopy classes, and the set of paths P is updated.

9.4.1 Theoretical Analysis

In the CONNECT() procedure, a vertex n in either T_s or T_g is used to create a path by connecting with a vertex in the other tree. The resulting path connects a subpath from the start to n to a subpath from n to the goal, which corresponds to finding a path from the start to the goal with a constraint of via n.

Property 14 states the convergence properties of the algorithm. It implicitly relies on Assumptions 1–3 from Karaman's RRT* paper.

Property 14. *Given Assumptions 1-3 from Karaman's RRT* paper [59], The cost of the path from the root to any vertex in the tree produced by RRT* almost surely converges to the optimal cost from root to vertex within a permissible string block.*

The start tree and the goal tree asymptotically converge to the optimal structures, which provide the optimal-to-come subpath and optimal-to-go subpath connected by a via-point. Concatenating the optimal subpaths derives the optimal path of the *via-point constraint*.

Lemma 5. *Given Assumptions 1-3 from Karaman's RRT* paper [59], the path created by concatenating a path from T_s via n to a path from T_g almost surely converges to the optimal path subject to the the constraint that the path stays within a permissible string block.*

There are two big limitations of the HARRT* algorithm. First, the algorithm only explores paths that are consistent with a given string block rather than all paths that are within a given homoptopy constraint. (Recall from the hierarchical decomposition of paths and strings that the strings associated with a given homotopy constraint can include multiple string blocks.) Second, not all interesting and useful paths in a string block can be achieved using the via-point method of connecting start and goal trees in HARRT*. For example, twisted or winding topologies may not be achievable using via-point connections. Consequently, HARRT* does not provide completeness in exploring all possible homotopy classes though it does enable useful exploration of string blocks that can later be clustered into relevant homotopy classes.

We now present the TARRT* algorithm, which uses a sequence-guided homotopy-aware sampling approach, that can explore all possible homotopy classes.

9.5 TARRT*: Sequence-Guided Homotopy-Aware Sampling

In this section, we present the TARRT* algorithm, a topology-aware random sampling algorithm that is derived from the standard RRT* algorithm [60]. Assumptions from [60]

about RRT* are inherited here. Since we are interested in path-planning subject to topological constraints, we restrict attention to paths that (a) start at a specified initial position and end at a specified goal position and (b) satisfy a topological constraint specified as a given set of homotopy classes. As an example of a topological constraint, suppose that the topological constraint is specified when a human draws a path on a map and specifies that the robot must find the optimal path from the homotopy class of the drawn path; then Σ is a single homotopy class. Similarly, if a human says to avoid a particular region, then this topological constraint corresponds to a set of different homotopy classes that avoid that region. Note that unlike HARRT*, which restricts exploration to string blocks, TARRT* can explore entire homotopy classes. We are interested in both non-simple paths and non-simple homotopy classes.

We had previously restricted attention to paths that were finite length, but we now need to place a stronger restriction on the types of paths that the algorithm will explore.

Assumption 1. $\forall \sigma^* = \arg\min_{\sigma \in \Sigma} \textsc{Cost}(\sigma), \exists \tau \geq 1, |v| \leq \tau |v^*|$ *and* $\sigma^* \in \Sigma_v$.

In this assumption, the topological constraint is represented by the set Σ that contains all homotopy classes consistent with this constraint. Assumption 1 says that if you give me an optimal path σ^* that produces string v^* then we can restrict search to paths σ and their corresponding strings v such that (a) the optimal path belongs to homotopy class induced by v and (b) the length of the strings is constrained to be less than a "stretching" factor times the length of of v^*. Stated simply, we assume that we can restrict search to paths that are in a homotopy class consistent with some optimal path and that don't produce really long strings. The value of τ determines how many string blocks need to be explored to find the optimal path in a homotopy classes. For example, when $\tau = 1$, only the shortest string block is needed to explore for finding the optimal path of a homotopy class.

This restriction addresses one of the limitations of the HARRT* algorithm by (a) allowing the algorithm to explore paths that may be consistent within the homotopy class once

the REPTrim algorithm is applied rather than (b) restricting exploration to only one string block with a homotopy class.

9.5.1 Expanding Topology

The basic idea of the Topology Aware RRT* (TARRT*) algorithm is that the tree produced by RRT* is "chunked" into subtrees when the paths in those subtrees produce different string blocks; extension of the trees is restricted to only those string blocks that are consistent with the desired topological constraint(s). The RRT* subtrees find the optimal path within each string block, and the optimal path for the homotopy class is the best of the paths for the string blocks. The algorithm can be naturally extended when the topological constraint consists of several homotopy classes but with a corresponding increase in the number of string blocks to explore.

We will be talking about two different trees as we discuss the algorithm: the tree produced by RRT* and the tree of subtrees produced when we group the RRT* branches into string blocks. To help distinguish between the elements of the tree of subtrees and the elements of the tree, we will refer to the subtrees that belong to the same string block as a *TARRT* node* and the individual elements of the complete tree as *RRT* vertices*.

Figure 9.6a illustrates a very simple world with four regions. These four regions are separated by four reference frames, "A1,B1,A2,B2", and the homtopic DFA adds the label for these references frames to the string whenever the path crosses the reference frame. The small red square indicates the start position and the small blue square indicates the goal position; for this world, the path must start in subregion "R1" and end in subregion "R4". Figure 9.6b illustrates some of the sequences of regions that could be visited on a path from the red square in "R1" to the blue square in "R4". Thus, they represent different string blocks and different sequences of TARRT* nodes that can be created, since each visit to a new region causes a new character to be added to the string. The parameter τ represents the "budget" given to the algorithm to explore different possible string blocks. In the figure, τ is

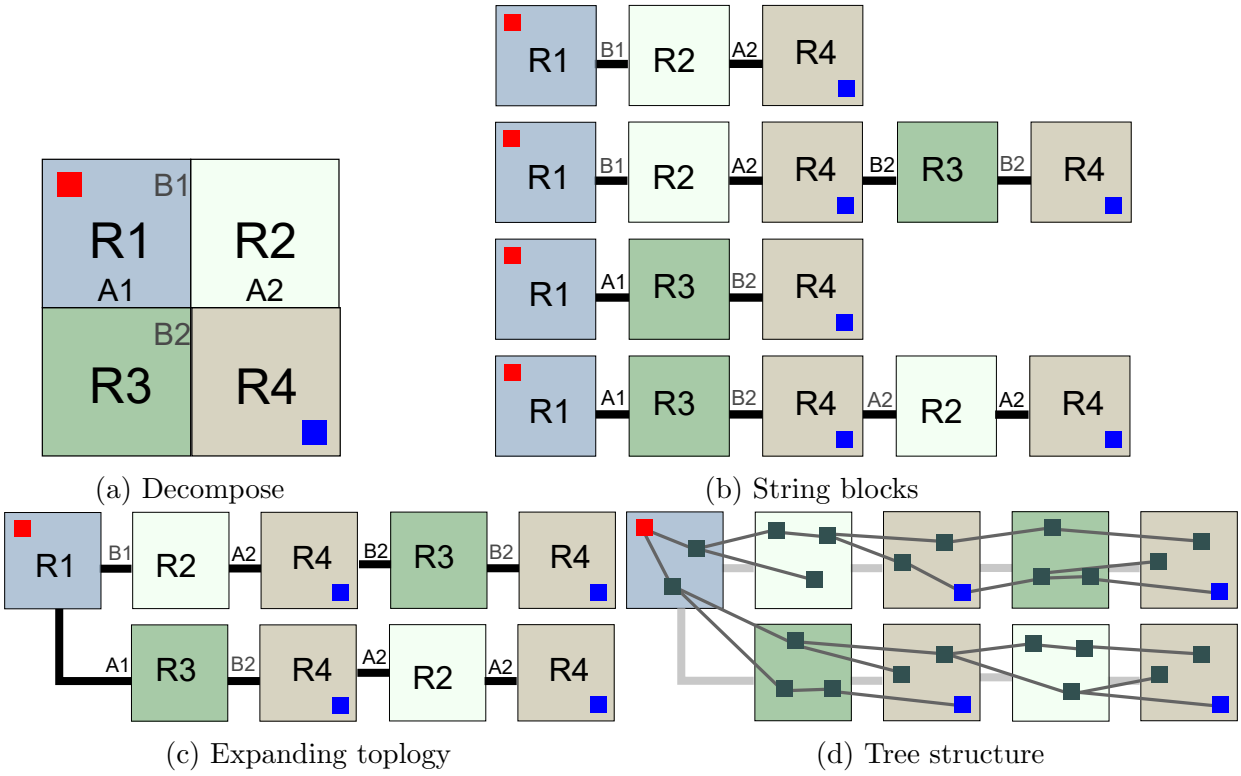


Figure 9.6: Expanding Toplogy.

large enough to allow strings up to four characters long, corresponding to paths that visit up to five TARRT* nodes.

Each TARRT* node is associated with a subregion, and each edge in TARRT* is associated with a reference frame. There exist similarities between string blocks. For example, all the string blocks start in the same initial subregion "R1". The string block "B1,A2,B2,B2" contains a substructure that is identical with the string block "B1,A2". We can thus use an expanding topology to efficiently express these string blocks, as shown in Figure 9.6c. The *root TARRT* node* is always associated with the start subregion. Denote a TARRT* node associated with the goal subregion as a *terminal TARRT* node*. Any path from the root expanding node to a terminal TARRT* node defines a string block, which is called a *string-block branch* of the TARRT* tree. In Figure 9.6c, each path from the TARRT* node "R1" to an TARRT* node "R4" is within one of the string blocks in Figure 9.6b.

RRT* uses directed random sampling to create new possible nodes in the RRT* subtrees. Since each one of the new possible RRT* vertices is located in a subregion, it is

possible that the location of the new node can be part of multiple string blocks and their corresponding TARRT* nodes. If we can generate an optimal structure like RRT* but sorted by string blocks, backtracking from a goal position in a terminal expanding node to the root obtains the optimal path of the corresponding string block.

9.5.2 Topology-Aware Space Sampling

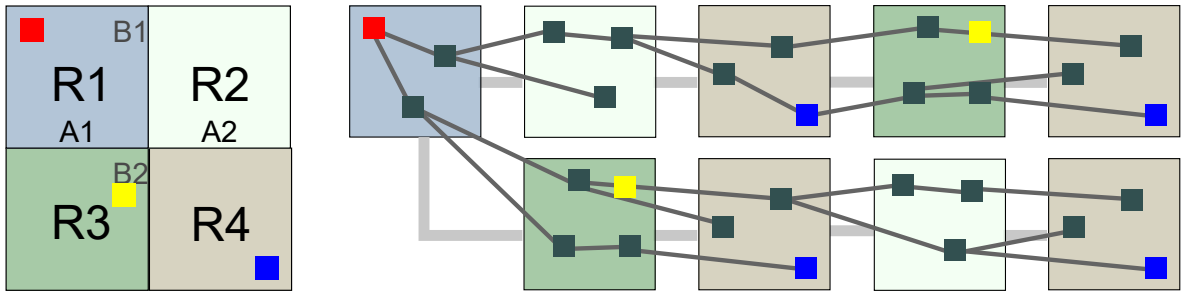
The TARRT* algorithm is given as Algorithm 6. It inherits optimal spatial sampling from RRT* but the tree generation process is guided by an expanding topology of TARRT* nodes. The branches of the tree are sorted by string-block branches of the expanding topology, like in Figure 9.6d. The algorithm enforces a "tree of subtrees" structure by ensuring that the parent RRT* vertex of any RRT* vertex can only be located (a) within the same TARRT* node as the RRT* vertex or (b) in the parent node of that TARRT* node. Moreover, if an existing RRT* vertex is linked with a new RRT* vertex, the edge between those vertices must visit reference frames as defined in the expanding topology of TARRT* nodes.

For example, consider the string block "B1, A2" at the top of Figure 9.6b and an RRT* vertex in the TARRT* node "R4". If the RRT* vertex has a parent in TARRT* node "R2", the edge between child and parent should cross the reference frame "A2". If the RRT* vertex has an edge connecting to a grandparent RRT* vertex in TARRT* node "R1", the edge should cross the reference frames "B1","A2" sequentially. If an RRT* vertex has an edge towards a node in the TARRT* node "R3", the edge violates the requirement of this string block and should not exist.

Because a subregion is associated with multiple TARRT* nodes in an expanding topology, when a new position is sampled as dictated by the directed sampling in RRT*, a new RRT* vertex will be created in each TARRT* node that is associated with the subregion that the new position is in. This means that when a new position is sampled, there are new RRT* vertices created in several associated TARRT* nodes. For example, a new position (the yellow square) is sampled in the subregion "R3" of a map, as illustrated in Figure 9.7a.

Algorithm 6 Topology-Aware Rapidly-exploring Random Tree* $G(V, E)$

$V \leftarrow \{x_{init}\}$; $E \leftarrow \emptyset$; $i \leftarrow 0$
while $i < N$ **do**
  $x_{rand} \leftarrow$ SAMPLE (i) ; $i \leftarrow i + 1$
  $x_{nrst} \leftarrow$ NEAREST (G, x_{rand})
  $x_{new} \leftarrow$ STEER $(x_{nrst}, x_{rand}, \eta)$
  if OBSTACLEFREE(x_{nrst}, x_{new}) **then**
    $S =$ SUBREGION (x_{new})
    for each *tarrt_node* in TARRTNODES (S) **do**
      tarrt_node $\leftarrow x_{new}$
      $G \leftarrow$ EXTEND (G, x_{new}, x_{nrst})



(a) Sampling (b) Create new nodes

Figure 9.7: Sampling and adding new nodes.

Two new RRT* vertices of the position are added to the two TARRT* nodes, one for the top string block topology and another for the bottom string block topology as shown in Figure 9.7b.

We now define several functions, using appropriately modified definitions from the RRT* algorithm in [60].

- SAMPLE(): Returns independent uniformly distributed samples from X_{free}.
- NEAREST(): Returns a position of the vertex whose position is closest to point x. NEAREST$(G = (V, E), x) = \arg\min_{v \in V} \|x - v\|$.
- STEER(): Given two points x and y, returns a point z on the line segment from x to y that that is no greater than η from y. STEER$(\ x, y, \eta\ ) = \arg\min_{z \in \boldsymbol{R}^d, \|z-x\| \leq \eta} \|z - y\|$.
- OBSTACLEFREE(x, x'): Returns True if $[x, x'] \subset X_{free}$, which is the line segment between x and x' lies in X_{free}.

- SUBEGION(x): Returns the subregion that position x is in.
- TARRTNODES(S): Returns all TARRT* nodes from the expanding topology that are associated with subregion S.

The RRT* vertices of the TARRT* tree are created and stored in TARRT* nodes. This provides information for how to add connections between new positions to potential parent RRT* vertices and also how to rewire RRT* vertices so that rewiring honors string block constraints. Thus, the EXTEND procedure of TARRT*, see Algorithm 7, is slightly different from the corresponding method of RRT*.

Algorithm 7 EXTEND($G, x_{new}, x_{nearest}$)

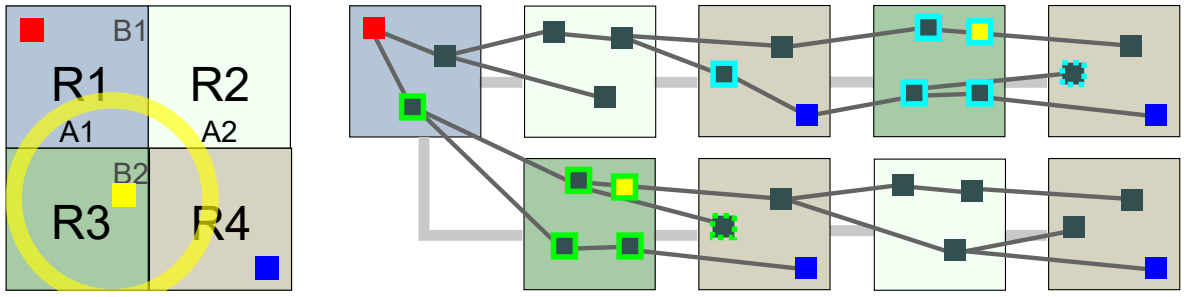
if $x_{new} = x_{nrst}$ **then return** $G = (V, E)$
$V' \leftarrow V \cup \{x_{new}\}$
$x_{min} \leftarrow x_{nrst}$
$X_{near} \leftarrow \text{NEAR}(G, x_{new}, |v|)$
for each $x_{near} \in X_{near}$ **do**
    if OBSTACLEFREE(x_{new}, x_{near}) **and** HOMOTOPYELIGIBLE(x_{new}, x_{near}) **then**
        $c'_k \leftarrow \text{COST}_k(x_{near}) + c_k(\text{ LINE}(x_{near}, x_{new})\ )$
        if $c'_k < \text{COST}_k(x_{new})$ **then**
            $x_{min} \leftarrow x_{near}$
$E' \leftarrow E' \cup \{(x_{min}, x_{new})\}$
for each $x_{near} \in X_{near} \setminus \{x_{min}\}$ **do**
    if OBSTACLEFREE(x_{near}, x_{new}) **and** HOMOTOPYELIGIBLE(x_{near}, x_{new}) **then**
        $c'_k \leftarrow \text{COST}_k(x_{new}) + c_k(\text{ LINE}(x_{new}, x_{near})\ )$
        if $c'_k < \text{COST}_k(x_{near})$ **then**
            $x_{parent} \leftarrow \text{PARENT}(x_{near})$
            $E' \leftarrow E' \setminus \{(x_{parent}, x_{near})\}$
            $E' \leftarrow E' \cup \{(x_{new}, x_{near})\}$
return $G' = (V', E')$

The precise definitions of the methods used in the Algorithm 7 are given below.

- NEAR($G, x, card$): Returns all vertices within the closed ball of radius

$$\gamma = \min\{\gamma_{RRT*}(\log(card)/card)^{1/d}, \eta\}$$

centered at x, in which $\gamma > (2(1 + 1/d))^{1/d}(\frac{\mu(X_{\text{free}})}{\zeta_d})^{1/d}$ [60].



(a) Radius (b) Near nodes

Figure 9.8: Radius and near nodes.

- HomotopyEligible(x_{from}, x_{to}): Return true if the sequence of reference frames a line visits is consistent with a required sequence of reference framse. The line is from x_{from} to x_{to}. The required sequence of reference frames is obtained by the sequence of edges from the TARRT* node that x_{from} is in to the TARRT* node that x_{to} is in.
- Line$(x, x') : [0, s] \leftarrow X_{free}$ denotes the path defined by line segment from x to x'.
- Cost(x): Returns cost of the unique path (because G is a tree) from x_{init} to the vertex $x \in V$. Cost$(x_{\text{init}}) = 0$.

The definition of HomotopyEligible() in Algorithm 7 differentiates TARRT* from conventional RRT*. It indicates n path constraint, in which the edge between the RRT* vertex and a possible RRT* vertex must not intersect a reference frame or they must intersect a reference frame in such a way that the path through the tree stays within the string block. Recall that there are multiple string blocks permitted by the homotopy constraint, so the entire homotopy constraint is explored (in constrast to HARRT*), though subject to the path length assumption made above.

For example, given a radius for the sampled position shown in yellow in Figure 9.8a, different string-block branches return different sets of possible RRT* vertex neighbors. In Figure 9.8b, they are shown with different colored borders. Figure 9.8b also illustrates the eligible nodes returned from HomotopyEligible() in solid borders in the process of wiring the new node to the tree. The illegible nodes are in dotted borders. The resulting tree

structure of TARRT* allows us to trace the path from goal to starting location while staying within a specified string block.

9.5.3 Acceptable String Blocks

Section 9.5 only uses two results directly from Section 9.3: the process for partitioning the world and the homotopic DFA. The removal of REP strings did not directly contribute to the TARRT* algorithm, but the properties of the REPTrim algorithm are very useful in improving TARRT efficiency.

Consider that the set of possible paths through different subregions of the world is infinite. We use the stretch parameter τ to constrain the length of the strings and, consequently, the set of string blocks that we can explore. For large values of τ, there are still a huge number of possible string blocks.

Fortunately, we can prune many of the potential string blocks by generating the string v for each potential string block less than the bound $\tau|v^*|$ by a combinatorial *a priori* search through the regions. We then pass the resulting string v into the REPTrim algorithm. Let REPTrim(v, τ) denote the resulting set of possible string blocks. Similarly, for each homotopic class in the topological constraint Σ, we find the minimum string for any path in the homotopy class using REPTrim. Let REPTrim(Σ) denote the resulting set of minimum length strings that satisfy the topological constraint. Only string blocks in REPTrim(v, τ) $\cap$ REPTrim(Σ) need to be considered in TARRT*.

9.5.4 Theoretical Analysis

We define a simple string-block branch as a branch in the expanding topology that contains no duplicate subregion and use this definition to prove optimality of paths in a simple string-block branch.

Lemma 6. *The subtree of a simple string-block branch is asymptotically optimal subject to the path-stretching constraint.*

Proof. By Theorem 38 in [60], when the radius γ is larger than a threshold

$$(2(1+1/d))^{1/d}(\frac{\mu(X_{\text{free}})}{\zeta_d})^{1/d},$$

RRT* structure is asymptotically optimal. A simple string-block branch implies that the sampling space is only a subset of the X_{free}. The function HomotopyEligible() trims all of the edges that lead to different string-block branches. So the subtree of a simple string-block branch is asymptotically optimal. □

We can then use the optimality of paths in simple string-block branches to prove asymptotic optimality for all string-block branches in TARRT*.

Theorem 2. *The solution of a string-block branch in TARRT* converges to the optimal path of a corresponding string block almost surely subject to the path-stretching constraint.*

Proof. Any string-block branch, including a winding one, can be represented as a concatenation of several simple string-block branches. Decompose a string-block branch into several sections, each of which is a simple string. Let R_i be the last TARRT* node of section v^j . There are a set of RRT* vertices that are connected with RRT* vertices in successor TARRT* nodes. The successor TARRT* nodes are all in the successor section v^{j+1}. Any member of the RRT* vertex set is a root to a tree, which extends into section v^{j+1}, which is equivalent to a simple string-block branch. By these vertices, subtrees of two sections can be connected.

By Lemma 6, such a tree is asymptotically optimal. The tree of the string-block branch can be assembled by connecting from the subtree of the last section to the subtree of the first section. Because the subtree of each section is asymptotically optimal, the tree of the string-block branch is asymptotically optimal. Thus the solution converges to the optimal almost surely. □

By Theorem 1, we establish the optimal path of a homotopy class by TARRT*. A homotopy class can contain an infinite number of string blocks by Property 3. It is impossible

to enumerate all the possible string blocks of one homotopy class in order to find the optimal path of the class. The optimal path of a homotopy class may belong to the corresponding shortest string block, but sometimes visiting extra subregions and then returning back to the parent subregion can lead to a path that has lower cost. Under Assumption 1, TARRT* can explore a reasonable number of string blocks for a homotopy class. We have Corollary 2.

Corollary 2. *With a τ that satisfies Assumption 1, TARRT* can find the optimal path of a homotopy class almost surely.*

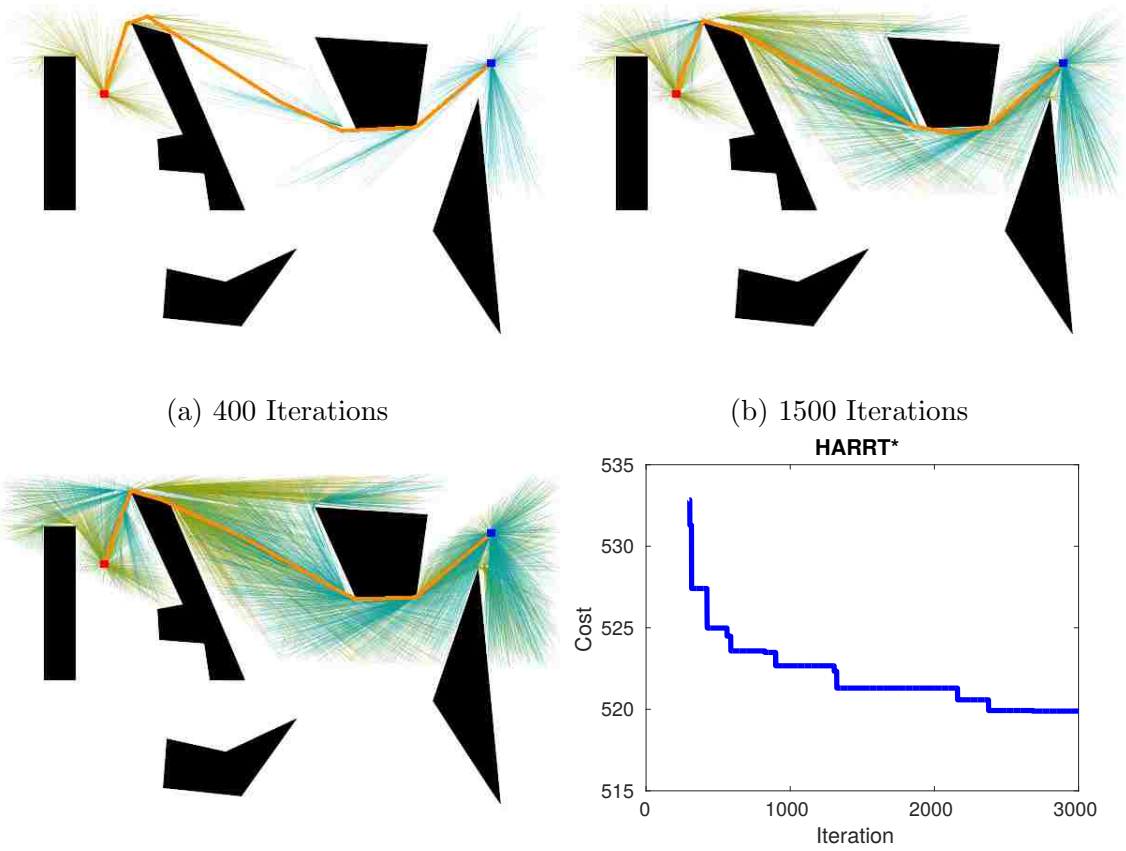
Proof. Under Assumption 1, there is a finite number of string blocks to explore in a homotopy class, which will contains the optimal path of the homotopy class under that assumption. The lengths of these string blocks are all less than or equal to $\tau|v^*|$, in which v^* is the shortest string of the string blocks in the homotopy class. By Theorem 2, we know that the optimal path found for each string block converges to optimal almost surely. Thus the set of optimal paths found (one for each member of the set of string blocks) converges to optimal almost surely. This means that the optimal path for each homotopy class will be found almost surely. □

9.6 Experiments

This section presents a series of illustrative examples that illustrate how HARRT* and TARRT* work. These examples provide empirical support and for the claims in the paper; they also illustrate some useful properties of the algorithms. Both HARRT* and TARRT* depend on a homotopic DFA. The differences are compared in Table 9.1. Comparing the asymptotical optimality of HARRT* and TARRT* reveals how TARRT* supports completeness in homotopy classes and how it works in winding homotopy classes.

Table 9.1: Comparison of HARRT* and TARRT*

Algorithm	**Type**	**Path space coverage**	**Homotopy-class exploration**
HARRT*	Bidirectional RRT*	String blocks	incomplete
TARRT*	Block-sequence RRT*	Simple and non-simple paths	complete



(a) 400 Iterations

(b) 1500 Iterations

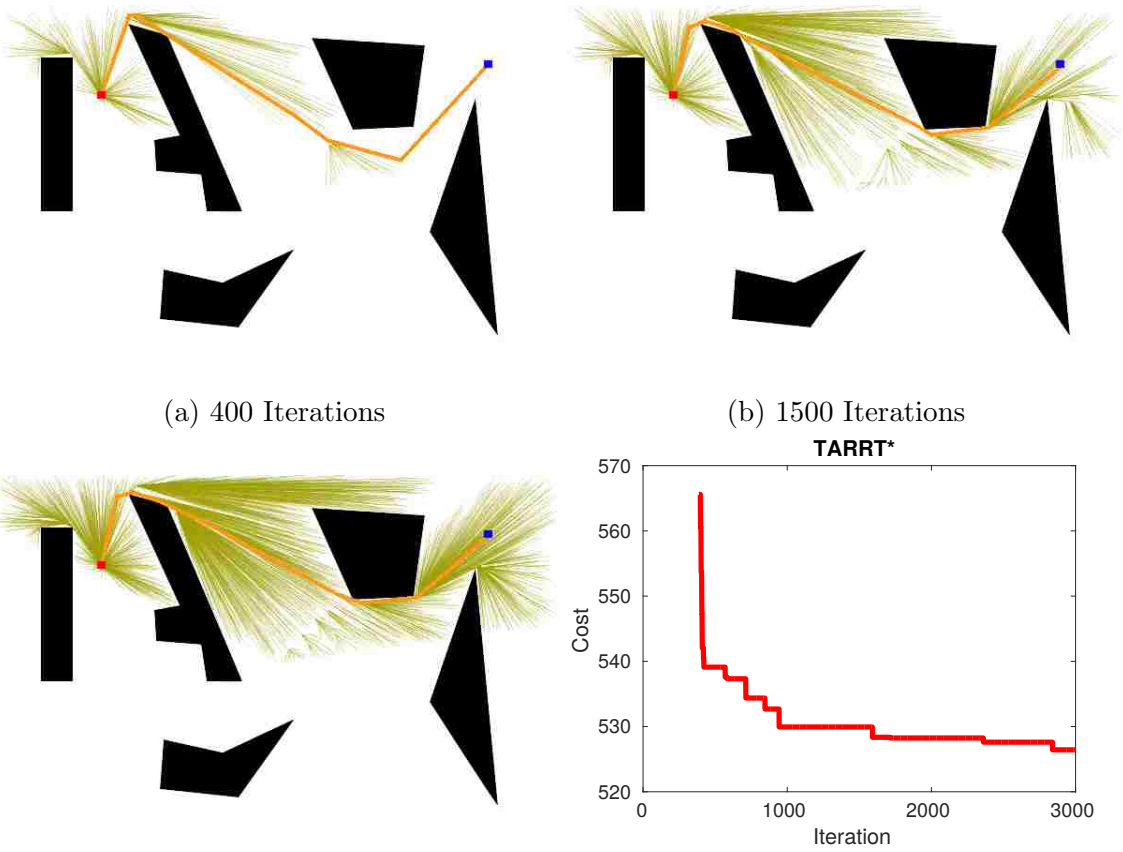
(c) 3000 Iterations

(d) Convergence

Figure 9.9: Optimal search of HARRT*

9.6.1 Optimality

Figure 9.9 illustrates an optimal search of HARRT* for a string block where the path is required to go above the second left obstacle at the top of the world and then below the third left obstacle at the top. Black blobs indicate obstacles. The orange line is a found path from the start (red point) to the goal (blue point). The olive lines visualize the start tree structure and the dark turquoise lines visualize the goal tree structure generated by HARRT*. The objective is chosen as minimizing the path length.



(a) 400 Iterations

(b) 1500 Iterations

(c) 3000 Iterations

(d) Convergence

Figure 9.10: Optimal search of TARRT*

Figure 9.9a to 9.9b show how the bidirectional tree extends and how the best path of the homotopy class is refined. Figure 9.9d shows the convergence of found best path in the homotopy class. You can see the string block restriction in the figures by noting how the exploration trees do not cross an approximately horizontal boundary near the center of the figure. This approximately horizontal boundary corresponds to a reference frame used by the homotopic DFA, and crossing that boundary would cause the string produced by the path to violate the string block constraint.

Figure 9.10 shows the results of applying the TARRT* algorithm on the same test case. It also converges to the same optimal path but in a slower convergence rate, because the bidirectional exploration of HARRT* speeds up the process.

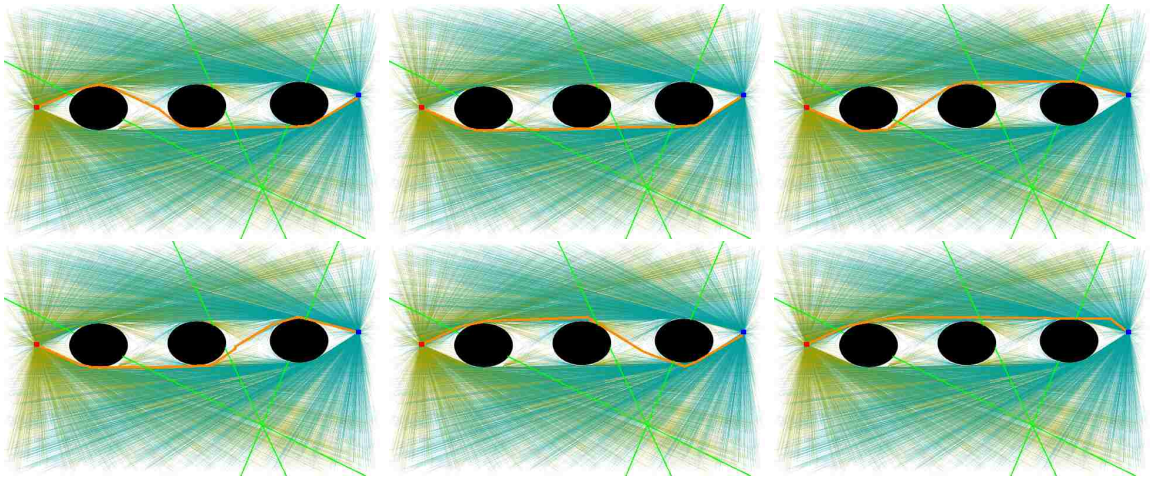


Figure 9.11: HARRT*.

9.6.2 Completeness

Completeness of a topology-based path-planning algorithm is defined to mean that the optimal path can be found for any homotopy class, given the stretching constraint imposed by Assumption 1. The world in Figure 9.11 was created to explore different types of "slalom"-like homotopy constraints through the world. We considered six of the possible homotopy classes and used minimum distance as the objective. The HARRT* algorithm uses a bidirectional structure to explore homotopy classes and it cannot find a successful path for all slalom-like homotopy constraints. Figure 9.11 shows six worlds in which HARRT* succeeded, but note that the algorithm failed in the bottom two constraints shown in Figure 9.12. The reason that HARRT* fails in those two cases is not because it restricts search to a string block; it is easy to specify a string block that slaloms in and out of the obstacles. Rather, it fails because the via-point constraint between the start tree and the goal tree is not satisfied.

By contrast, TARRT* uses a direct exploration of the homotopy class, which means that TARRT* can explore any homotopy class that allows finite length paths. Figure 9.12 shows that eight homotopy classes are explored in the same map with that in Figure 9.11. The two homotopy classes where HARRT* fails but TARRT* succeeds are framed in red for emphasis.

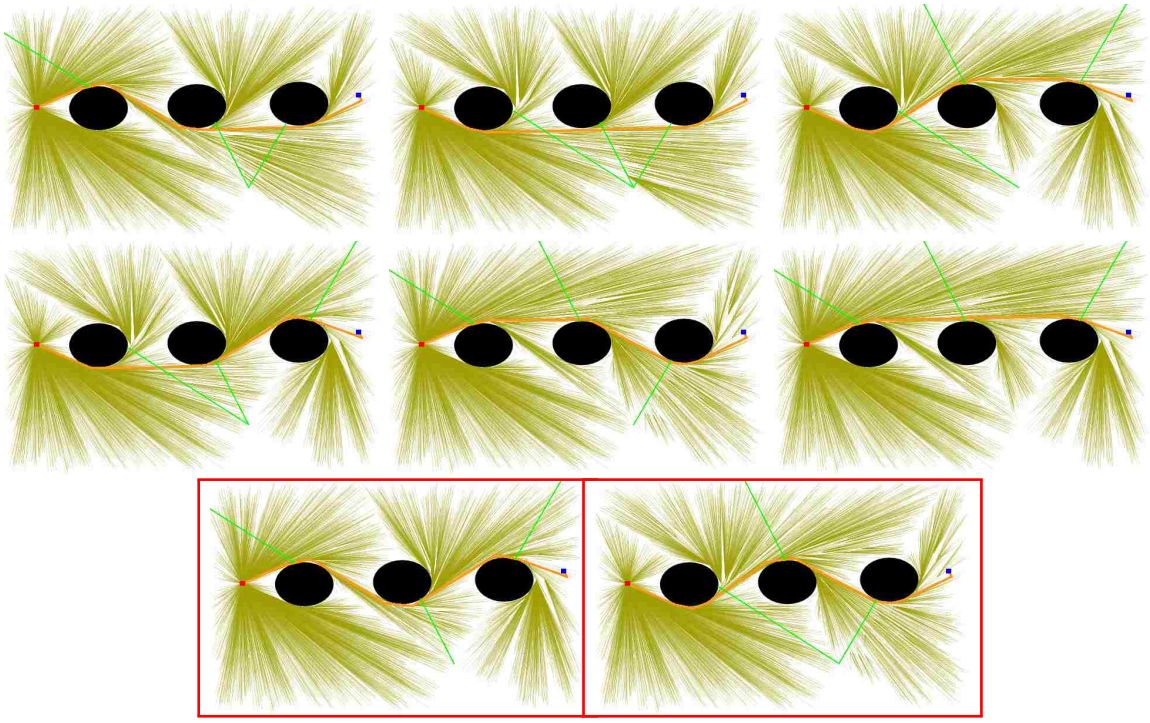


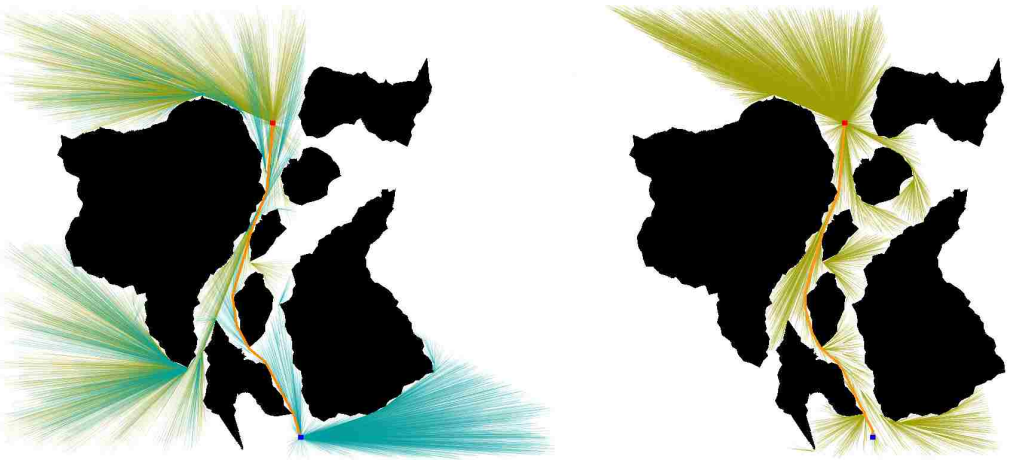
Figure 9.12: TARRT*.

9.6.3 Narrow Passage

A case of narrow passage is shown in Figure 9.13, where the path is required to pass through the narrow area in the map. We collect the results of running HARRT* and TARRT* 20 times respectively.

Figures 9.14a and 9.14b show the how well performance improves over time using shaded plots in HARRT* and TARRT*, respectively. The solid lines indicate the mean path cost over the 20 iterations and the shaded boundaries indicate one standard deviation. Notice how HARRT* converges more quickly. The explanation is simple: since HARRT* uses a bidirectional structure, it focuses sampling effort more efficiently than TARRT*. The efficiency of TARRT* can be improved by importing a bidirectional structure.

Figure 9.14c reemphasizes the efficiency of HARRT* by showing in a box plot of first iteration that a feasible path is found. Again, because HARRT* focuses sampling effort within a string block, it finds feasible paths more quickly. Figure 9.14d plots the error bar of the costs along the iterations. Again we see that it takes fewer iterations for HARRT* to



(a) HARRT* (b) TARRT*

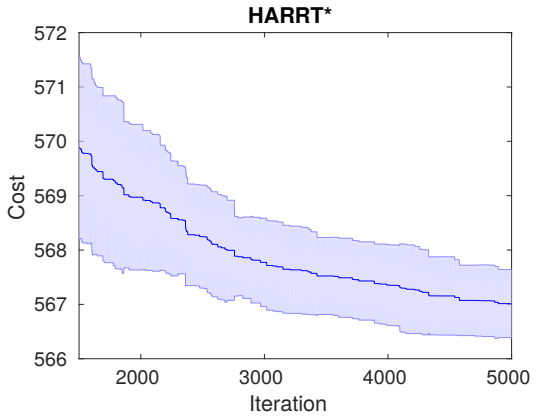
Figure 9.13: Narrow passage.

find a path than TARRT*. We can also see that the solutions of HARRT* converge to an optimal solution faster than those of TARRT*.

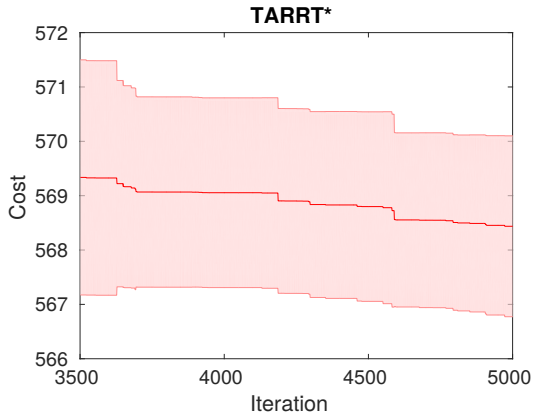
9.6.4 Winding Topology

We also provide a test case of a winding topology, in which the path is expected to go around an "x"-shape building clockwise and then pass along the right side of the bottom-right building. Figure 9.15a shows how the topology is defined, and Figure 9.15b shows the best path of the homotopy class that is found. In this case, the objective is to maximize the distance from the obstacles and the world boundary. The costmap is visualized in Figure 9.15b, in which lighter shades represent higher costs and darker shades lower costs.

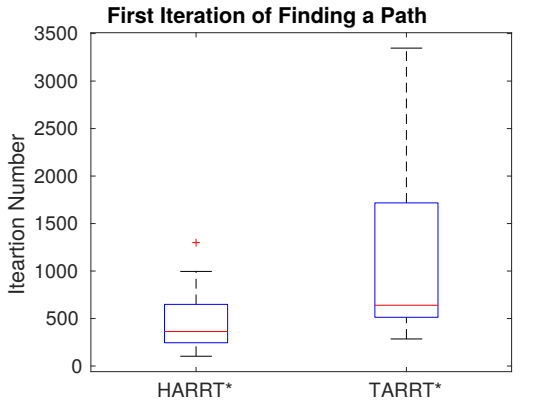
Because TARRT* has includes random sampling, we collected data from running TARRT* 20 times. Figure 9.15c shows the shaded error plot of the cost convergence and Figure 9.15d shows how fast TARRT* can find a feasible path subject to the defined homotopy class. Note that HARRT* is unable to solve this problem, because it cannot support a winding topology.



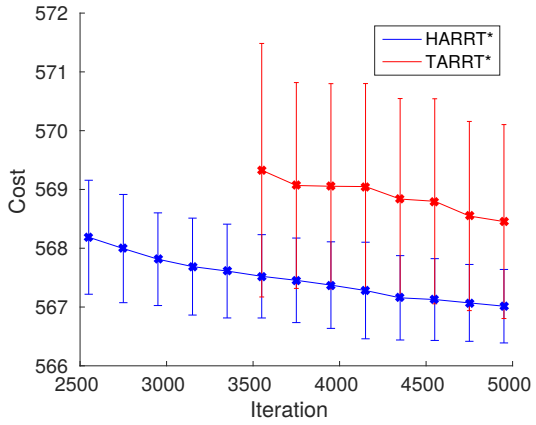
(a) HARRT*



(b) TARRT*

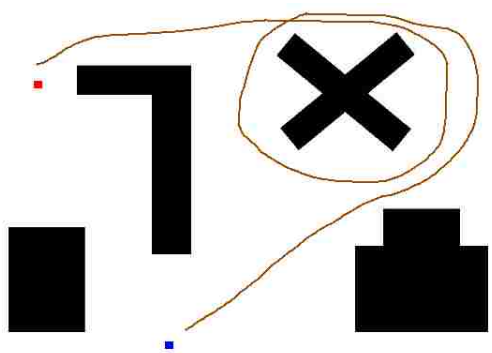


(c) First iteration of path found

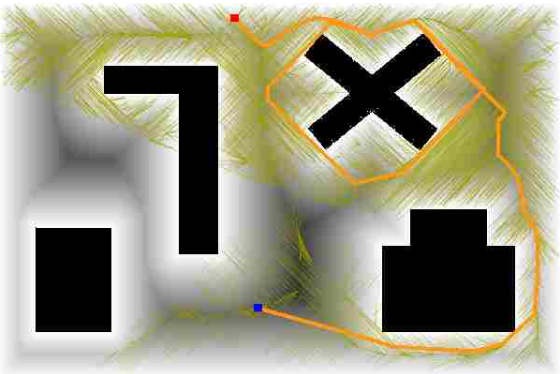


(d) Convergence with standard deviation

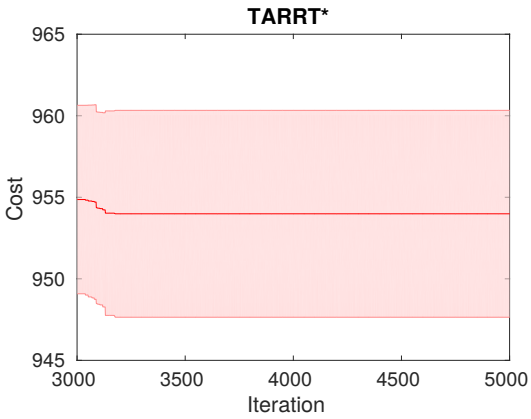
Figure 9.14: Performance comparison in narrow passage.



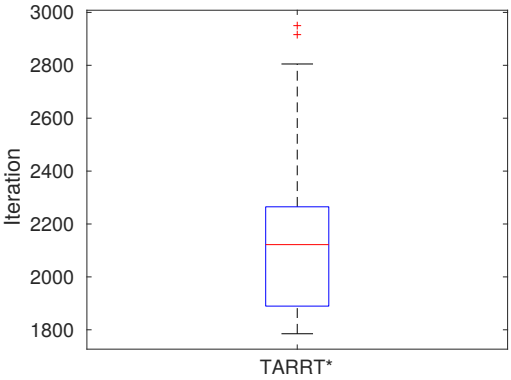
(a) Sketch



(b) Planned path

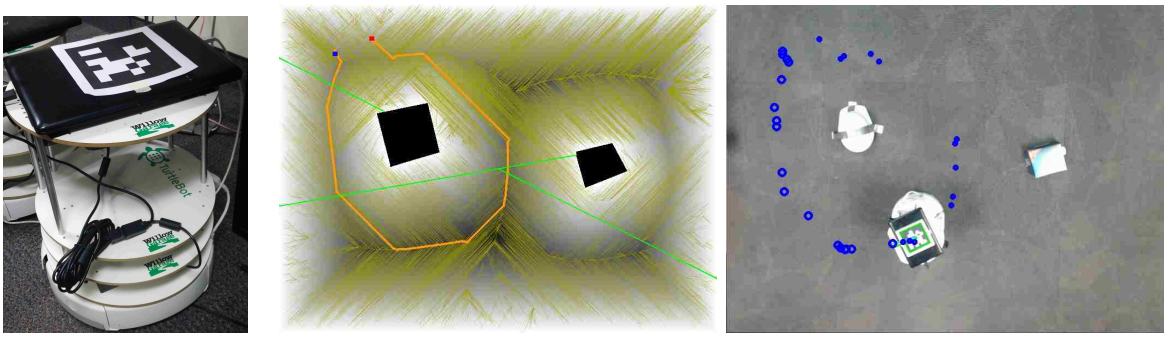


(c) Sketch



(d) Planned path

Figure 9.15: Winding Topology.



(a) Turtlebot (b) Planned path (c) Navigation

Figure 9.16: Robot navigation.

9.6.5 Physical Robot Demo

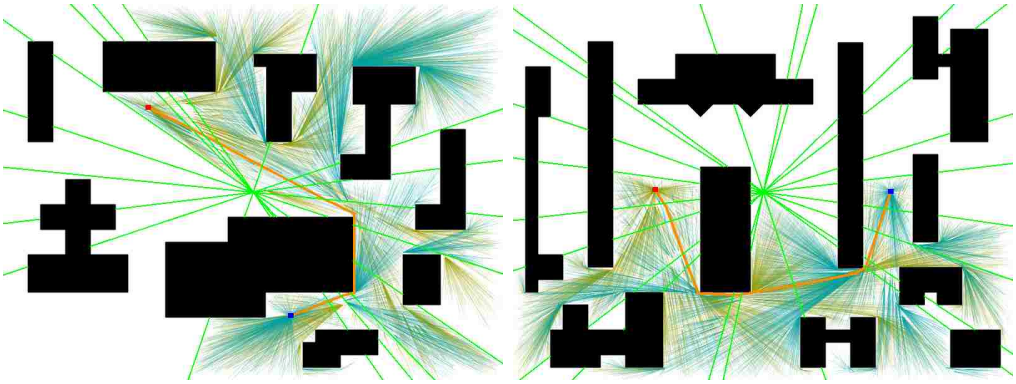
We implemented interactive topological path-planning system for a physical Turtlebot robot. Topological preference is obtained via a simple graphical user interface, which defines a topological constraint. Either of the topological path-planners, HARRT* or TARRT*, can be used to find an optimal path (Figure 9.16b shows the results from TARRT*) subject to the topological constraint. The resulting path is transferred to a sequence of waypoints. The Turtlebot then navigates by following the planned path by visual servoing, as shown in Figure 9.16c.

9.6.6 Single versus Multiple Homotopy Classes

Consider a path-planning problem where a human supervisor can express topological constraints in the following ways.

1. *Quickly go from point A to point B through a sequence of specific regions.* Topologically, such a path is constrained to one homotopy class, so this homotopy class becomes the constraint of the optimization problem and "quickly" becomes the objective to optimize [49].

2. *Quickly go from point A to point B making sure to visit some regions and avoid other regions.* Topologically, such a path is constrained to be among the set of homotopy classes that include the desired regions and avoid the undesired regions. The corresponding



(a) Example A (b) Example B

Figure 9.17: Optimal paths with hard constraints.

homotopic constraint restricts the optimal path to the set of homotopy classes that satisfy the requirements.

Note that we use the word "quickly" to represent the optimization criterion; in practice, many possible criteria exist. In the results in this section, we use the Euclidean distance as the objective to minimize because optimality can easily be verified. The objective can be replaced with any other type in applications.

Single Homotopy Class

This subsection considers the first way of expressing intent: *Quickly go from point A to point B through a sequence of specific regions.* In this case, the algorithm simply seeks to find the path that minimizes the Euclidean distance between two points subject to the path belonging to a single homotopy class.

HARRT* can often be efficient for the simple examples in this section because HARRT* performs an "early check" using the string block that avoids exploring a lot of the state space. This results in search efficiency and often produces acceptable paths, but as shown in a previous section it possible to construct examples where this early check would prevent the discovery of the optimal path. Future work should explore variations of the StringCheck() method that allow a budgeted amount of deviation from a path that generates a non-repeating

string. For example, the algorithm could leave the current string block by crossing a reference frame but do not cross another reference frame. Effectually, this example means that the algorithm could allow a two character palindrome to be part of the string, allowing exploration of paths that leave a subregion to avoid an area of high cost and then return to the subregion once they have circumvented the high cost area. Certainly, this future work would need to explore tradeoffs in the deviation budget, the spacing of sample points that generate the radial structure, and the structure of the cost function, but the future work might be able to find an "in between ground" that approaches the results of TARRT* without using as much time since it wouldn't sample from as large a set of string blocks.

Multiple Homotopy Classes

This section considers the second way of expressing intent: *Quickly go from point A to point B making sure to visit some regions and avoid other regions.* In this case, the set of regions to visit and regions to avoid create a set of possible homotopy classes.

This section gives an example of the kinds of possible solutions that can be generated when multiple homotopy classes are explored simultaneously. Figure 9.18, which will be referenced again in the next section, shows the optimal solutions returned by HARRT* for six different homotopy classes. For this second type of human intent expression, the path with the lowest cost would be returned.

It is perhaps unintuitive why we used HARRT* to solve this multi-class problem when TARRT* seems more appropriate. Recall that the cost function in this example is Euclidean distance. When the cost function is Euclidean distance it is easy to specify the string blocks within the homotopy classes that will contain the optimal solutions. For more complex cost functions, TARRT* would often be easire to use because it would not be obvious which string blocks would produce the optimal solution.

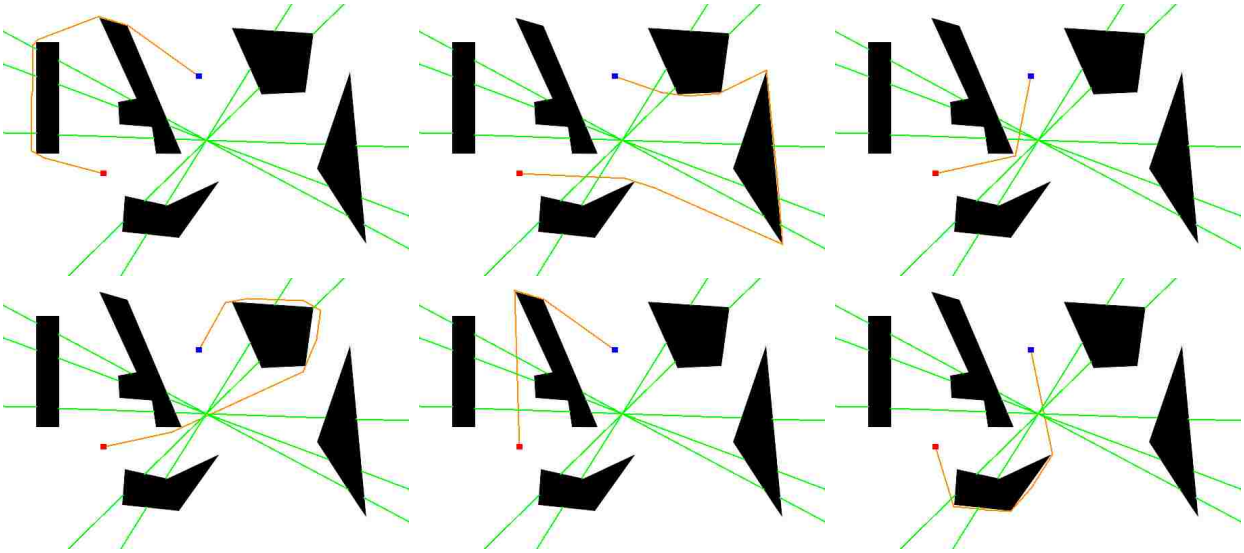


Figure 9.18: Optimal paths in six homotopy classes.

9.7 Summary

In this paper, we proved that the decomposition method and homotopic DFA can be (a) applied not only to simple paths but also to non-simple paths and (b) used to identify homotopic equivalence between different paths. We presented the HARRT* and the TARRT* algorithms, which used the decomposition method and which provided efficient sampling structures for exploring topological constraints over multiple homotopy classes. HARRT* utilized a bidirectional sampling structure that explores a subclass of a specified homotopy class; HARRT* had a limitation in what problems it can solve because it used a technical condition that we called the "via-point constraint". TARRT* enforced sampling that honored a set of possible homotopy classes and rewired the RRT* tree so that it explores multiple homotopy classes in parallel; TARRT* had a technical condition that we called the "stretching constraint", but this constraint could be tuned to tradeoff computation time for larger exploration. HARRT* showed better efficiency due to its bidirectional structure and its more efficient sampling within a homotopy subclass, while TARRT* provided completeness in searching all possible homotopy classes, including winding homotopy classes.

Future work should apply HARRT* and TARRT* to different decomposition methods and in higher dimensional spaces such as in a robotic manipulation problem using a 3D decomposition method. Practical complexity of the algorithm will likely become an issue for these problems.

Chapter 10

Expressing Homotopic Requirements for Mobile Robot Navigation through Natural Language Instructions [1]

Abstract

Allowing a human to express topological requirements to a robot in language enables untrained users to guide robot movement without requiring the human to understand sophisticated robot algorithms. By using a homotopy class or classes to represent one or more topological requirements, we build a framework that helps a robot understand a human's intent. This paper reviews a homotopic decomposition method that is used to convert any path into a string, which allows homotopic path equivalence to be performed by comparing strings. We then integrate the Homotopic Distributed Correspondence Graph (HoDCG) to infer the homotopic constraint in the format of strings from a language instruction. Finally, we use a homotopic path-planning algorithm that finds the optimal paths for a given objective and homotopic constraint. Experiment results show how a language instruction is converted into a path driven by an implicit topological requirement.

[1] To appear in 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems(IROS). Authors are Daqing Yi, Thomas M. Howard, Michael A. Goodrich and Kevin D. Seppi.

10.1 Introduction

Language-based interactions between a human and a robot theoretically extend the scope of interaction from only trained users to anyone who can use language. Language-based interactions require that a robot can understand what a human supervisor intends when he or she describes a task. Since humans are ostensibly good at high-level spatial reasoning, telling a robot where to move and where to avoid is a direct and efficient way for a human to express intent in assigning a task. With proper algorithmic support, humans need not transform the human-like high-level information into robot-based quantitative models for robot path-planning. We address this by allowing a human supervisor to specify a path topology and describe it to a robot. For example, a human could say "go to the left of the fountain and on to the hospital" in order to avoid the ambulance in Figure 10.1a, and "go between the hotel and the shop on the way to the hospital" in order to keep away from the traffic in Figure 10.1b.

(a) Left of the fountain (b) Between shop and hotel

Figure 10.1: Topological requirement for navigation

Such constraints can be expressed by, for example, having a human draw a path on a map, but it is desirable to explore other ways of expressing constraints. One advantage of using language to express constraints is that a human could think and express without converting into a graphical or robotic perspective. Allowing the human to use language

to express topological constraints requires that a robot is able to understand an abstract topological description and then plan a path that honors the desired topology. In this paper, a topological constraint for robot path-planning is derived from a language instruction. The topological constraint defines a set of eligible paths that have the required topology shape. The path-planning problem is finding the optimal path within this set of paths.

The mathematical notion of a homotopy is widely used in defining a topology constraint of paths because it defines the similarity between paths within the same path topology. When a path can be deformed into another path without encroaching into any obstacle, the two paths are homotopic [12]. In path-planning, when the start position and the end position are constrained, all the feasible paths can be classified into homotopy classes [12], each of which includes all the paths that are homotopic. In this paper, we explore the spatial relations between paths and objects in a world. By considering objects as obstacles, we can represent a topological requirement by a homotopy class or a set of homotopy classes. We express a topological requirement in a problem of instructing robotic navigation by

- translating a language instruction into a homotopic requirement; and
- planning a path subject to the homotopic requirement.

10.2 Related Work

There are a few approaches to helping a robot understand the topological information in a language instruction. Understanding the requirement of an instruction for execution planning depends on grounding the spatial information according to the phrases [63]. An execution plan is inferred from the grounded information. A semi-structured grammar, e.g. Tactical Behavior Specification [9, 15], is proposed as a guide for a human to express commands that a robot can understand. The grounded spatial constraint from a command is integrated into path-planning algorithms to get paths. The language understanding is extended to all types of natural language. Language examples are used to train semantic parsers to extract intents [25, 76]. Execution plans could be obtained by applying the extracted intents on the

world. For better generalization, graphical models are used to model the relationships between phrases and groundings [112]. In [51], DCG (Distributed Correspondence Graph) grounds implicit constraints, which model the correspondences between phrases and groundings. The correspondences between phrases and groundings can be factorized by conditional independence assumption. A path-planning problem can be created to generate a path that satisfies an inferred constraint. It has been successfully applied in spatial-constraint inference from instructions [51].

As we are interested in inferring homotopic constraints, we need a format of representing a homotopy class for identification. The homotopy class identification depends on the recognition of the spatial relation between obstacles and a path. Homology, a different type of similarity, is used as an approximation to homotopy. In [12], a map with obstacles is modeled into a complex plane with undefined points. Then the homology of paths can be recognized by comparing the complex integral values based on Cauchy theorem. Decomposition is the most common approach to identifying the homotopies of paths. Voronoi diagram, the classic decomposition method, is introduced to decompose a map that generates a topology structure of decomposed disjoint regions. Different walks on this structure derive reference paths of different homotopy classes [8]. Paths of diverse homotopy classes can be obtained by deforming the reference paths. Another way is using a set of line segements that decompose a map as reference frames [48]. How a path sequentially crosses the reference frames can be used to represent its topological shape.

In addition to a homotopic constraint, a human also intends a specific objective in a language instruction. Most motion-planning algorithms for homotopic constraints can only find feasible paths [8, 48] or the shortest paths [44, 49]. We also include an objective in planning a path, which leads to an optimal path-planning problem.

In this paper, we propose a framework for language-guided motion-planning algorithm, which translates a language instruction into a path-planning problem and finds the optimal

path accordingly. We describe the framework in Section 10.3, and provides experiment results in Section 10.4 to support its performance.

10.3 A Framework of Language-instructed Path-Planner

We propose a language-instructed path-planning system that can understand the homotopic requirement of a language instruction and generate the optimal path subject to the homotopic requirement. Figure 10.2 shows the structure of the system. The Homotopic Distributed Correspondence Graph (HoDCG), discussed in Section 10.3.1, infers spatial relations from a language instruction. A map is decomposed according to the spatial relations in Section 10.3.2, using a homotopic Deterministic Finite Automata (homotopic DFA) [128] to convert paths into string representations that honor topological similarities. The spatial relations are then used to generate rules that define *qualified reference frame sequences* in Section 10.3.3; these sequences implicitly define a grammar of eligible strings. The joint set of this eligible string grammar and the grammar defined by the homotopic DFA is the inferred homotopic constraint. The Homotopy-Aware RRT* (HARRT*) [128], a homotopic path-planner, is then used to find the optimal path in the homotopic constraint in Section 10.3.4.

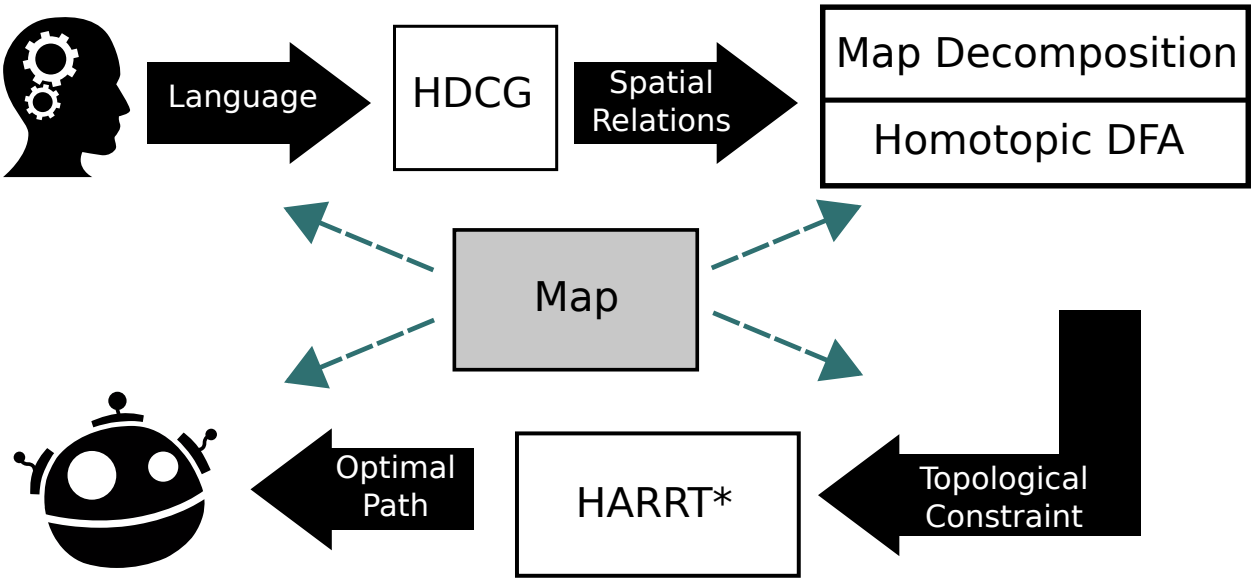


Figure 10.2: System Framework

10.3.1 Inferring Spatial Relations from Language

In this section, we adopt a graphical model that grounds spatial relations from a language instruction. The spatial relations are used to derive a homotopic constraint. The homotopic constraint represents a human supervisor's requirement of path topology.

Our graphical model is derived from the Distributed Correspondence Graph (DCG) [51], which is a factor graph that can efficiently ground language sentences. The graph consists of a set of factor models. Each factor model defines a relationship among a correspondence ϕ_i, a phrase λ_i, a grounding γ_i, and a set of child elements Γ_{c_i}. A factor model is visualized in Figure 10.3.

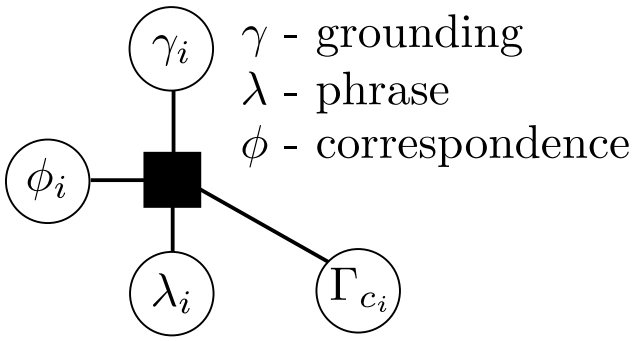


Figure 10.3: Factor model

Essentially, such a factor graph builds a distribution that represents the correspondences between groundings and a sentence. This distribution depends on a phrase structure that is parsed from a sentence by a grammar parser. Figure 10.4 gives an example of a parse tree from a sentence "walk by the left of the table". The parse tree determines the factor graph. The factor graph (Figure 10.5) is created by assembling the correspondences between groundings and phrases (Figure 10.3) according to the parse tree (Figure 10.4). The inference of the factor graph returns a set of groundings. The details are stated in [51].

We propose HoDCG (Homotopic Distributed Correspondence Graph) algorithm that extends DCG for supporting homotopic requirements. HoDCG includes new types of groundings to represent homotopies. Inferred groundings are organized to construct a homotopic constraint by phrase structure information.

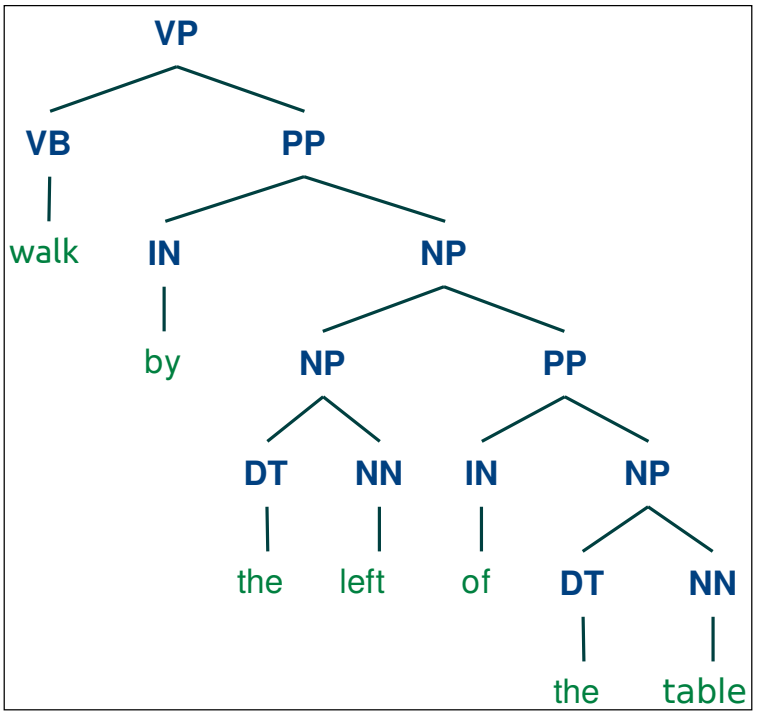


Figure 10.4: "walk by the left of the table"

HoDCG includes a graph model that supports spatial relations as new groundings and a mechanism that derives homotopic constraints from associated groundings. Figure 10.5 illustrates a structure of HoDCG, which extends the phrase structure in Figure 10.4. Given the phrases as observed nodes, the inference process is a bottom-up maximum likelihood search, as formalized in Equation (10.1).

$$\Phi^* = \arg\max_{\phi_{ij} \in \Phi} \prod p(\phi_{ij} \mid \gamma_{ij}, \lambda_i, \Gamma_{c_{ij}}, \Upsilon). \tag{10.1}$$

$p(\phi_{ij} \mid \gamma_{ij}, \lambda_i, \Gamma_{c_{ij}}, \Upsilon)$ corresponds to a factor model, as shown in Figure 10.3, Υ is the world, and the output is a set correspondence values. As illustrated in Figure 10.3, when a correspondence variable is true the grounding associated with the correspondence variable is said to correspond with a given phrase.

In implementations, each factor model $p(\phi_{ij} \mid \gamma_{ij}, \lambda_i, \Gamma_{c_{ij}}, \Upsilon)$ is approximated by a log-linear model with binary features, which is written as in Equation (10.2). Tuning the binary feature weight μ changes the approximation of the probability p. The Limited-memory

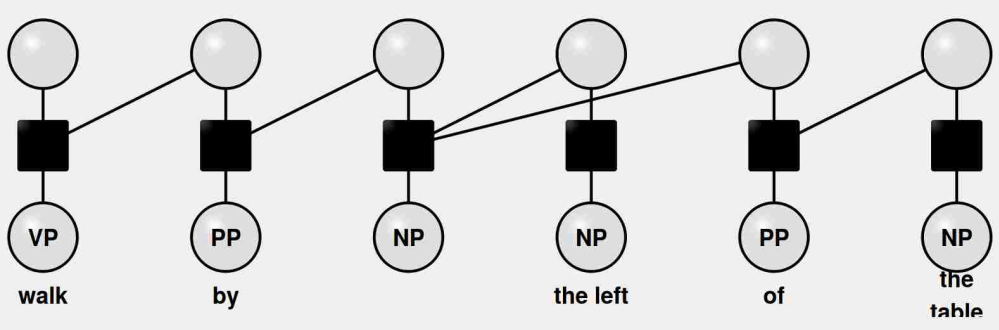


Figure 10.5: HoDCG structure of "walk by the left of the table".

Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) [71] algorithm is used in the optimization in the training process.

$$\Phi^* = \arg\max_{\phi_{ij}\in\Phi} \prod \Psi(\phi_{ij}, \gamma_{ij}, \lambda_i, \Gamma_{c_{ij}}, \Upsilon),$$
$$\Psi(\phi_{ij}, \gamma_{ij}, \lambda_i, \Gamma_{c_{ij}}, \Upsilon) = \frac{e^{\sum_l \mu_l f_l(\phi_{ij}, \gamma_{ij}, \lambda_i, \Gamma_{c_{ij}}, \Upsilon)}}{\sum_q e^{\sum_l \mu_l f_l(\phi_q, \gamma_{ij}, \lambda_i, \Gamma_{c_{ij}}, \Upsilon)}}. \tag{10.2}$$

HoDCG grounds spatial relations from a language instruction. We use the spatial relations to obtain a homotopic constraint.

10.3.2 Encoding Path Homotopy

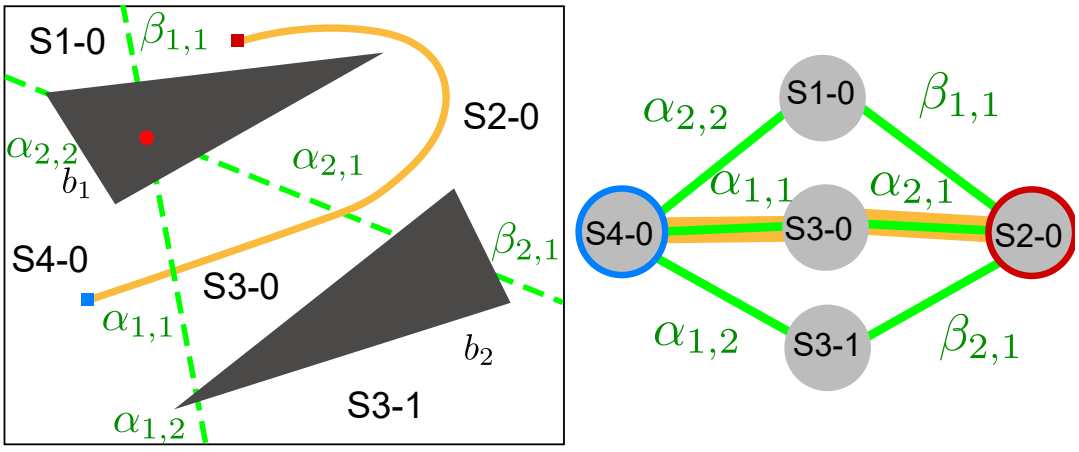
We firstly need a format that encodes the shape of a path, which represents a homotopy class. By the definition of homotopy, we know that paths in the same homotopy class share the same topology that is defined by spatial relations with obstacles. This implies a connection between a homotopy class and a given spatial relation.

We use a decomposition method [128] that divides a map into subregions, which is derived from the Jenkins method as in [48]. The decomposition supports the extraction of topological information, which is obtained by how the path sequentially visits the subregions. Based on the proposed decomposition method, a homotopic DFA (Deterministic Finite Automata) $\boldsymbol{M}^h$ [128] can be constructed that converts a path σ to a string representation v,

which is written as $v = \boldsymbol{M}^h(\sigma)$. The string representation is used to identify the homotopic equivalence, which distinguishes topological difference of paths.

In this paper, we present a novel decomposition method, which is modified from the one in [128]. The change we made is moving the center point c into an obstacle so that more generated reference frames are associated with the obstacle. Specifically in our case, the center point locates in one of the obstacles that is associated with inferred spatial relations. This makes the decomposition centered at the obstacle.

The map decomposition method starts with a random sampling process. First, a representative point b_k is randomly sampled from each obstacle B_k. This point may not lie on a line that connects any other two representative points. A center point c is sampled inside one of the obstacles, which must also not lie on a line that connects any two representative points. Restricting the center point in an obstacle reduces ambiguity in homotopy identification. A radial structure is created from the center point c toward all other representative points b_k. Obstacles in the map cut the radial structure into line segments. The end of each line segment terminates within either an obstacle or a map boundary. We use the line segments to identify the homotopy of paths. The line segments are defined as *reference frames* $\boldsymbol{R}$, which separate the map into subregions $\boldsymbol{S}$. Figure 10.6a gives an example of the map decomposition.



(a) Ref. frames

(b) State trans. (ST)

Figure 10.6: Map with obstacles.

A sequence of reference frames that a path visits reveals the topological information. We assign an *ID character* to each reference frame. The sequence of reference frames can be represented by a *string* of ID characters. In a path-planning problem, both a start position and a goal position are given, which imply a start subregion and a goal subregion, respectively. In Figure 10.6a, the blue point is the start position and the maroon point is the goal position. Let each subregion be a state of the DFA, and each reference be an edge of the DFA. We can have a DFA [128] that represents how subregions in Figure 10.6a are connected in Figure 10.6b. The DFA in Figure 10.6b is then modified for homotopic equivalence, which derives a homotopic DFA [128] in Figure 10.6b.

The homotopic DFA abstracts any path into a string representation. The REPTrim() algorithm in [128] translates a string for any path into the smallest possible string for any path within that homotopy class. Thus, the homotopy of any two paths can be identified by comparing the strings generated by the homotopic DFA after processed by the REPTrim() algorithm. Notice that we use a map decomposition method that is slightly different from our prior work in [128]. Because the center point is moved into an obstacle, there is no reference frame that is connected to the center point. It removes the ambiguity brought by reference frames that are connected to the center point, but preserves the validity of other properties. We restate the key property from [128] because it will be important to the results of this paper:

Theorem 3. $\text{REPTrim}(\boldsymbol{M}^h(\sigma_i)) = \text{REPTrim}(\boldsymbol{M}^h(\sigma_j))$

This theorem means that $\text{REPTrim}(\boldsymbol{M}^h())$ encodes a path into a string, and we can use the strings of paths to identify homotopies.

10.3.3 Rules from Spatial Relation Functions

As discussed above, we suppose that humans can reason and express intent using spatial language. Thus, natural language does not necessarily describe a path topology. Rather, a human's instruction implies a spatial relation that constrains the allowed path topologies. In

Section 10.3.2, a homotopic DFA is used to represent abstract topologies in a string format when a map, a start position and a goal position are known. In this section, we describe how to determine which of the strings generated by a homotopic DFA satisfy a spatial relation constraint specified by a human.

Suppose that a spatial relation constraint is always associated with objects in a map. We define a *spatial relation function* over objects O and returns a corresponding rule ℓ.

A path will never cross multiple reference frames at the same time. Given this condition, we propose a set of operators that assemble ID characters of reference frames into strings, and describe how these character operators correspond to rules on spatial relations.

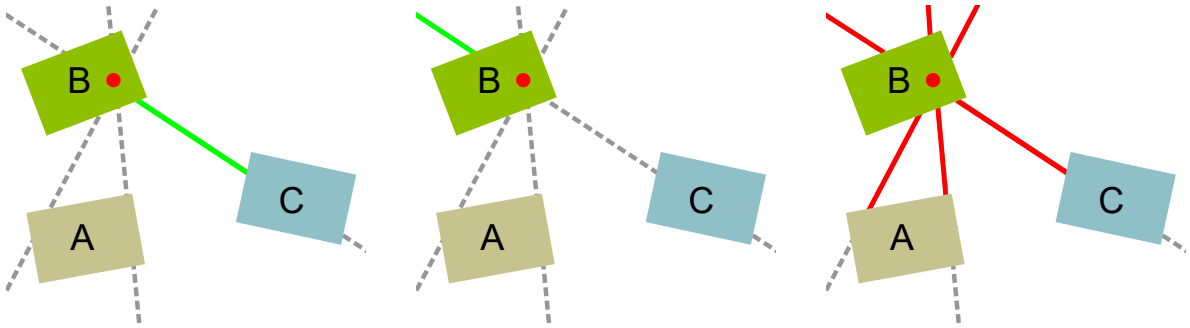
- **CONCATENATION** $\cap$ is used to concatenate two ID characters. It corresponds to a sequential order of visiting corresponding reference frames.
- **UNION** $\cup$ is used to union two ID characters. It indicates that a path may visiting either of the regions that correspond to the two ID characters of the reference frames.
- **NEGATE** $\neg$ is used to negate one ID character. It indicates that the region associated with an ID character should be avoided.

The following examples of spatial relation functions illustrate how these functions are translated into operators on the IDs:

- $(r_1 \cup \cdots \cup r_n) = \textsc{InBetween}(o_1, o_2)$,
- $(r_1 \cup \cdots \cup r_n) = \textsc{LeftOf}(o) \mid \textsc{RightOf}(o) \mid \textsc{TopOf}(o) \mid \textsc{BottomOf}(o)$,
- $\neg(r_1 \cup \cdots \cup r_n) = \textsc{Avoid}(\textsc{InBetween}(o_1, o_2))$,
- $\neg(r_1 \cup \cdots \cup r_n) = \textsc{Avoid}(\textsc{LeftOf}(o))$,
- $\neg(r_1 \cup \cdots \cup r_n) = \textsc{Avoid}(o) = \textsc{Avoid}(\textsc{LeftOf}(o) \cup \textsc{RightOf}(o) \cup \textsc{TopOf}(o) \cup \textsc{BottomOf}(o))$.

A *rule* ℓ describes a logical proposition of reference frames in the disjunctive normal form. For example, a sentence "go between B and C " for the world in Figure 10.7a is

translated to the rule InBetween(B, C), which in turn returns the ID character from visiting the reference frame that connects object B to object C. A sentence "go by the left of B" is associated with LeftOf(B) returns ID characters that locate at the left of object B, which is shown in Figure 10.7b. Also, a sentence "avoid B" is associated with Avoid(B), which equals to avoiding four direction near object B. Also by the logic, we can have $\neg(r_1 \cup \cdots \cup r_n) = \neg r_1 \cap \cdots \neg \cap r_n$. It indicates that all the reference frames associated with object C should be avoided.



(a) InBetween(B, C) (b) LeftOf(B) (c) Avoid(B)

Figure 10.7: Spatial Relation Functions.

We can see that a spatial relation function can return multiple associated reference frames. Assembling rules by operators into a new rule supports complicated semantic in a language expression.

- **Sequence** There are often many orders implied in forcing a sequence of ID characters of reference frames. For example, "go to left of A before going to bottom of B". Let ℓ^A be a rule for spatial relation with object A and ℓ^B be a rule for spatial relation with object B. We can have the rule for the command as $\ell^A \cap * \cap \ell^B$.
- **Reverse** It is used to negate a given rule. For example, "never go to left of A". It means all the reference frames that are on the left of object "A" should be avoided, which is written as $\neg\ell^A$.

If we do an exhaustive search on the topology of homotopic DFA, we can have all the possible strings from a start state to a goal state. Each string represents a sequence of

reference frames. Rules derived from a spatial relation function can be used to filter out ineligible strings. The set of eligible strings defines a homotopic constraint. Only paths that the homotopic DFA generates strings belonging to the set are considered in planning.

10.3.4 Homotopic Path-Planning

We use a homotopic path-planner to find a path by a homotopic constraint. Because of the lack of precision in an instruction, one or more homotopy classes may be included in defining a homotopic constraint. In above sections, we can derive a homotopic constraint from a language expression.

We also want to support adverb in a language instruction [123]. For example, "carefully" and "quickly" indicate different criteria in measuring the performance. This implies a specific objective to consider in planning a path. The objective differs with different verbs and adverbs in language. Thus, we define our problem as a *homotopy-based optimal path-planning problem* [128]. The objective and homotopic constraint are both inferred from a language expression.

We need an algorithm that could explore several homotopy classes in parallel and return the optimal path of each class, because a homotopic constraint could consist of several homotopy classes. HARRT* (Homotopy-Aware RRT*) proposed in [128] is a homotopy-based optimal path-planner, which utilizes the RRT* structure to explore a map according to homotopic information. A homotopic DFA is used to identify the homotopic information of each branch of an RRT* structure. The homotopic information is used to constrain the exploration only in necessary regions. There are two tree that are extended from the start position and the goal position bidirectionally. By the asymptotic optimality of RRT*, both trees converge to optimal structures. To any position, one provides an optimal-to-come subpath, while the other provides an optimal-to-go subpath. Concatenating two subpaths gets a path that is optimal from the start to the goal subject to a constraint of visiting the concatenating position. An example is shown in Figure 10.8. A node highlighted in orange

color in the goal tree. A red dash-line circle indicates the neighboring nodes for adding and rewire process in tree extension. A path in orange color is obtained from this node. At each iteration, one path is found in the start tree, and one path is found in the goal tree. If any new path is better than the current best found path in the hotomopy class it belongs to, the optimal found path will be updated. The optimal paths of multiple homotopy classes can be obtained, because of the variation in selecting concatenating positions. We can thus have the optimal paths of homotopy classes after iterations.

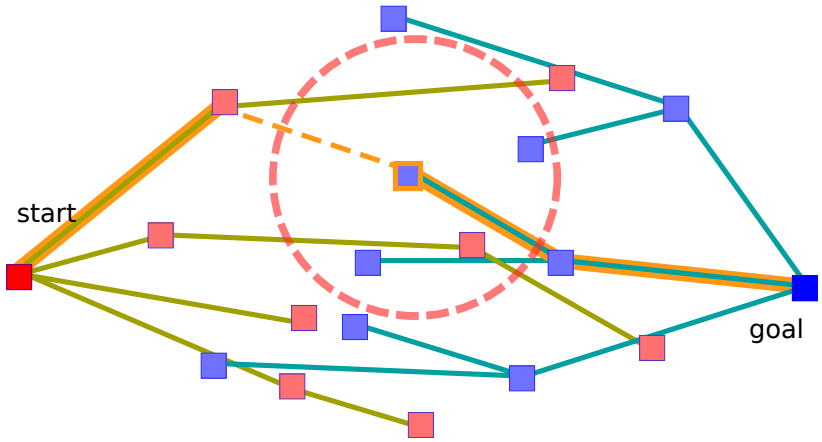
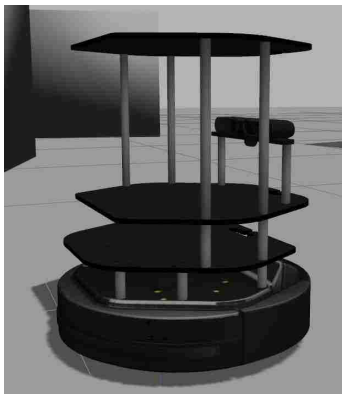


Figure 10.8: HARRT*

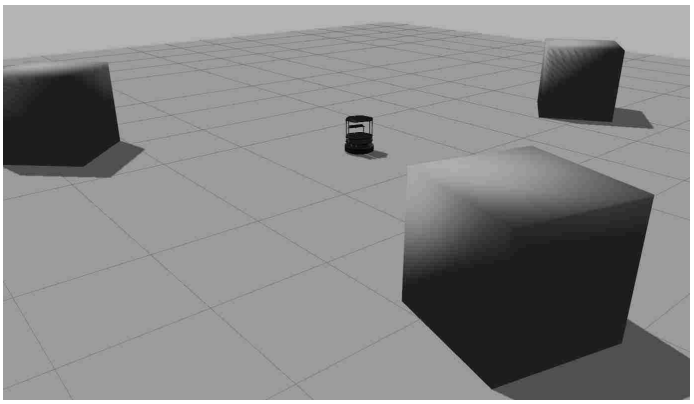
10.4 Experiment

In this section, we use simulations to validate the language-instructed path-planner. In [51], how the goal is inferred from a language instruction is demonstrated. In this paper, we focus on only how a homotopic constraint can be inferred and how it is used for instructing the navigation of a robot. The experiments were running in Gazebo simulator, in which we instructed a Turtlebot moving in among numbered boxes.

Currently the training of HoDCG has not been applied to a big dataset. But it has only been verified with a small dataset with five examples that enumerates all the possible spatial relation functions. The inferred result is consistent with the training dataset. Some examples are shown in Table 10.1. Because HoDCG uses the same training and inference



(a) Turtlebot



(b) Gazebo

Figure 10.9: Simulation environment.

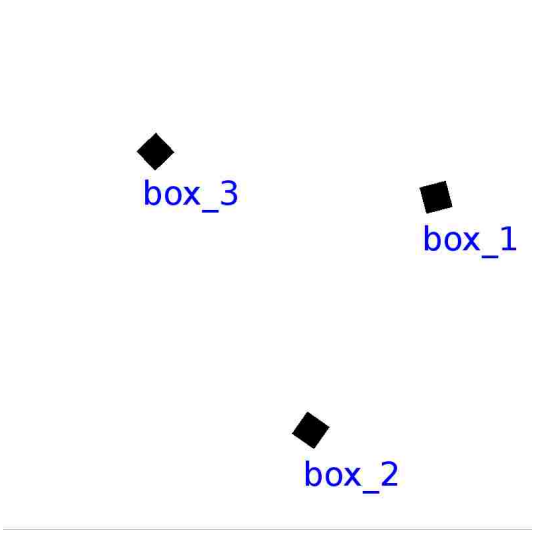
algorithm with DCG and only extends the search space with new types of grounding, we observe that HoDCG has a consistent performance with DCG [27, 51].

Table 10.1: Examples for training

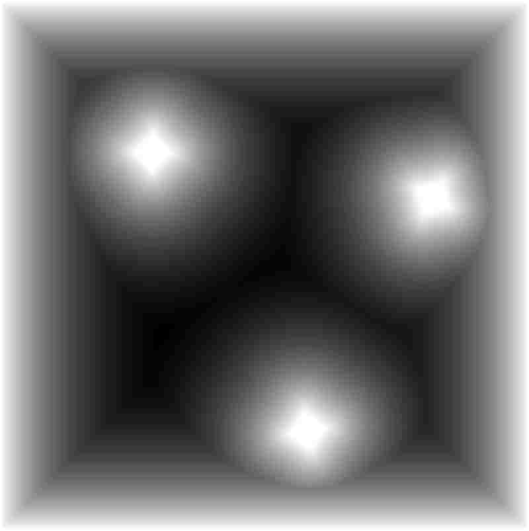
LeftOf(o)	"Swing by the left of ..."
InBetween(o_1, o_2)	"Go between ... and ..."
Avoid(InBetween(o_1, o_2))	"Avoid going between ... and ..."
Avoid(LeftOf(o))	"Stay away from the left of ..."

We obtained a map of the environment with three boxes from the Gazebo, which is shown in Figure 10.10a. Each box is labeled for reference. We choose maximizing the distance to any obstacle or environment boundary as the objective for planning, and have the costmap of the environment shown in Figure 10.10b. The darker a position is, the lower cost there is.

Figure 10.11 gives an example of an instruction "walk by the left of box 3". Figure 10.11a illustrates a HoDCG that is created from the instruction. From the inferred spatial relation, the path is required to two eligible reference frames, which are green bold lines in Figure 10.11b. The red point is the start position, and the blue point is the goal position. From the costmap in Figure 10.10b, the optimal path is obtained and is shown in orange color.

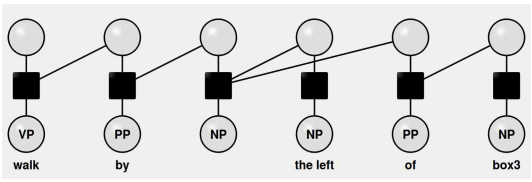


(a) Map with three boxes

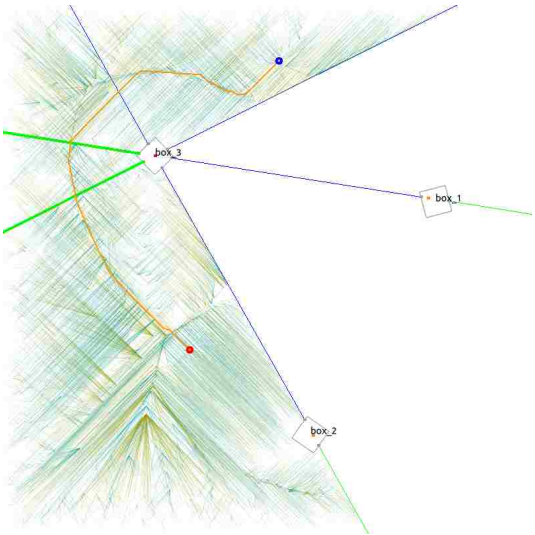


(b) Collision avoidance

Figure 10.10: Map and costmap of an environment with three boxes



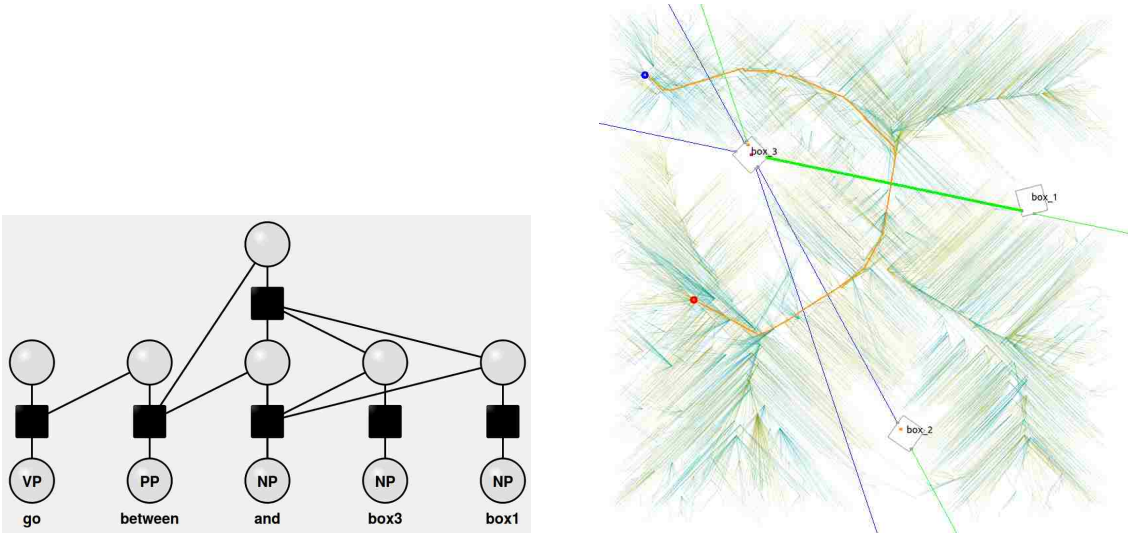
(a) HoDCG



(b) HARRT* and path

Figure 10.11: "Walk by the left of box 3".

Similarly, we can have a HoDCG structure generated for "go between box 3 and box 1", which is shown in Figure 10.12a. As a result, a reference frame that connects box 3 and box 1 is inferred to be required to visit, which leads to a planned path accordingly in Figure 10.12b.



(a) HoDCG (b) HARRT* and path

Figure 10.12: "Go between box 3 and box 1".

We also tested the negation of a spatial relation. Figure 10.13 shows an example for "Avoid going in between box 3 and box 2". A negation of in-between spatial relation is inferred from a corresponding HoDCG. A bold red line is shown in each subfigure of Figure 10.13, which indicates a reference frame that is not allowed to cross. Subfigures show three different solutions in different homotopy classes all satisfy this homotopic constraint. It reveals that there is lack of precision in the instruction. We can either take the optimal of the three as the solution for robot navigation or introduce an interactive process for human to select.

We also tested in a more complicated environment, which is shown in Figure 10.14a. There are nice boxes in the environment, which means a higher variety in spatial relations and a bigger number of homotopy classes. The corresponding costmap for collision avoidance is given in Figure 10.14b.

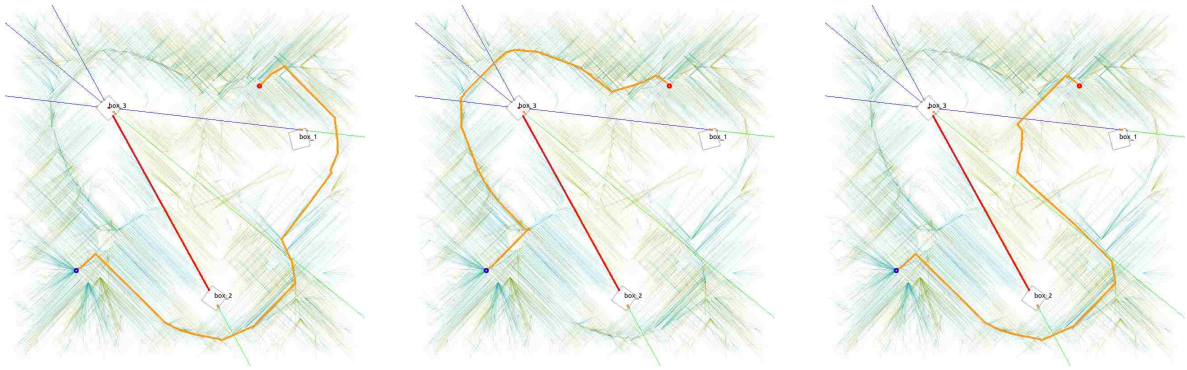
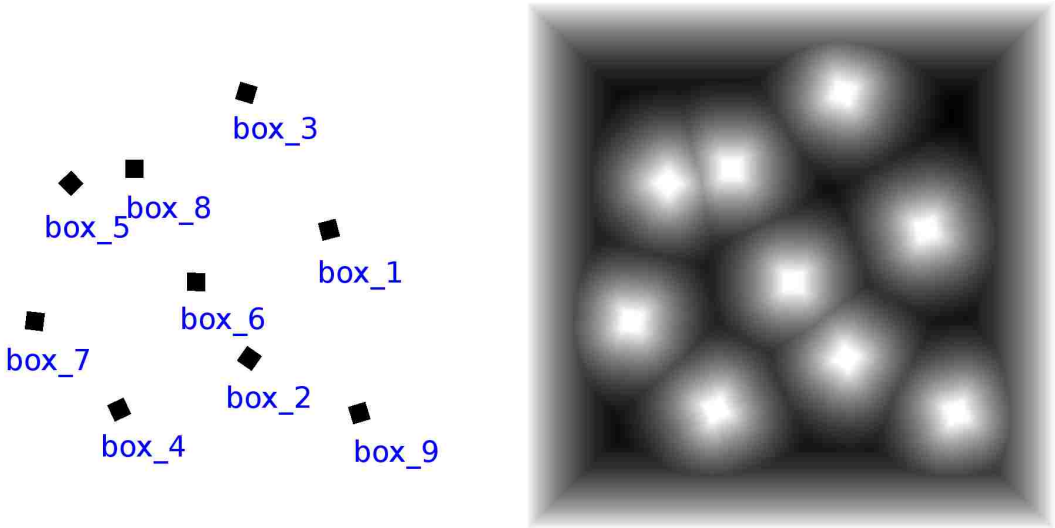


Figure 10.13: “Avoid going in between box 3 and box 2”.



(a) Map with nine boxes (b) Collision avoidance

Figure 10.14: Map and costmap of an environment with nine boxes

With more possible spatial relations and homotopy classes, the potential imprecision of a language instruction is increased. Figure 10.15 shows four paths of different homotopy classes all satisfy the inferred in-between spatial relation, which is inferred from an instruction "go between box 5 and box 8".

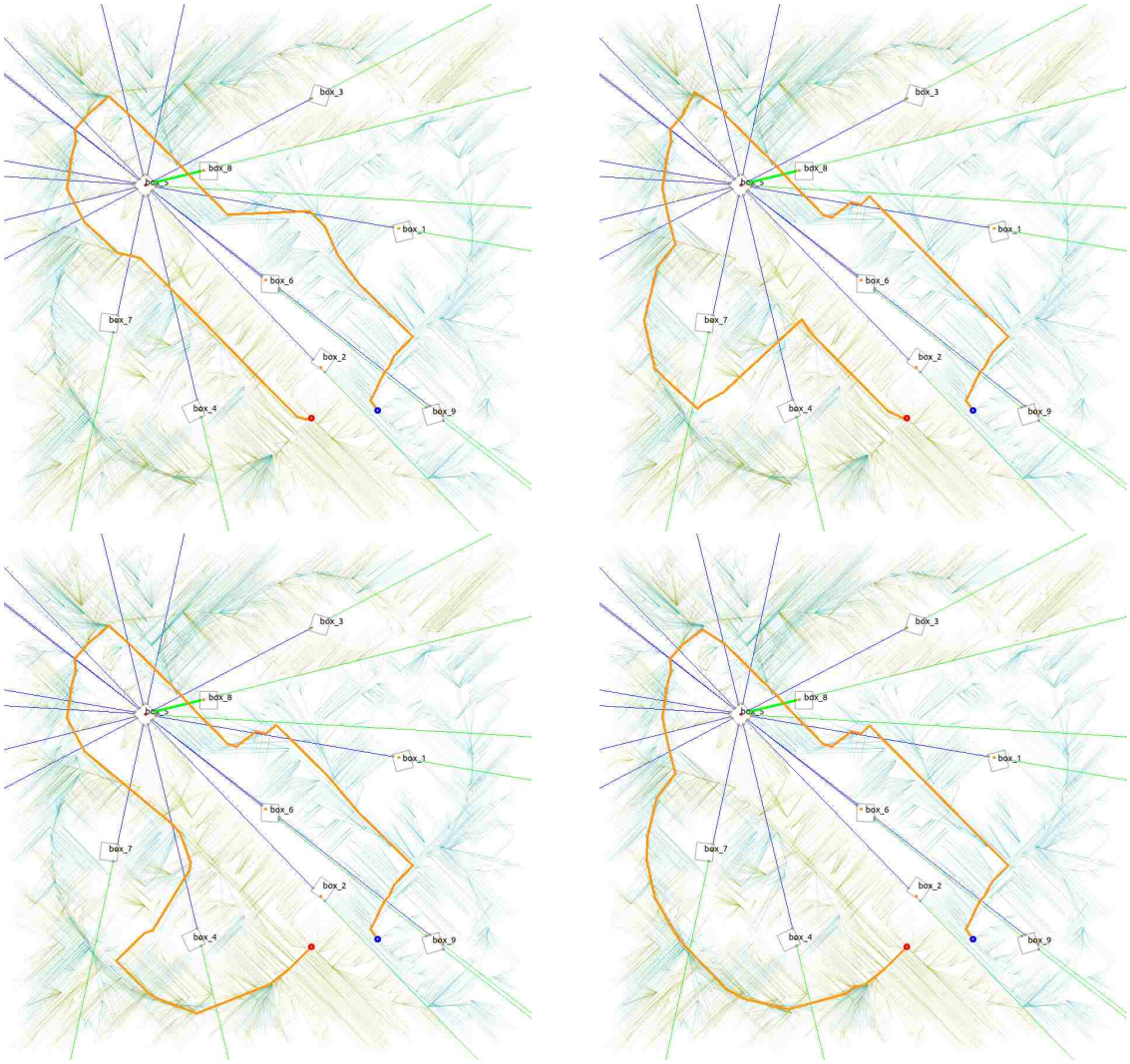


Figure 10.15: "Go between box 5 and box 8".

If fewer number of homotopy classes are to be considered, more constraints should be stated in an instruction to reduce ambiguity. Figure 10.16 shows an example of an instruction "Go between box 8 and box 6 and avoid the left of box 1". Less paths are obtained because less homotopy classes are eligible for the inferred homotopic constraint.

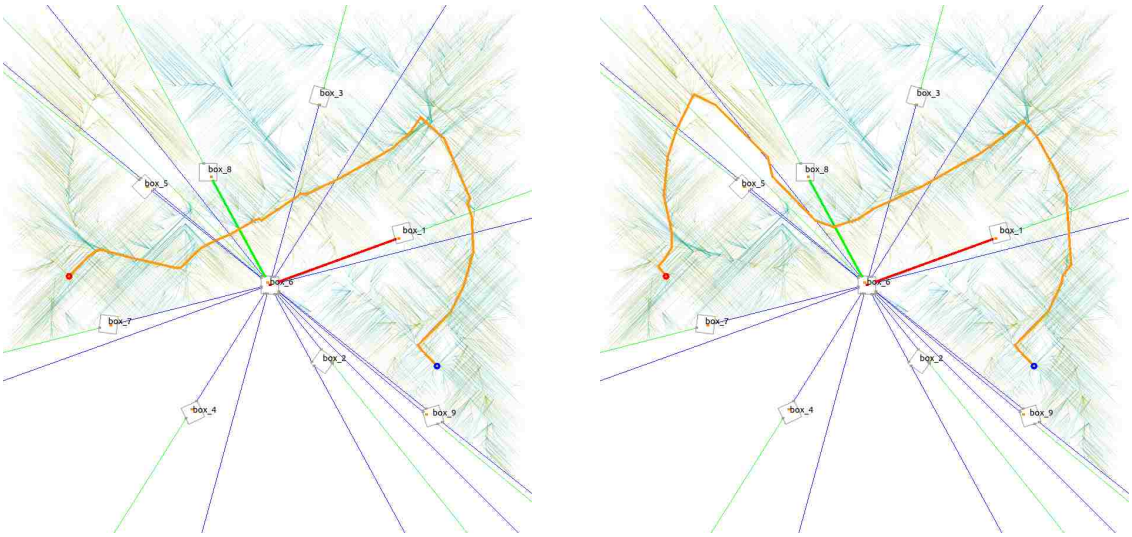


Figure 10.16: "Go between box 8 and box 6 and avoid the left of box 1".

10.5 Future Work and Conclusion

The sequence of spatial relations, which supports a sentence like "go ..., then ..." , is not included in the experiment yet. It depends on that the inference of HoDCG could provide sequential information of a set of groundings, which will be added in future work. Enabling the sequence of spatial relation would be another efficient way of reducing ambiguity in a language instruction, which constrains a problem to less number of eligible homotopy classes in a complex environment. We are also going to integrate the framework with other planning information for expanding the features of language instructions, which will be a hierarchical structure [27] that include goal constraints and different objectives. Soft constraints will then be introduced to better support human requirements. For example, in many scenarios, "avoid" only indicates a soft constraint instead of a hard one.

We propose a framework of language-instructed path-planning that integrates a language inference model (HoDCG) and a homotopic path-planning algorithm (HARRT*) so that a robot could read a homotopic requirement in a human's instruction and planning a required path. The translation from an instruction to a path consists of a mapping from

paths to strings, a graph model that interpret an instruction into homotopy classes and a path-planning algorithm that finds optimal paths of the homotopy classes.

Chapter 11

Summary and Future Work

11.1 Summary

The research presented in this dissertation constructs a framework that translates qualitative information from a human into quantitative paths for a robot. In the translation process, qualitative requirements are extracted from human information (e.g. natural language instructions). The qualitative requirements are converted into objectives and constraints, which define path-planning problems. Planners derive paths from planning problems and send them to a robot to execute.

In converting from qualitative to quantitative, we presented algorithmic solutions to the following four problems: grounding words to partially suport language understanding for path-planning, multi-objective path-planning, homotopy-based optimal path-planning, and submodular path-planning with a reference-path constraint.

11.2 Future Work

This section presents some natural extensions from current research. We present the future work as a broad theme, and then elaborate on key pieces of this broad theme.

In the research presented in this dissertation, we used objectives and constraints to translate from high-level reasoning to low-level planning. Human information is input as high-level reasoning. A navigation plan is output from the low-level planning, as in Figure 11.1. It is possible to do the reverse by taking raw data in the form of a demonstration be the input to the low-level planning and then learn a policy as output from the high-level

reasoning; this is illustrated in Figure 11.2. Such a bi-directional information flow between qualitative and quantitative can potentially enable better exchange of information between robot and human representations and problem solving.

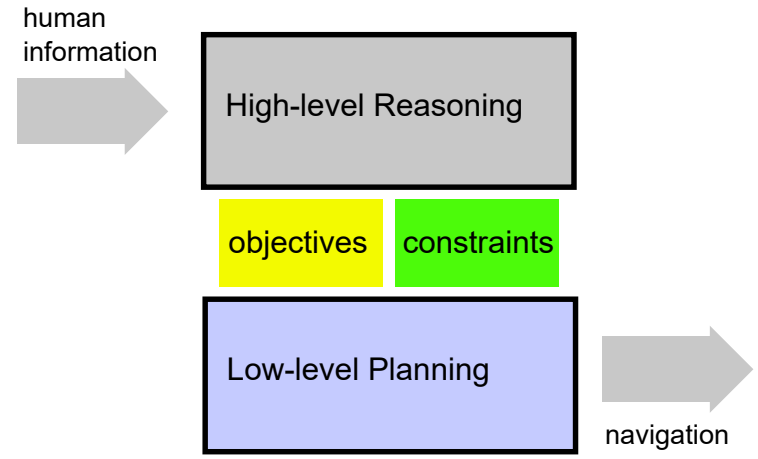


Figure 11.1: From high-level to low-level.

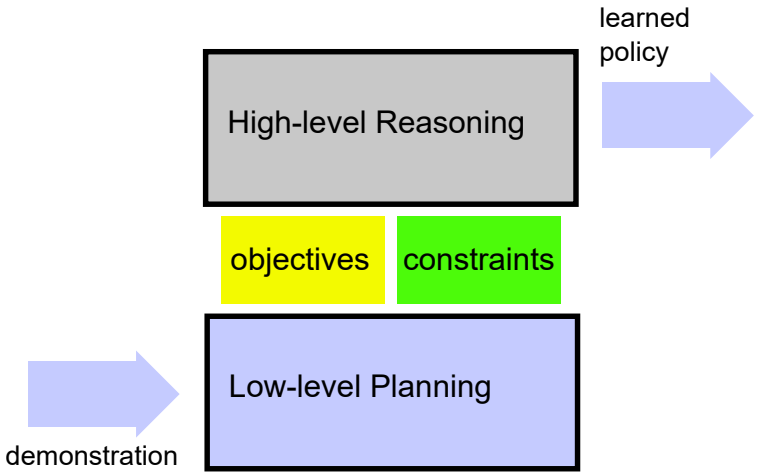


Figure 11.2: From low-level to high-level.

A natural next step is to extend human-robot collaboration so that it depends less on prior information. Properly done, perhaps the robot could think using high-level, human-like reasoning that makes it more robust to situation changes and utilizes more types of human information.

In many scenarios, there is no precise map available for execution planning. A human only provides principles or a rough plan for execution, e.g. a topology defines spatial

relations with landmarks in a navigation task. A robot needs be capable of updating the map information through observation will executing and then re-planning local paths that follow the humans requirements in an online manner. Future work should extend multi-objective planning and homotopy-based optimal planning to support dynamic and unknown environments. These extensions require that the algorithms can adapt to situation changes and execute re-planning efficiently. A robot can make autonomous decision for adapting to situation dynamics in navigation, but the adaption and re-planning are constrained by the high-level plan so that mission requirements can still be satisfied.

If a robot is able to actively collect information for the purpose of learning the environment, it should be able to work autonomously without prior information. However, the robot may actively seek a human's help in execution. The language model could be extended so that it can generate language instructions from semantic information. The robot could then seek request information and help from a human coworker. Also the robot can either interpret natural and qualitative human-provided information or simplify big and complicated data into simple and human-natural representations. The robot would be able to actively learn and refine human intent, which will greatly benefit the interaction between human and robot.

References

[1] Julie A. Adams, Scott A. DeLoach, and Matthias Scheutz. Shared mental models for human-robot teams. In *2014 AAAI Fall Symposium Series*, 2014. URL `https://www.aaai.org/ocs/index.php/FSS/FSS14/paper/view/9109`.

[2] Vlada Aginsky, Catherine Harris, Ronald Rensink, and Jack Beusmans. Two strategies for learning a route in a driving simulator. *Journal of Environmental Psychology*, 17 (4):317–331, 1997.

[3] F. Ahmed and K. Deb. Multi-objective path planning using spline representation. In *Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on*, pages 1047–1052, Dec 2011. doi: 10.1109/ROBIO.2011.6181426.

[4] Faez Ahmed and Kalyanmoy Deb. Multi-objective optimal path planning using elitist non-dominated sorting genetic algorithms. *Soft Computing*, 17(7):1283–1299, 2013. ISSN 1432-7643. doi: 10.1007/s00500-012-0964-8. URL `http://dx.doi.org/10.1007/s00500-012-0964-8`.

[5] N. Ahmed and M. Campbell. Variational bayesian data fusion of multi-class discrete observations with applications to cooperative human-robot estimation. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 186–191, May 2010. doi: 10.1109/ROBOT.2010.5509521.

[6] Nisar Razzi Ahmed, Rina Tse, and Mark Campbell. Enabling robust human-robot cooperation through flexible fully bayesian shared sensing. In *2014 AAAI Spring Symposium Series*, 2014.

[7] N.R. Ahmed, E.M. Sample, and M. Campbell. Bayesian multicategorical soft data fusion for human-robot collaboration. *Robotics, IEEE Transactions on*, 29(1):189–206, Feb 2013. ISSN 1552-3098. doi: 10.1109/TRO.2012.2214556.

[8] Bonny Banerjee and B Chandrasekaran. A framework of voronoi diagram for planning multiple paths in free space. *Journal of Experimental & Theoretical Artificial Intelligence*, 25(4):457–475, 2013.

[9] Daniel J. Barber, Thomas M. Howard, and Matthew R. Walter. A multimodal interface for real-time soldier-robot teaming, 2016. URL http://dx.doi.org/10.1117/12.2224401.

[10] P. Bhattacharjee, P. Rakshit, I. Goswami, A. Konar, and A. K. Nagar. Multi-robot path-planning using artificial bee colony optimization algorithm. In *Nature and Biologically Inspired Computing (NaBIC), 2011 Third World Congress on*, pages 219–224, Oct 2011. doi: 10.1109/NaBIC.2011.6089601.

[11] S. Bhattacharya, M. Likhachev, and V. Kumar. Topological constraints in search-based robot path planning. *Autonomous Robots*, 33(3):273–290, 2012. ISSN 0929-5593. doi: 10.1007/s10514-012-9304-1.

[12] Subhrajit Bhattacharya. Search-based path planning with homotopy class constraints, 2010. URL https://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1920.

[13] Jonathan Binney and G. Sukhatme. Branch and bound for informative path planning. In *2012 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2147–2154, May 2012. doi: 10.1109/ICRA.2012.6224902.

[14] MohammadReza Bonyadi, Zbigniew Michalewicz, and Xiaodong Li. An analysis of the velocity updating rule of the particle swarm optimization algorithm. *Journal of Heuristics*, 20(4):417–452, 2014. ISSN 1381-1231. doi: 10.1007/s10732-014-9245-2. URL http://dx.doi.org/10.1007/s10732-014-9245-2.

[15] A. Boularias, F. Duvallet, J. Oh, and A. Stentz. Grounding spatial relations for outdoor robot navigation. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 1976–1982, May 2015. doi: 10.1109/ICRA.2015.7139457.

[16] J. M. Bradshaw, M. Sierhuis, A. Acquisti, P. Feltovich, R. Hoffman, R. Jeffers, D. Prescott, N. Suri, A. Uszok, and R. Van Hoof. Agent autonomy. In H. Hexmoor, R. Falcone, and C. Castelfranchi, editors, *Adjustable Autonomy and Human-Agent Teamwork in Practice: An Interim Report on Space Applications*. Kluwer, 2002.

[17] D. Bratton and J. Kennedy. Defining a standard for particle swarm optimization. In *Swarm Intelligence Symposium, 2007. SIS 2007. IEEE*, pages 120–127, April 2007. doi: 10.1109/SIS.2007.368035.

[18] Michael Brenner, Nick Hawes, John D Kelleher, and Jeremy L Wyatt. Mediating between qualitative and quantitative representations for task-orientated human-robot interaction. In *IJCAI*, pages 2072–2077, 2007.

[19] Gordon Briggs and Matthias Scheutz. Facilitating mental modeling in collaborative human-robot interaction through adverbial cues. In *Proceedings of the SIGDIAL 2011 Conference*, SIGDIAL '11, pages 239–247, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics. ISBN 978-1-937284-10-7. URL `http://dl.acm.org/citation.cfm?id=2132890.2132916`.

[20] O. Brock and O. Khatib. Real-time re-planning in high-dimensional configuration spaces using sets of homotopic paths. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 1, pages 550–555 vol.1, 2000. doi: 10.1109/ROBOT.2000.844111.

[21] Steve Burion. Human detection for robotic urban search and rescue. *Diploma Work*, 2004.

[22] J.L. Burke and R.R. Murphy. Human-robot interaction in usar technical search: two heads are better than one. In *Robot and Human Interactive Communication, 2004. ROMAN 2004. 13th IEEE International Workshop on*, pages 307 – 312, sept. 2004. doi: 10.1109/ROMAN.2004.1374778.

[23] J. Casper and R.R. Murphy. Human-robot interactions during the robot-assisted urban search and rescue response at the world trade center. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 33(3):367 –385, june 2003. ISSN 1083-4419. doi: 10.1109/TSMCB.2003.811794.

[24] C. Chekuri and M. Pal. A recursive greedy algorithm for walks in directed graphs. In *46th Annual IEEE Symposium on Foundations of Computer Science, 2005. FOCS 2005.*, pages 245–253, Oct 2005. doi: 10.1109/SFCS.2005.9.

[25] David L. Chen and Raymond J. Mooney. Learning to interpret natural language navigation instructions from observations. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence (AAAI-2011)*, pages 859–865, August 2011.

[26] Howie M Choset. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[27] Istvan Chung, Oron Propp, Matthew R Walter, and Thomas M Howard. On the performance of hierarchical distributed correspondence graphs for efficient symbol

grounding of robot instructions. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 5247–5252. IEEE, 2015.

[28] S. Clark and M.A. Goodrich. A hierarchical flight planner for sensor-driven uav missions. In *2013 IEEE RO-MAN*, pages 509–514, Aug 2013. doi: 10.1109/ROMAN.2013.6628555.

[29] ChristopherW. Cleghorn and AndriesP. Engelbrecht. A generalized theoretical deterministic particle swarm model. *Swarm Intelligence*, 8(1):35–59, 2014. ISSN 1935-3812. doi: 10.1007/s11721-013-0090-y. URL `http://dx.doi.org/10.1007/s11721-013-0090-y`.

[30] M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, 6(1):58–73, Feb 2002. ISSN 1089-778X. doi: 10.1109/4235.985692.

[31] Maurice Clerc and Edited Riccardo Poli. Stagnation analysis in particle swarm optimisation or what happens when nothing happens. Technical report, 2006.

[32] T.M. Cover and J.A. Thomas. *Elements of information theory*. Wiley-interscience, 2006.

[33] Indraneel Das and J. E. Dennis. Normal-boundary intersection: A new method for generating the pareto surface in nonlinear multicriteria optimization problems. *SIAM Journal on Optimization*, 8(3):631–657, 1998. doi: 10.1137/S1052623496307510. URL `http://dx.doi.org/10.1137/S1052623496307510`.

[34] K. Deb and H. Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. *Evolutionary Computation, IEEE Transactions on*, 18(4):577–601, Aug 2014. ISSN 1089-778X. doi: 10.1109/TEVC.2013.2281535.

[35] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, 2002.

[36] Douglas Demyen and Michael Buro. Efficient triangulation-based pathfinding. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 1*, AAAI'06, pages 942–947. AAAI Press, 2006. ISBN 978-1-57735-281-5.

[37] B. Doroodgar, M. Ficocelli, B. Mobedi, and G. Nejat. The search for survivors: Cooperative human-robot interaction in search and rescue environments using semi-autonomous robots. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2858 –2863, may 2010. doi: 10.1109/ROBOT.2010.5509530.

[38] Felix Duvallet, Matthew Walter, Thomas Howard, Sachithra Hemachandra, Jean Hyaejin Oh, Seth Teller, Nicholas Roy, and Anthony (Tony) Stentz. Inferring maps and behaviors from natural language instructions. In *International Symposium on Experimental Robotics*, June 2014.

[39] Alon Efrat, Stephen G. Kobourov, and Anna Lubiw. Computing homotopic shortest paths efficiently. *Computational Geometry*, 35(3):162 – 172, 2006. ISSN 0925-7721. doi: http://dx.doi.org/10.1016/j.comgeo.2006.03.003.

[40] J. Fasola and M.J. Mataric. Using semantic fields to model dynamic spatial relations in a robot architecture for natural language instruction of service robots. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 143–150, Nov 2013. doi: 10.1109/IROS.2013.6696345.

[41] J.L. Fernandez-Martinez and E. Garcia-Gonzalo. Stochastic stability analysis of the linear continuous and discrete pso models. *Evolutionary Computation, IEEE Transactions on*, 15(3):405–423, June 2011. ISSN 1089-778X. doi: 10.1109/TEVC.2010.2053935.

[42] MichaelA. Goodrich and Daqing Yi. Toward task-based mental models of human-robot teaming: A bayesian approach. In Randall Shumaker, editor, *Virtual Augmented and Mixed Reality. Designing and Developing Augmented and Virtual Environments*, volume 8021 of *Lecture Notes in Computer Science*, pages 267–276. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-39404-1. doi: 10.1007/978-3-642-39405-8_30. URL `http://dx.doi.org/10.1007/978-3-642-39405-8_30`.

[43] Jesse Gray and Cynthia Breazeal. Manipulating mental states through physical action. *International Journal of Social Robotics*, 6(3):315–327, 2014. ISSN 1875-4791. doi: 10.1007/s12369-014-0234-2. URL `http://dx.doi.org/10.1007/s12369-014-0234-2`.

[44] D. Grigoriev and A. Slissenko. Polytime algorithm for the shortest path in a homotopy class amidst semi-algebraic obstacles in the plane. In *Proceedings of the 1998 International Symposium on Symbolic and Algebraic Computation*, ISSAC '98, pages 17–24, New York, NY, USA, 1998. ACM. ISBN 1-58113-002-3. doi: 10.1145/281508.281528. URL `http://doi.acm.org/10.1145/281508.281528`.

[45] S. Guadarrama, L. Riano, D. Golland, D. Gouhring, Yangqing Jia, D. Klein, P. Abbeel, and T. Darrell. Grounding spatial relations for human-robot interaction. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 1640–1647, Nov 2013. doi: 10.1109/IROS.2013.6696569.

[46] Allen Hatcher. Algebraic topology. 2002. *Cambridge UP, Cambridge*, 606(9), 2002.

[47] E. Hernandez, M. Carreras, J. Antich, P. Ridao, and A. Ortiz. A topologically guided path planner for an AUV using homotopy classes. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2337–2343, May 2011. doi: 10.1109/ICRA.2011.5980108.

[48] Emili Hernandez, Marc Carreras, and Pere Ridao. A comparison of homotopic path planning algorithms for robotic applications. *Robotics and Autonomous Systems*, 64 (0):44 – 58, 2015. ISSN 0921-8890. doi: http://dx.doi.org/10.1016/j.robot.2014.10.021. URL `http://www.sciencedirect.com/science/article/pii/S0921889014002425`.

[49] John Hershberger and Jack Snoeyink. Computing minimum length paths of a given homotopy class. *Computational Geometry*, 4(2):63 – 97, 1994. ISSN 0925-7721. doi: http://dx.doi.org/10.1016/0925-7721(94)90010-8. URL `http://www.sciencedirect.com/science/article/pii/0925772194900108`.

[50] G. Hoffman and C. Breazeal. Collaboration in human-robot teams. In *Proc. of the AIAA 1st Intelligent Systems Technical Conference, Chicago, IL, USA*, 2004.

[51] Thomas M Howard, Stefanie Tellex, and Nicholas Roy. A natural language planner interface for mobile manipulators. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 6652–6659. IEEE, 2014.

[52] Jason K Howlett, Timothy W McLain, and Michael A Goodrich. Learning real-time a* path planner for unmanned air vehicle target sensing. *Journal of Aerospace Computing, Information, and Communication*, 3(3):108–122, 2006.

[53] Kevin D Jenkins. The shortest path problem in the plane with obstacles: A graph modeling approach to producing finite search lists of homotopy classes. Master's thesis, Naval Postgraduate School, Monterey, CA, June 1991.

[54] M. Jiang, Y.P. Luo, and S.Y. Yang. Stochastic convergence analysis and parameter selection of the standard particle swarm optimization algorithm. *Information Processing Letters*, 102(1):8 – 16, 2007. ISSN 0020-0190. doi: http://dx.doi.org/10.1016/j.ipl.2006.10.005. URL `http://www.sciencedirect.com/science/article/pii/S0020019006003103`.

[55] Zhong-Ping Jiang and Yuan Wang. Input-to-state stability for discrete-time nonlinear systems. *Automatica*, 37(6):857 – 869, 2001. ISSN 0005-1098. doi: http:

//dx.doi.org/10.1016/S0005-1098(01)00028-0. URL http://www.sciencedirect.com/science/article/pii/S0005109801000280.

[56] A. Jones, M. Schwager, and C. Belta. A receding horizon algorithm for informative path planning with temporal logic constraints. In *2013 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5019–5024, May 2013. doi: 10.1109/ICRA.2013.6631294.

[57] Catholijn M. Jonker, M. Birna van Riemsdijk, and Bas Vermeulen. Shared mental models: A conceptual analysis. In *Proceedings of the 6th International Conference on Coordination, Organizations, Institutions, and Norms in Agent Systems*, number 20 in COIN@AAMAS'10, pages 132–151, Berlin, Heidelberg, 2011. Springer-Verlag. URL http://dl.acm.org/citation.cfm?id=2018118.2018128.

[58] M. Jordan and A. Perez. Optimal bidirectional rapidly-exploring random trees. Technical Report MIT-CSAIL-TR-2013-021, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, August 2013. URL http://dspace.mit.edu/bitstream/handle/1721.1/79884/MIT-CSAIL-TR-2013-021.pdf.

[59] S. Karaman and E. Frazzoli. Incremental sampling-based algorithms for optimal motion planning. In *Robotics: Science and Systems (RSS)*, Zaragoza, Spain, June 2010.

[60] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. J. Rob. Res.*, 30(7):846–894, June 2011. ISSN 0278-3649. doi: 10.1177/0278364911406761. URL http://dx.doi.org/10.1177/0278364911406761.

[61] Hassan K Khalil and JW Grizzle. *Nonlinear systems*, volume 3. Prentice hall New Jersey, 1996.

[62] Soonkyum Kim, S. Bhattacharya, and V. Kumar. Path planning for a tethered mobile robot. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 1132–1139, May 2014. doi: 10.1109/ICRA.2014.6906996.

[63] T. Kollar, S. Tellex, D. Roy, and N. Roy. Toward understanding natural language directions. In *Human-Robot Interaction (HRI), 2010 5th ACM/IEEE International Conference on*, pages 259–266, March 2010. doi: 10.1109/HRI.2010.5453186.

[64] Andreas Krause and Daniel Golovin. Submodular function maximization. *Tractability: Practical Approaches to Hard Problems*, 3, 2012.

[65] J. J. Kuffner and S. M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 2, pages 995–1001 vol.2, 2000. doi: 10.1109/ROBOT.2000.844730.

[66] Benjamin Kuipers. The spatial semantic hierarchy. *Artificial Intelligence*, 119(12):191 – 233, 2000. ISSN 0004-3702. doi: http://dx.doi.org/10.1016/S0004-3702(00)00017-5. URL `http://www.sciencedirect.com/science/article/pii/S0004370200000175`.

[67] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[68] Christian Lebiere, Florian Jentsch, and Scott Ososky. Cognitive models of decision making processes for human-robot interaction. In Randall Shumaker, editor, *Virtual Augmented and Mixed Reality. Designing and Developing Augmented and Virtual Environments*, volume 8021 of *Lecture Notes in Computer Science*, pages 285–294. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-39404-1. doi: 10.1007/978-3-642-39405-8_32. URL `http://dx.doi.org/10.1007/978-3-642-39405-8_32`.

[69] Daniel S Levine. *Information-rich path planning under general constraints using rapidly-exploring random trees*. PhD thesis, MIT, 2010.

[70] L. Lin and M.A. Goodrich. A bayesian approach to modeling lost person behaviors based on terrain features in wilderness search and rescue. *Computational & Mathematical Organization Theory*, 16(3):300–323, 2010.

[71] Dong C. Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical Programming*, 45(1):503–528, 1989. ISSN 1436-4646. doi: 10.1007/BF01589116. URL `http://dx.doi.org/10.1007/BF01589116`.

[72] L. Mandow and J. L. P´erez De la Cruz. A new approach to multiobjective a* search. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, pages 218–223, San Francisco, CA, USA, 2005. Morgan Kaufmann Publishers Inc. URL `http://dl.acm.org/citation.cfm?id=1642293.1642328`.

[73] R Timothy Marler and Jasbir S Arora. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26(6):369–395, 2004.

[74] Maja J Mataric. Integration of representation into goal-driven behavior-based robots. *Robotics and Automation, IEEE Transactions on*, 8(3):304–312, 1992.

[75] John E Mathieu, Tonia S Heffner, Gerald F Goodwin, Eduardo Salas, and Janis A Cannon-Bowers. The influence of shared mental models on team process and performance. *Journal of Applied Psychology*, 85(2):273, 2000.

[76] Cynthia Matuszek, Evan Herbst, Luke Zettlemoyer, and Dieter Fox. Learning to parse natural language commands to a robot control system. In Jaydev P. Desai, Gregory Dudek, Oussama Khatib, and Vijay Kumar, editors, *Experimental Robotics*, volume 88 of *Springer Tracts in Advanced Robotics*, pages 403–415. Springer International Publishing, 2013. ISBN 978-3-319-00064-0. doi: 10.1007/978-3-319-00065-7_28.

[77] Mark McClelland, Mark Campbell, and Tara Estlin. Qualitative relational mapping for robotic navigation. *AIAA Infotech@ Aerospace*, 2012.

[78] Mark McClelland, Mark Campbell, and Tara Estlin. Qualitative relational mapping for mobile robots with minimal sensing. *Journal of Aerospace Information Systems*, 11(8): 497–511, 2014.

[79] J.R. Medina, M. Shelley, Dongheui Lee, W. Takano, and S. Hirche. Towards interactive physical robotic assistance: Parameterizing motion primitives through natural language. In *RO-MAN, 2012 IEEE*, pages 1097–1102, Sept 2012. doi: 10.1109/ROMAN.2012.6343895.

[80] Nimrod Megiddo, Eitan Zemel, and S Louis Hakimi. The maximum coverage location problem. *SIAM Journal on Algebraic Discrete Methods*, 4(2):253–261, 1983.

[81] Yongguo Mei, Yung-Hsiang Lu, Y Charlie Hu, and CS George Lee. Deployment strategy for mobile robots with energy and timing constraints. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 2816–2821. IEEE, 2005.

[82] Kaisa Miettinen. *Nonlinear multiobjective optimization*, volume 12. Springer, 1999.

[83] V. Narayanan, P. Vernaza, M. Likhachev, and S.M. LaValle. Planning under topological constraints using beam-graphs. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 431–437, May 2013. doi: 10.1109/ICRA.2013.6630611.

[84] M.A. Neerincx, T. de Greef, N.J.J.M. Smets, and M.P. Sam. Shared mental models of distributed human-robot teams for coordinated disaster responses. In *2011 AAAI Fall Symposium Series*, 2011.

[85] Stefanos Nikolaidis and Julie Shah. Human-robot teaming using shared mental models. *ACM/IEEE HRI*, 2012.

[86] Donald A Norman. *The design of everyday things: Revised and expanded edition*. Basic books, 2013.

[87] Illah R Nourbakhsh, Katia Sycara, Mary Koes, Mark Yong, Michael Lewis, and Steve Burion. Human-robot teaming for search and rescue. *Pervasive Computing, IEEE*, 4 (1):72–79, 2005.

[88] R. Parasuraman, T.B. Sheridan, and C.D. Wickens. A model for types and levels of human interaction with automation. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 30(3):286–297, 2000.

[89] E.J.Solteiro Pires, J.A.Tenreiro Machado, and P.B. de Moura Oliveira. Robot trajectory planning using multi-objective genetic algorithm optimization. In Kalyanmoy Deb, editor, *Genetic and Evolutionary Computation GECCO 2004*, volume 3102 of *Lecture Notes in Computer Science*, pages 615–626. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-22344-3. doi: 10.1007/978-3-540-24854-5_64. URL `http://dx.doi.org/10.1007/978-3-540-24854-5_64`.

[90] Florian T. Pokorny, Majd Hawasly, and Subramanian Ramamoorthy. Multiscale topological trajectory classification with persistent homology. In *Proceedings of Robotics: Science and Systems*, Berkeley, USA, July 2014.

[91] R. Poli. Mean and variance of the sampling distribution of particle swarm optimizers during stagnation. *Evolutionary Computation, IEEE Transactions on*, 13(4):712–721, Aug 2009. ISSN 1089-778X. doi: 10.1109/TEVC.2008.2011744.

[92] Riccardo Poli. Dynamics and stability of the sampling distribution of particle swarm optimisers via moment analysis. *J. Artif. Evol. App.*, 2008:15:1–15:10, January 2008. ISSN 1687-6229. doi: 10.1155/2008/761459. URL `http://dx.doi.org/10.1155/2008/761459`.

[93] Riccardo Poli and David Broomhead. Exact analysis of the sampling distribution for the canonical particle swarm optimiser and its convergence during stagnation. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, GECCO '07, pages 134–141, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-697-4. doi: 10.1145/1276958.1276977. URL `http://doi.acm.org/10.1145/1276958.1276977`.

[94] Yutao Qi, Xiaoliang Ma, Fang Liu, Licheng Jiao, Jianyong Sun, and Jianshe Wu. MOEA/D with adaptive weight adjustment. *Evol. Comput.*, 22(2):231–264, June 2014. ISSN 1063-6560. doi: 10.1162/EVCO_a_00109. URL `http://dx.doi.org/10.1162/EVCO_a_00109`.

[95] M. Roscheck and M.A. Goodrich. Detection likelihood maps for wilderness search and rescue. In *Systems, Man, and Cybernetics (SMC), 2012 IEEE International Conference on*, pages 327–332. IEEE, 2012.

[96] Kenneth Rosen. *Discrete Mathematics and Its Applications*. McGraw-Hill Science, 2011.

[97] N Ruangpayoongsak, H Roth, and J Chudoba. Mobile robots for search and rescue. In *Safety, Security and Rescue Robotics, Workshop, 2005 IEEE International*, pages 212–217. IEEE, 2005.

[98] E. Salas, S.M. Fiore, and M.P. Letsky. *Theories of team cognition: Cross-disciplinary perspectives*. Routledge Academic, 2011.

[99] N.R. Samal, A. Konar, S. Das, and A. Abraham. A closed loop stability analysis and parameter selection of the particle swarm optimization dynamics for faster convergence. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 1769–1776, Sept 2007. doi: 10.1109/CEC.2007.4424687.

[100] Manuel Schmitt and Rolf Wanka. Particle swarm optimization almost surely finds local optima. In *Proceeding of the Fifteenth Annual Conference on Genetic and Evolutionary Computation Conference*, GECCO '13, pages 1629–1636, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1963-8. doi: 10.1145/2463372.2463563. URL `http://doi.acm.org/10.1145/2463372.2463563`.

[101] E. Schmitzberger, J.L. Bouchet, M. Dufaut, D. Wolf, and R. Husson. Capture of homotopy classes with probabilistic road map. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 3, pages 2317–2322 vol.3, 2002. doi: 10.1109/IRDS.2002.1041613.

[102] Jason R Schott. Fault tolerant design using single and multicriteria genetic algorithm optimization. Technical report, DTIC Document, 1995.

[103] David Schuster and Florian Jentsch. Increasing robot autonomy effectively using the science of teams. In Randall Shumaker, editor, *Virtual Augmented and Mixed*

Reality. Designing and Developing Augmented and Virtual Environments, volume 8021 of *Lecture Notes in Computer Science*, pages 313–320. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-39404-1. doi: 10.1007/978-3-642-39405-8_35. URL http://dx.doi.org/10.1007/978-3-642-39405-8_35.

[104] Danelle C Shah and Mark E Campbell. A qualitative path planner for robot navigation using human-provided maps. *The International Journal of Robotics Research*, 32(13): 1517–1535, 2013.

[105] D.C. Shah and M.E. Campbell. A robust qualitative planner for mobile robot navigation using human-provided maps. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 2580–2585, May 2011. doi: 10.1109/ICRA.2011.5980331.

[106] Thomas B Sheridan and William L Verplank. Human and computer control of undersea teleoperators. Technical report, DTIC Document, 1978.

[107] Pradyumn Kumar Shukla. *On the Normal Boundary Intersection Method for Generation of Efficient Front*, pages 310–317. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-72584-8. doi: 10.1007/978-3-540-72584-8_40. URL http://dx.doi.org/10.1007/978-3-540-72584-8_40.

[108] Amarjeet Singh, Andreas Krause, Carlos Guestrin, and William J. Kaiser. Efficient informative sensing using multiple robots. *J. Artif. Int. Res.*, 34(1):707–755, April 2009. ISSN 1076-9757.

[109] A.R. Soltani, H. Tawfik, J.Y. Goulermas, and T. Fernando. Path planning in construction sites: performance evaluation of the dijkstra, a, and {GA} search algorithms. *Advanced Engineering Informatics*, 16(4):291 – 303, 2002. ISSN 1474-0346. doi: http://dx.doi.org/10.1016/S1474-0346(03)00018-1. URL http://www.sciencedirect.com/science/article/pii/S1474034603000181.

[110] J Starek, Edward Schmerling, Lucas Janson, and Marco Pavone. Bidirectional fast marching trees: An optimal sampling-based algorithm for bidirectional motion planning. In *Workshop on Algorithmic Foundations of Robotics*, 2014.

[111] P.B. Sujit and R. Beard. Multiple uav path planning using anytime algorithms. In *American Control Conference, 2009. ACC '09.*, pages 2978–2983, June 2009. doi: 10.1109/ACC.2009.5160222.

[112] Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew R Walter, Ashis Gopal Banerjee, Seth J Teller, and Nicholas Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *AAAI*, 2011.

[113] S. Thrun, W. Burgard, D. Fox, et al. *Probabilistic robotics*, volume 1. MIT press Cambridge, MA, 2005.

[114] Sebastian Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71, 1998.

[115] Ioan Cristian Trelea. The particle swarm optimization algorithm: convergence analysis and parameter selection. *Information Processing Letters*, 85(6):317 – 325, 2003. ISSN 0020-0190. doi: http://dx.doi.org/10.1016/S0020-0190(02)00447-7. URL `http://www.sciencedirect.com/science/article/pii/S0020019002004477`.

[116] F. van den Bergh and A.P. Engelbrecht. A study of particle swarm optimization particle trajectories. *Information Sciences*, 176(8):937 – 971, 2006. ISSN 0020-0255. doi: http://dx.doi.org/10.1016/j.ins.2005.02.003. URL `http://www.sciencedirect.com/science/article/pii/S0020025505000630`.

[117] Frans van den Bergh and Andries Petrus Engelbrecht. A convergence proof for the particle swarm optimiser. *Fundam. Inf.*, 105(4):341–374, December 2010. ISSN 0169-2968. URL `http://dl.acm.org/citation.cfm?id=2010420.2010421`.

[118] Pieter Vansteenwegen, Wouter Souffriau, Dirk Van Oudheusden, Pieter Vansteenwegen, Wouter Souffriau, and Dirk Van Oudheusden. The orienteering problem: A survey. *European Journal of Operational Research*, 209(1):1 – 10, 2011. ISSN 0377-2217. doi: http://dx.doi.org/10.1016/j.ejor.2010.03.045. URL `http://www.sciencedirect.com/science/article/pii/S0377221710002973`.

[119] Paul Vernaza, Venkatraman Narayanan, and Maxim Likhachev. Efficiently finding optimal winding-constrained loops in the plane. In *Proceedings of Robotics: Science and Systems*, Sydney, Australia, July 2012.

[120] Matthew R Walter, Sachithra Hemachandra, Bianca Homberg, Stefanie Tellex, and Seth Teller. A framework for learning semantic maps from grounded natural language descriptions. *The International Journal of Robotics Research*, 33(9):1167–1190, 2014.

[121] J. Gregory Trafton William G. Kennedy. Using simulations to model shared mental models. In *Proceedings of the 8th International Conference on Cognitive Modeling*, pages 253–254. Taylor & Francis/ Psychology Press, 2007.

[122] John Yen, Xiaocong Fan, Shuang Sun, Rui Wang, Cong Chen, and Kaivan Kamali. Implementing shared mental models for collaborative teamwork. In *In the Workshop on Collaboration Agents: Autonomous Agents for Collaborative Environments in the IEEE/WIC Intelligent Agent Technology Conference(2003) Halifax, Canda.115-126*, 2003.

[123] Daqing Yi and Michael A Goodrich. Supporting task-oriented collaboration in human-robot teams using semantic-based path planning. In *Proc. SPIE 9084, Unmanned Systems Technology XVI*, volume 9084, 2014.

[124] Daqing Yi, M.A. Goodrich, and K.D. Seppi. Informative path planning with a human path constraint. In *Systems, Man and Cybernetics (SMC), 2014 IEEE International Conference on*, pages 1752–1758, Oct 2014. doi: 10.1109/SMC.2014.6974170.

[125] Daqing Yi, Michael A Goodrich, and Kevin D Seppi. MORRF*: sampling-based multi-objective motion planning. In *Proceedings of the 24th International Conference on Artificial Intelligence. AAAI Press*, pages 1733–1739, 2015.

[126] Daqing Yi, Kevin D Seppi, and Michael A Goodrich. Input-to-state stability analysis on particle swarm optimization. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 81–88. ACM, 2015.

[127] Daqing Yi, Michael A. Goodrich, Thomas M Howard, and Kevin D. Seppi. Topology-aware RRT* for parallel optimal sampling in topologies. In *Systems, Man and Cybernetics (SMC), 2016 IEEE International Conference on*, Oct 2016.

[128] Daqing Yi, Michael A. Goodrich, and Kevin D. Seppi. Homotopy-aware RRT* : Toward human-robot topological path-planning. In *Proceedings of the Eleventh Annual ACM/IEEE International Conference on Human-Robot Interaction*, HRI '16, 2016.

[129] Daqing Yi, Thomas M. Howard, Michael A. Goodrich, and Kevin D. Seppi. Expressing homotopic requirements for mobile robot navigation through natural language instructions. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2016.

[130] Qingfu Zhang and Hui Li. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *Evolutionary Computation, IEEE Transactions on*, 11(6):712–731, Dec 2007. ISSN 1089-778X. doi: 10.1109/TEVC.2007.892759.

[131] Yong Zhang, Dun wei Gong, and Jian hua Zhang. Robot path planning in uncertain environment using multi-objective particle swarm optimization. *Neurocomputing*, 103:

172 – 185, 2013. ISSN 0925-2312. doi: http://dx.doi.org/10.1016/j.neucom.2012.09.019. URL http://www.sciencedirect.com/science/article/pii/S0925231212007722.

[132] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evol. Comput.*, 8(2):173–195, June 2000. ISSN 1063-6560. doi: 10.1162/106365600568202. URL http://dx.doi.org/10.1162/106365600568202.

Appendix A

Input-to-State Stability Analysis on Particle Swarm Optimization [1]

Abstract

This paper examines the dynamics of particle swarm optimization (PSO) by modeling PSO as a feedback cascade system and then applying input-to-state stability analysis. Using a feedback cascade system model we can include the effects of the global-best and personal-best values more directly in the model of the dynamics. Thus in contrast to previous study of PSO dynamics, the input-to-state stability property used here allows for the analysis of PSO both before and at stagnation. In addition, the use of input-to-state stability allows this analysis to preserve random terms which were heretofore simplified to constants. This analysis is important because it can inform the setting of PSO parameters and better characterize the nature of PSO as a dynamic system. This work also illuminates the way in which the personal-best and the global-best updates influence the bound on the particle's position and hence, how the algorithm exploits and explores the fitness landscape as a function of the personal best and global best.

[1]Published in GECCO '15 Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation. Authors are Daqing Yi, Kevin D. Seppi and Michael A. Goodrich.

A.1 Introduction

Particle Swarm Optimization (PSO) is a popular and well-studied algorithm that was originally motivated by the flocking behaviors of birds and insects. Soon after its first publication it was discovered that the structure of the PSO algorithm is amenable to formal analysis using dynamical systems theory (sometimes referred to as dynamic systems) [30]. The use of this theory has informed the setting of parameters [54, 115], led to the proposal of new variants of the algorithm [30], and allowed for the analysis of the behavior of the algorithm [100], especially the behavior at stagnation, that is, when the algorithm fails to find better solutions [30].

While the study of the algorithm at stagnation is important and a significant first step, it only answers questions about the behavior at the point that PSO has degenerated into random search. At that point the algorithm can be mimicked by simply sampling from the appropriate distribution [91]. In this paper we extend the limited work that has been done to understand the behavior *before* stagnation, that is, when the unique mechanisms of PSO are directing the behavior of the algorithm.

By using a *feedback cascade model* we are able to include both what we refer to as the *position update* which comes from the PSO equations, but also the *input update*, that is, the effect of the personal and global best. This paper does so in contrast to prior work which focuses on the position update. A cascade model also allows us to make fewer assumptions in mapping from PSO to a dynamical system model. Using this model we are able to derive the conditions under which the process is input-to-state stable [55], prove bounds on both the particle motion and the mean of particle motion. The input-to-state stable conditions and the bounds can inform parameter adjustments and other properties that can, in turn, control the extent to which the algorithm explores or exploits the fitness landscape. This is especially valuable in the context of the design of future PSO variants.

The body of this paper is organized as described here. In section A.3, we model the PSO dynamics as a feedback cascade system, which enables the input-to-state stability analysis. The definition of input-to-state stability (ISS) and its meaning in the context of PSO are also given. Section A.4 shows that for particular parameter values, the position-update component of a particle is input-to-state stable. Using the ISS property we then give the bound on particle motion. We also use the ISS property in the context of the analysis of the moments (the mean and higher moments) of particle motion. In section A.5, we use the ISS property to help analyze the dynamics of the particle. Using the ISS property of the input-update component, we can analyze the dynamics of the particle before and at stagnation.

A.2 Related Work

Although the input-to-state stable analysis given in this paper can be applied to many versions of PSO, for this work we use the formulas from Kennedy's most recent definition of PSO[17] for the constricted position-update rule. The constricted position-update rule is

$$\begin{aligned} v_{ij}(k+1) = \chi[v_{ij}(k) + \phi^P u_{ij}^P(k)(x_{ij}^P(k) - x_{ij}(k)) \\ + \phi^G u_{ij}^G(k)(x_{ij}^G(k) - x_{ij}(k))], \end{aligned} \tag{A.1a}$$

$$x_{ij}(k+1) = x_{ij}(k) + v_{ij}(k+1). \tag{A.1b}$$

$x_{ij}(k)$ represents the position of particle i in dimension j at time k. Similarly, $v_{ij}(k)$ represents the velocity of particle i in dimension j also at time k. $x_{ij}^G(k)$ and $x_{ij}^P(k)$ are global best (actually the topology best or local best) and personal best positions observed by the swarm and the particle respectively. $u_{ij}^G(k)$ and $u_{ij}^P(k)$ are independent random values drawn from $[0, 1]$. $\chi \in (0, 1)$, ϕ^P and ϕ^G are algorithm parameters.

Due to the stochastic nature of the particle's path and the social interaction represented by the topology, the dynamics of the algorithm is hard to evaluate in general. Understanding how the particles move guides how to improve the algorithm design [14], particularly the stochastic factors in the velocity update of Equation (A.1a). However once a particle is no longer able to find improvements in x^G and x^P, it exhibits the *stagnation phenomenon* [31]. In this state the analysis is easier since there is no effect from the topology.

Previous work that assumes stagnation can be categorized into two groups each based on how the analysis treats the stochastic factors. The first approach is to ignore the stochastic factors. Using this simplification, the convergence of a particle at stagnation can be analyzed [29, 30]. The convergence trajectories can be estimated [116] and the conclusions are compared with empirical results [29]. By building a linear system model [99], the PSO algorithm can be viewed as a closed loop system and the convergence can be analyzed. Based on such a convergence analysis, parameters can be set for best effect [115].

The second approach for handling the stochastic factors is based on stochastic analysis. By taking the mean of the stochastic variables, the stochastic terms can be converted into constant terms. A convergence analysis of the mean and variance of a particle at stagnation can also be obtained by using the characteristic equation in a discrete-time model [54]. In a similar way, other moments can be computed [91–93]. Using the discrete-time system model of different moments, the equilibrium can be found. The stability requirements can be obtained from the norm by setting the root values of the characteristic equation to all be less than 1.

There is also some work that addresses the dynamics when a particle is not in the stagnation phase. The discrete-time dynamics of PSO, that is, the dynamics of particle trajectory, can be approximated using a continuous-time model [41]. Furthermore, the probability of convergence in time can be analyzed by viewing the update process as a random search process [117]. The process of particles reaching a local optimum has also been analyzed [100].

A.3 Input-to-state stability of PSO

Input-to-state stability analysis consists of two parts:

- the decomposition of the PSO algorithm into components, and
- the input-to-state stability of each component.

In this section, we model particle motion using a feedback cascade model. Then we review the definition of the input-to-state stability (ISS). We also explain why ISS should be applied to PSO.

A.3.1 Feedback Cascade Model

For the purpose of input-to-state stability analysis we decompose the PSO algorithm into components as shown in Figure A.1. This decomposition is comprised of cascaded components (the input update, followed by the position update) and the feedback of the historical state. These two components are the *input-update component* for the global best ($x_i^G(k)$) and the personal best ($x_i^P(k)$), and the *position-update component* for particle position ($x_i(k+1)$), which depends on the inputs $x_i^G(k)$ and $x_i^P(k)$ as well as the previous position $x_i(k)$.

The properties of this system can be analyzed using the input-to-state stability of the position-update component and the input-update component. Given an input-to-state stable position-update component, we will see that the convergence of $x_i(k)$ depends on bounds on $x_i^G(k)$ and $x_i^P(k)$.

A.3.2 Input-to-state stability

Before reviewing the definition of input-to-state stability, we first introduce several types of functions [55].

- K-function $\mathbb{K}$: a function $\alpha : [0, a) \to [0, \infty)$ is continuous, strictly increasing and $\alpha(0) = 0$.

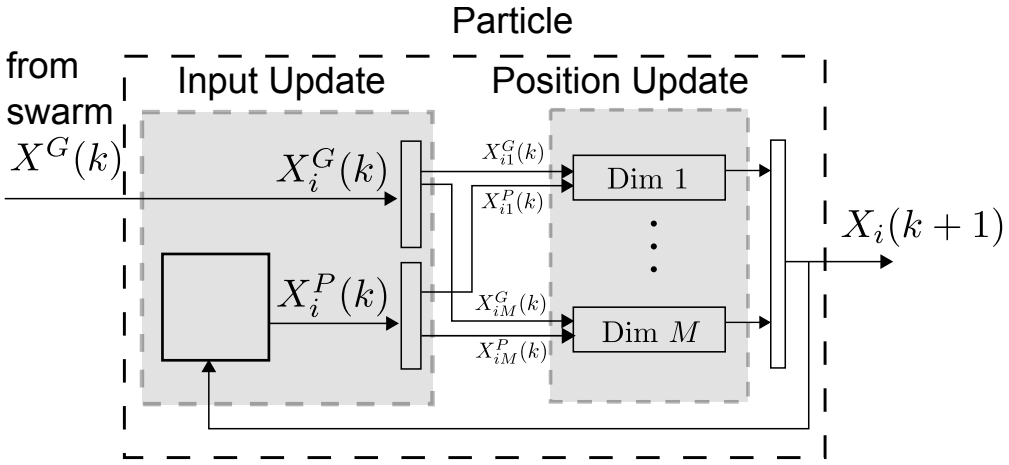


Figure A.1: System structure of PSO

- K_∞-function $\mathbb{K}_\infty$: a function $\alpha : [0, a) \to [0, \infty)$ is a K-function and $\alpha(s) \to \infty$ as $s \to \infty$.
- KL-function $\mathbb{KL}$: a function $\beta : [0, a) \times [0, \infty) \to [0, \infty)$ satisfies:

  1. $\forall t \geq 0$, $\beta(\cdot, t)$ is a K-function;
  2. $\forall s \geq 0$, $\beta(s, \cdot)$ is decreasing and $\beta(s, t) \to 0$ as $t \to \infty$.

  These functions are used to define input-to-state stability in Definition 1.

Definition 1 (Input-to-state stable [55])**.** *For x, a discrete-time system defined as follows:*

$$x(k+1) = f(x(k), u(k)), \tag{A.2}$$

with $f(0, 0) = 0$ [2], the system is (globally) input-to-state stable *if there exist a KL-function β and a K-function γ such that, for each input $u \in l_\infty^m$ and each $\xi \in \mathbb{R}^n$, it holds that $\forall k \in \mathbb{Z}^+$,*

$$|x(k, \xi, u)| \leq \beta(|\xi|, k) + \gamma(\|u\|). \tag{A.3}$$

The $\beta()$ term in Equation (A.3) defines an initial bound with a decaying property. The $\gamma()$ term in Equation (A.3) defines a bound determined by the input. This means that the $\beta()$ term gradually decreases to zero and the position is bounded by a range determined by the bound on the input.

[2]This means that $x = 0$ is an equilibrium of the 0-input system.

A.3.3 Importance of input-to-state stability

Under certain conditions the dynamics of complex systems can be understood by first decomposing the system into a set of individual input-to-state stable components. We will take this approach with PSO. Parallel combination of input-to-state stable components yields a combined structure that is also input-to-state stable [61]. In the case of PSO and as shown in Figure A.1, if each component (representing a single dimension) is input-to-state stable, the position-update component which combines all the dimensions is also input-to-state stable. Thus we have Property 1.

Property 1. *The position-update component is input-to-state stable if the position update in each dimension is input-to-state stable.*

This simplifies the analysis of the system since it allows us to consider each dimension separately. The serial connection and the feedback connection also lead to some interesting property from input-to-state stability, which will be discussed in Section A.5.

The input-to-state stability analysis also provides the tool for the analysis of the convergence of PSO and the analysis of bounds on particle motion. PSO is designed to strike an effective balance between *exploring* and *exploiting* a fitness landscape. A bound on a particle's state is an indicator of the nature of that balance. When this bound is large the particle is exploring. However, as a particle finishes exploring and reach stagnation, a particle's position should converge.

Input-to-state stability implies that the state of the system is bounded in a range determined by the bounds on the input. Before stagnation, when the personal best and global best values have not converged, we can expect only a loose bound on the particle state. These looser bounds reflect both what is know about the update process itself and what is know about the inputs to the update process, that is, the personal best and the global best.

We call the bounds on the global best and personal best the "exploit radius" and the bounds on the particle's position a "explore radius". The ratio of the explore radius to the exploit radius is determined by the parameters of the position-update component. However, if the personal best and global best converge to an estimated optimal position, the exploit radius falls to zero and the explore radius converges to a bound.

A.4 Analysis of Input-to-state stability in PSO

We then show PSO satisfies the definition of input-to-state stability when the parameters of PSO are set in a requisite range. We also derive the bounds implied by the ISS property and

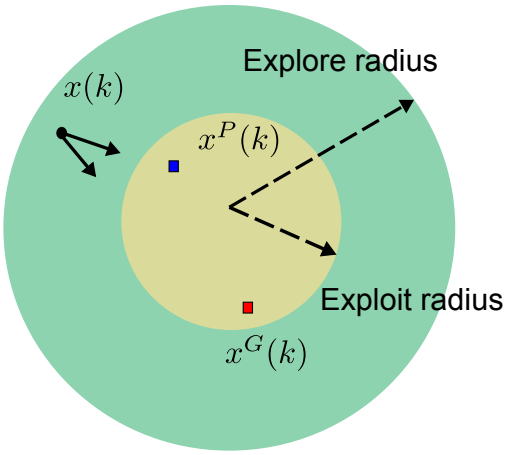


Figure A.2: Exploration and exploitation.

use the ISS property in Section A.5 to find bounds on particle motion. Last, we will use ISS to analyze the moments (the mean and higher moments) of particle motion.

In our analysis of the PSO algorithm, we seek to understand how the particles converge to some position x^*, which is intended (not guaranteed) by the algorithm to be the global minimum position of the objective function.

For this analysis we use a one-dimension particle and extract the linear form of the position-update component. As noted above, the one dimensional case can be extended to many dimensions.

We begin our analysis of PSO input to state stability by rewriting the PSO equations in (A.1) in the following way:

$$X(k+1) = A(k)X(k) + B(k)U(k) \tag{A.4}$$

with

$$A(k) = \begin{bmatrix} \chi & -\chi\phi^G u^G(k) - \chi\phi^P u^P(k) \\ \chi & 1 - \chi\phi^G u^G(k) - \chi\phi^P u^P(k) \end{bmatrix}$$

and

$$B(k) = \begin{bmatrix} \chi\phi^G u^G(k) & \chi\phi^P u^P(k) \\ \chi\phi^G u^G(k) & \chi\phi^P u^P(k) \end{bmatrix}.$$

The system state is $X(k) = [v(k),\ x(k) - x^*]^T$, and the system input is $U(k) = [x^G(k) - x^*,\ x^P(k) - x^*]^T$ [3]. The convergence of this model means that $v(k) \to 0$ and $x(k) \to x^*$.

[3]We use x^* to represent an equilibrium point to the system. In PSO, it can be a local optimum, a global optimum, or an estimated optimum. We use it as a reference point to check the bounds.

A.4.1 Conditions for input-to-state stability for position update in PSO

Using the definition of the PSO position update as given in Equation (A.4), PSO can be shown to be input-to-state stable as defined in definition 1.

Theorem 1. *When* $|\lambda_{\max}(A(k))| < 1$*, the position-update component of PSO* (A.4) *is input-to-state stable.*

The proof follows the ISS-Lyapunov-function approach. The ISS-Lyapunov function, defined in Appendix A.6, can be used to prove the input-to-state stability of a system and analyze the state bound[55]. The details of the proof are given in Appendix A.6.

Note that in Equation (A.4), $[v(k),\ x(k) - x^*]^T = [0,\ 0]^T$ is an equilibrium position when the input $[x^G(k) - x^*,\ x^P(k) - x^*]^T = [0,\ 0]^T$. Without loss of generality, for an arbitrary optimization problem x^* would typically not be at the origin. In such a problem, input-to-state stability means that the boundaries of $|v(k)|$ and $|x(k) - x^*|$ would be transformed and thus determined by $|x^G(k) - x^*|$ and $|x^P(k) - x^*|$, but the properties of ISS apply independent of where the function is centered.

Having shown that PSO is input-to-state stable we can now state a bound on particle position.

Corollary 1. *Given a bound on the input* $||u||$ *in the position-update component, we have the bound on the particle position from Equation* (A.4)*.*

$$\begin{aligned} &\forall k, |x(k) - x^*| \leq \\ &\quad \max\left(|x(0) - x^*|, \gamma(|\left[x^G(k) - x^*,\ x^P(k) - x^*\right]^T|)\right), \end{aligned} \tag{A.5}$$

in which $\gamma = \alpha_3^{-1} \circ \sigma$*. (*$\alpha_3$ *and* σ *are defined in Appendix A.6.)*

Proof. This is obtained from Remark 3.7 in [55] and by choosing P be a symmetric identity matrix. Furthermore we drop the velocity part because $|x(k) - x^*| \leq |[v(k),\ x(k) - x^*]^T|$. □

The max part is needed to account for the effect of the starting point, represented by the first parameter. Eventually the effect of the starting point no longer affects the system, formally:

$$\exists T, \forall k \geq T, |x(k) - x^*| \leq \gamma(|[x^G(k) - x^*,\ x^P(k) - x^*]^T|). \tag{A.6}$$

Figure A.3 gives an example on how a particle's boundary is determined by the personal best and global best.

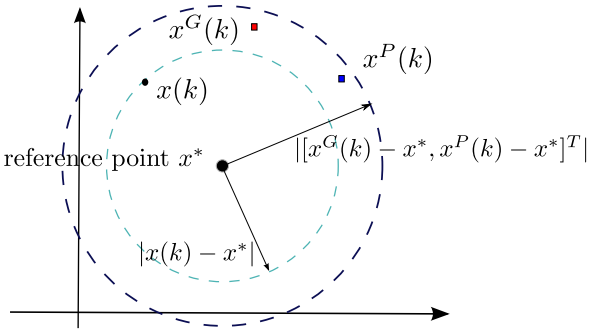


Figure A.3: A bound on a particle's position by a reference point x^* from Equation (A.6). The ratio of two radii indicates γ.

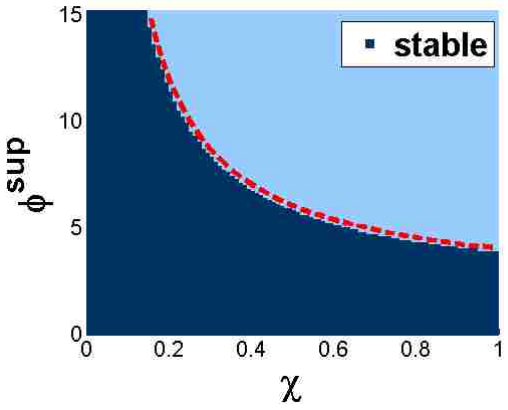


Figure A.4: Parameter space

Corollary 2. *Let* $A(k) = \begin{bmatrix} \chi & -\chi\phi \\ \chi & 1-\chi\phi \end{bmatrix}$, *in which* $\phi \in [0, \phi^{sup}]$, $\phi^{sup} = \phi^P + \phi^G$ *and* $\chi \in (0,1)$. *When* $\phi^{sup} \in \left(0, \frac{2(1+\chi)}{\chi}\right)$, *the system* (A.4) *is input-to-state stable.*

The proof is given in Appendix A.6.

Figure A.4 shows the parameter space. The x-axis is $\phi^{sup} = \phi^P + \phi^G$ and the y-axis is χ. The stable region in dark blue is obtained from eigenvalue test on Theorem 1. The red boundary is obtained from Corollary 2. It indicates the equivalence of the results from Theorem 1 and Corollary 2.

A.4.2 Moment Analysis

Using the same perspective of the feedback cascade system, the input-to-stable stability analysis can also be applied to moment analysis. Like the others [54, 92], we derive models for the statistical features (moments) of the particle's position at stagnation. In contrast to

this prior work, input-to-state stability analysis can also provide bounds before a particle reaches stagnation.

We adopt the approach of Jiang, Luo & Yang [54] to construct an ISS model for the mean. Using that model $E(x(k))$ converges to

$$\hat{x} = \frac{\phi^P x^P + \phi^G x^G}{\phi^P + \phi^G}$$

in stagnation. If we treat $\hat{x}$ as a swarm average estimation on the optimum, we are interested in how $E(x(k))$ deviates from $\hat{x}$.

$$\begin{aligned} \begin{bmatrix} E(x(k+1)) - \hat{x} \\ E(x(k)) - \hat{x} \end{bmatrix} &= A_m \begin{bmatrix} E(x(k)) - \hat{x} \\ E(x(k-1)) - \hat{x} \end{bmatrix} \\ &\quad + B_m \begin{bmatrix} E(x^P(k)) - \hat{x} \\ E(x^G(k)) - \hat{x} \end{bmatrix}, \end{aligned} \tag{A.7}$$

with

$$A_m = \begin{bmatrix} 1 + \chi - \frac{\chi\phi^P}{2} - \frac{\chi\phi^G}{2} & -\chi \\ 1 & 0 \end{bmatrix}$$

and

$$B_m = \begin{bmatrix} \frac{\chi\phi^P}{2} & \frac{\chi\phi^G}{2} \\ 0 & 0 \end{bmatrix}.$$

The convergence of $E(x(k))$ in stagnation is given in [54, 92]. Even without the stagnation assumption, the input-to-state stable analysis on Equation (A.7) indicates how $E(x(k))$ will deviate from $\hat{x}$ anytime we know how $E(x^G(k))$ and $E(x^P(k))$ deviate from $\hat{x}$. Stagnation is a simple case of knowing how $E(x^G(k))$ and $E(x^P(k))$ deviate from $\hat{x}$. This special case is discussed more later in this paper.

We now proceed to show the conditions that must hold for the mean model to be input-to-state stable.

Theorem 2. *The system* (A.7) *is input-to-state stable, if* $|\lambda_{\max}(A_m)| < 1$.

Proof. The proof process is similar with Theorem 1, but we can get a constant symmetric positive definite Q_m from $A_m^T P A_m - P = -Q_m$. □

Similar to Corollary 2, we have Corollary 3 for parameter selection on mean convergence. Note that when the condition in Corollary 2 is satisfied, the condition in Corollary 3 is also guaranteed. This means that when the system (A.4) is input-to-state stable, the mean dynamics (A.7) is also input-to-state stable.

Corollary 3. *Let* $A_m = \begin{bmatrix} 1+\chi-\frac{\chi\phi^P}{2}-\frac{\chi\phi^G}{2} & -\chi \\ 1 & 0 \end{bmatrix}$, *in which* $\phi \in [0, \phi^{sup}]$ *and* $\phi^{sup} = \phi^P + \phi^G$ *and* $\chi \in (0,1)$. *When* $\phi^{sup} \in \left(0, \frac{4(1+\chi)}{\chi}\right)$, *the system* (A.7) *is input-to-state stable.*

The proof is given in Appendix A.6.

Similar to Corollary 1, we can use the Q_m to determine the state bound.

Corollary 4. *If the system* (A.7) *is input-to-state stable, we have a bound*

$$\begin{aligned} &\exists T, \forall k > T, \\ &|E(x(k)) - \hat{x}| \leq \gamma_m | \left[E(x^P(k)) - \hat{x},\ E(x^G(k)) - \hat{x}\right]^T |, \end{aligned} \tag{A.8}$$

with

$$\gamma_m = \frac{2\|A_m\|^2\|B_m\|^2 + \lambda_{min}(Q_m)^2\|B_m\|^2}{2(\lambda_{min}(Q_m))^3}. \tag{A.9}$$

In a similar way, we can apply the input-to-state stability analysis to the variance model [54] and higher order moment models [93].

A.5 Implications of particle ISS

In this section, we add the input-update component that was first shown in Figure A.1 and then analyze the convergence of particle position.

Since by Theorem 1 PSO is input-to-state stable, and therefore by Corollary 1 the stability of the cascade system depends on the output of the input-update component. We can say:

1. If the input-update component generates converging personal best and global best, the bound of the particle position will converge;
2. If the personal best and global best vary within a bound, the particle will converge within a bound;
3. If the personal best and global best become constant, the particle will converge within a bound.
4. If the personal best and global best are constant and the same, the particle will converge toward the global best.

By Theorem 2 and 4, we can make similar statements about the particle mean. As well, this boundary analysis could be applied to higher moments.

Furthermore, by Equation (A.5), we know that the convergence of a particle's position $x(k)$ to x^* depends on how $x^P(k)$ and $x^G(k)$ converge to x^* when the position-update

component is input-to-state stable. In particular, the bound on the distance between a particle's position and x^* is determined by the initial distance $x(0) - x^*$, $x^P(k) - x^*$ and $x^G(k) - x^*$.

A.5.1 Stagnation

Since stagnation is defined as a state where a particle fails to find better positions, in stagnation $x^P(k)$ and $x^G(k)$ are constant in k, and can thus be represented as x^P and x^G respectively. If we assume that $u^P(k)$ and $u^G(k)$ are equivalent and constant, it can be stated that

$$\hat{x} = \frac{\phi^P x^P + \phi^G x^G}{\phi^P + \phi^G} \tag{A.10}$$

is an equilibrium point for stagnation as noted in previous work [30].

By Theorem 2, in stagnation the mean of the position will converge. Corollary 1 describes a bound on position as a function of the PSO parameters. Similarly, assuming that parameters are chosen that will also lead to the convergence of higher moments similar to previous work [54, 93], the pattern of particle movement at stagnation could be simulated by a distribution constructed to be consistent with PSO moment information[93].

By letting $x^* = \hat{x}$ be the reference point, and by Corollary 1, we can go beyond prior work and can identify a bound on PSO behavior at stagnation:

$$\exists T, \forall k > T, |x(k) - \hat{x}| \leq \gamma_d |[x^P - \hat{x},\ x^G - \hat{x}]^T|, \tag{A.11}$$

with

$$\gamma_d = \frac{2\|A'\|^2\|B'\|^2 + \lambda_{min}(Q'))^2\|B'\|^2}{2(\lambda_{min}(Q'))^3}. \tag{A.12}$$

Particularly, when $x^P = x^G$, we have $\hat{x} = x^G = x^P$. By Equation (A.11) we know that $\exists T, x(T) = x^G$, which means $x(k) \to x^G$. Thus we have shown the convergence of PSO in stagnation without treating the random terms as constants required by the work described in Section A.2.

A.5.2 Before stagnation

Input-to-state stable analysis also supports understanding the cases before stagnation. When the $x^P(k)$ and $x^G(k)$ are not constant, the system state depends on the property of the input-update component. The personal-best update is

$$x_i^P(k) = \arg \max_{x \in \{x_i(k), x_i^P(k-1)\}} f(x). \tag{A.13}$$

The global-best update is

$$x_i^G(k) = \arg \max_{x \in \{x_i(k), x_i^G(k-1)\}} f(x). \tag{A.14}$$

As in Figure A.1, there exists a feedback cascade system structure for a particle. If we assume that there is another model for swarm information sharing, which implements the $x^G(k)$ update. Then in the particle, the $x^G(k)$ can be viewed as an input that is independent with the current particle state. The feedback loop uses the current state to update the $x^P(k)$. As in Equation (A.13) and (A.14), the input-to-state stability of the input-update component depends on $f(x)$, which indicates that the input-to-state stability relies on the shape of the fitness distribution. Assuming that the PSO parameters are set such that the position-update component is input-to-state stable, there are three cases in analyzing the dynamics of a particle.

- *When only the $x^G(k)$ is constant*
  This happens usually when a "good" global best is found. Thus the swarm stops finding better global bests. The input from the swarm can be modeled as a constant factor of the system. However, the particle still finds new personal best positions and updates the personal best. The system dynamics of the particle is determined by the feedback state, which updates the personal best. In this process, $f(x^P) < f(x^G)$, otherwise, x^G will be updated. In this case, the two components form a feedback loop structure. The small gain theorem [55] can be used. If the multiplication of the gain factors of two components is less than 1, the system will still converge.
- *When only the $x^P(k)$ is constant*
  This usually means that this particle is "stuck" in exploring a local region but some other particles are continuously finding new better position. Thus the global best is being updated. In this case, the feedback of the system does not impact the input-update component and thus nor does it impact the position-update component. In this case the system model can be simplified by ignoring the feedback. As a result, the system falls into only a serial cascade system because there is no feedback for this particle. If the input-update component is input-to-state stable, the serial connection of two input-to-state stable component is still input-to-state stable (it can be shown that a serial cascade of ISS components is also input-to-state stable [61]). Since this serial cascade system is input-to-state stable, the new position will be bounded somewhere around $x^G(k)$. This implies that the particle will converge toward the $x^G(k)$. If we have $f(x^G) > f(x^P)$, in the assumption of the continuity in the fitness space, when the

particle gets closer to the global best, there exits some region that $f(x) > f(x^P)$. The x^P will start to change.

- *When both the $x^G(k)$ and $x^P(k)$ are not constant*
  This usually means that both the particle and the swarm it belongs to keep on finding better positions. Thus both the particle updates the personal best and the swarm updates the global best. In this case, the input-to-state stability of the input-update component is harder to guarantee. However, if the change of $x^G(k)$ and $x^P(k)$ is bounded, the movement of the particle is still bounded by Corollary 1.

Generally, in the PSO, the dynamics of a particle switches in between these cases before eventually reaching the stagnation. Understanding the dynamics of the particles before stagnation supports the exploration and exploitation capability of particles in optimal search.

A.6 Conclusion

In this paper, we have decomposed the particle in the PSO algorithm into a feedback cascade model, which consists of input-update and position-update components. We introduce the input-to-state stability analysis to the position-update component. For an input-to-state stable position-update component, if the input to this component is bounded, the state is bounded; also if the input to the component converges, the state converges. The convergence of a particle in PSO is determined by the output of the input-update component, which are the personal best and global best. If they are in stagnation, the particle converges.

The analysis of a cascade structure used here can be applied to a wide range of PSO variants. In the cases that the same position-update component but different input-update components are used, the convergence and the boundary of the particles are determined by whether the input-update component generates converging or bounded personal best and global best. For variants that use a different position-update component, the ISS properties would need to be verified.

The ISS property of the input-update component depends on the fitness distribution. We provide several scenarios for the dynamics of the particle. We show that the optimal search process switches among these scenarios and how the input-to-state stability analysis should be applied into different scenarios.

Appendix

ISS-Lyapunov function

Using the definitions of a K-function and a KL-function in Section A.3 we can define an ISS-Lyapunov function as follows, an ISS-Lyapunov function $V : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ satisfies:

1. $\exists \alpha_1, \alpha_2 \in \mathbb{K}$ such that $\forall \xi \in \mathbb{R}^n$, $\alpha_1(|\xi|) \leq V(\xi) \leq \alpha_2(|\xi|)$.
2. $\exists \alpha_3 \in \mathbb{K}_\infty, \sigma \in \mathbb{K}$ such that $\forall \xi \in \mathbb{R}^n, \forall \mu \in \mathbb{R}^m$,$V(f(\xi, \mu)) - V(\xi) \leq -\alpha_3(|\xi|) + \sigma(|\mu|)$.

Proof of Theorem 1

Proof. Let P be an identity matrix. As $|\lambda_{\max}(A(k))| < 1$, we have $\|A^T(k)PA(k)\| \leq \|P\|\|A(k)\|^2 \leq \|P\||\lambda_{\max}(A(k))|^2 < \|P\|$. Because P is an identity matrix it is positive definite, and thus $A^T(k)PA(k)$ is positive definite or positive semi-definite by definition. So by positive definite ordering we have $A^T(k)PA(k) < P$.

Let $-Q(k) = A^T(k)PA(k) - P$. Since $A^T(k)PA(k) < P$ then $-Q(k) < 0$ furthermore $\exists Q' \forall k, Q(k) > Q' > 0$.

By the Lemma 3.5 in [55], if we can show that a proposed positive definite Lyapunov function is an ISS-Lyapunov function, the system is input-to-state stable.

Define a Lyapunov function

$$V(X(k)) = X^T(k)PX(k). \tag{A.15}$$

We can have $\lambda_{min}(P)|X(k)|^2 \leq V(X(k)) \leq \lambda_{max}(P)|X(k)|^2$ and $\lambda_{min}(P) = \lambda_{max}(P)$.

Let $\alpha_1(\xi) = \lambda_{min}\xi^2$ and $\alpha_2(\xi) = \lambda_{max}\xi^2$, we have $V(x)$ satisfying condition 1 of the ISS-Lyapunov function definition.

By applying Equation (A.4) to $V(X(k+1)) - V(X(k))$, we have

$$\begin{aligned}
&V(X(k+1)) - V(X(k)) \\
=&[X^T(k)A^T(k) + U^T(k)B^T(k)]P[A(k)X(k) + B(k)U(k)] \\
&- X^T(k)PX(k) \\
=&X^T(k)A^T(k)PA(k)X(k) + X^T(k)A^T(k)PB(k)U(k) \\
&+ U^T(k)B^T(k)PA(k)X(k) + U^T(k)B^T(k)PB(k)U(k) \\
&- X^T(k)PX(k)
\end{aligned} \tag{A.16}$$

As P is identity matrix, it is symmetric, thus

$$[X^T(k)A^T(k)PB(k)U(k)]^T = U^T(k)B^T(k)PA(k)X(k). \tag{A.17}$$

$V(X(k+1)), V(X(k)) \in \mathbb{R}$, we have $X^T(k)A^T(k)PB(k)U(k)$ and $U^T(k)B^T(k)PA(k)X(k)$ are both real value (like 1×1 matrix). Thus,

$$X^T(k)A^T(k)PB(k)U(k) = U^T(k)B^T(k)PA(k)X(k). \tag{A.18}$$

We then have

$$\begin{aligned}
& V(X(k+1)) - V(X(k)) \\
=& - X^T(k)[A^T(k)PA(k) - P]X(k) \\
& + U^T(k)B^T(k)PB(k)U(k) \\
& + 2X^T(k)A^T(k)PB(k)U(k) \\
\leq& - X^T(k)Q'X(k) + U^T(k)B^T(k)PB(k)U(k) \\
& + 2X^T(k)A^T(k)PB(k)U(k)
\end{aligned} \tag{A.19}$$

By applying matrix norm, we have

$$\begin{aligned}
& V(X(k+1)) - V(X(k)) \\
\leq& - \lambda_{min}(Q')|X(k)|^2 + |B^T(k)PB(k)||U(k)|^2 \\
& + 2|A^T(k)PB(k)||U(k)||X(k)| \\
=& - \frac{1}{2}\lambda_{min}(Q')|X(k)|^2 + |B^T(k)PB(k)||U(k)|^2 \\
& - \frac{1}{2}\lambda_{min}(Q')|X(k)|^2 + 2|A^T(k)PB(k)||U(k)||X(k)| \\
=& - \frac{1}{2}\lambda_{min}(Q')|X(k)|^2 \\
& + \left(\frac{2|A^T(k)PB(k)|^2}{(\lambda_{min}(Q'))^2} + |B^T(k)PB(k)|\right)|U(k)|^2 \\
& - \frac{1}{2}\lambda_{min}(Q')[|X(k)|^2 - \frac{4|A^T(k)PB(k)|}{\lambda_{min}(Q')}|X(k)||U(k)| \\
& + \frac{4|A^T(k)PB(k)|^2}{(\lambda_{min}(Q'))^2}|U(k)|^2]
\end{aligned} \tag{A.20}$$

By completing the square, we have

$$\begin{aligned}
&V(X(k+1)) - V(X(k)) \\
\leq &-\frac{1}{2}\lambda_{min}(Q')|X(k)|^2 \\
&+\left(\frac{2|A^T(k)PB(k)|^2}{(\lambda_{min}(Q'))^2} + |B^T(k)PB(k)|\right)|U(k)|^2 \\
&-\frac{1}{2}\lambda_{min}(Q')\left(|X(k)| - \frac{2|A^T(k)PB(k)|}{\lambda_{min}(Q')}|U(k)|\right)^2 \\
\leq &-\frac{1}{2}\lambda_{min}(Q')|X(k)|^2 \\
&+\left(\frac{2\|A^T(k)PB(k)\|^2}{(\lambda_{min}(Q'))^2} + \|B^T(k)PB(k)\|\right)|U(k)|^2.
\end{aligned} \tag{A.21}$$

Because $u^P(k) \in [0,1]$, there exist an A' and B' such that $\|A(k)\| \leq \|A'\|$ and $\|B(k)\| \leq \|B'\|$. We have $\|A^T(k)PB(k)\| \leq \|A'\|\|P\|\|B'\|$ and $\|B^T(k)PB(k)\| \leq \|P\|\|B'\|^2$.

Since the identity matrix P has $||P|| = 1$:

$$\begin{aligned}
&V(X(k+1)) - V(X(k)) \\
\leq &-\frac{1}{2}\lambda_{min}(Q')|X(k)|^2 + \left(\frac{2\|A'\|^2\|B'\|^2}{(\lambda_{min}(Q'))^2} + \|B'\|^2\right)|U(k)|^2.
\end{aligned} \tag{A.22}$$

Let

$$\alpha_3(\xi) = \frac{1}{2}\lambda_{min}(Q')\xi^2,$$

and

$$\sigma(\xi) = \left(\frac{2\|A'\|^2\|B'\|^2}{(\lambda_{min}(Q'))^2} + \|B'\|^2\right)\xi^2.$$

Thus we have $V(X(k+1)) - V(X(k))$ satisfying condition 2 of the ISS-Lyapunov function definition and so (A.15) is an ISS-Lyapunov function. Using Jiang's Lemma 3.5[55], the position-update component of PSO (Equation (A.4)) is input-to-state stable. □

Proof of Corollary 2

Proof. Let $a = (1+\chi) - \chi\phi$. The eigenvalues of $A(k)$ are

$$\lambda = \frac{a \pm \sqrt{a^2 - 4\chi}}{2}.$$

There can be two cases.

1. If $a^2 \geq 4\chi$, the eigenvalues are real number. We have $a \geq 2\sqrt{\chi}$ or $a \leq -2\sqrt{\chi}$.

If $a \geq 2\sqrt{\chi}$, then $|\lambda_{\max}| < 1$ derives

$$0 < \frac{a - \sqrt{a^2 - 4\chi}}{2} \leq \frac{a + \sqrt{a^2 - 4\chi}}{2} < 1.$$

It means that $2\sqrt{\chi} \leq a < 1 + \chi$.

If $a \leq 2\sqrt{\chi}$, then $|\lambda_{\max}| < 1$ derives

$$-1 < \frac{a - \sqrt{a^2 - 4\chi}}{2} \leq \frac{a + \sqrt{a^2 - 4\chi}}{2} < 0.$$

It means that $-(\chi + 1) < a \leq -2\sqrt{\chi}$.

2. If $a^2 < 4\chi$, the eigenvalues are complex number. We have $-2\sqrt{\chi} < a < 2\sqrt{\chi}$.

   $|\lambda_{\max}| < 1$ derives

   $$\frac{a^2}{4} + \frac{a^2 - 4\chi}{4} < 1.$$

   It means that $-2\sqrt{2(1+\chi)} < a < 2\sqrt{2(1+\chi)}$. Because $\sqrt{2(1+\chi)} > 2\sqrt{\chi}$, we have $-2\sqrt{\chi} < a < 2\sqrt{\chi}$.

Combining these two cases, we have $-(1 + \chi) < a < 1 + \chi$. It equals to $\phi \in \left(0, \frac{2(1+\chi)}{\chi}\right)$. □

Proof of Corollary 3

Proof. The proof is similar with that in Subsection A.6. In this case, $a = (1+\chi) - \frac{\phi}{2}\chi$. Similarly, we can have two cases and derive $-(1 + \chi) < a < 1 + \chi$. It equals to $\phi \in \left(0, \frac{4(1+\chi)}{\chi}\right)$. □