Mission and Motion Planning for Multi-robot Systems in Constrained Environments

by

Kangjin Kim

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved November 2019 by the
Graduate Supervisory Committee:

Georgios Fainekos, Chair
Chitta Baral
Joohyung Lee
Spring Berman

ARIZONA STATE UNIVERSITY

December 2019

ABSTRACT

As robots become mechanically more capable, they are going to be more and more integrated into our daily lives. Over time, human's expectation of what the robot capabilities are is getting higher. Therefore, it can be conjectured that often robots will not act as human commanders intended them to do. That is, the users of the robots may have a different point of view from the one the robots do.

The first part of this dissertation covers methods that resolve some instances of this mismatch when the mission requirements are expressed in Linear Temporal Logic (LTL) for handling coverage, sequencing, conditions and avoidance. That is, the following general questions are addressed:

- What cause of the given mission is unrealizable?
- Is there any other feasible mission that is close to the given one?

In order to answer these questions, the LTL Revision Problem is applied and it is formulated as a graph search problem. It is shown that in general the problem is NP-Complete. Hence, it is proved that the heuristic algorihtm has 2-approximation bound in some cases. This problem, then, is extended to two different versions: one is for the weighted transition system and another is for the specification under quantitative preference. Next, a follow up question is addressed:

- How can an LTL specified mission be scaled up to multiple robots operating in confined environments?

The Cooperative Multi-agent Planning Problem is addressed by borrowing a technique from cooperative pathfinding problems in discrete grid environments. Since centralized planning for multi-robot systems is computationally challenging and easily results in state space explosion, a distributed planning approach is provided through agent coupling and de-coupling.

In addition, in order to make such robot missions work in the real world, robots should take actions in the continuous physical world. Hence, in the second part of this thesis, the resulting motion planning problems is addressed for non-holonomic robots.

That is, it is devoted to autonomous vehicles' motion planning in challenging environments such as rural, semi-structured roads. This planning problem is solved with an on-the-fly hierarchical approach, using a pre-computed lattice planner. It is also proved that the proposed algorithm guarantees resolution-completeness in such demanding environments. Finally, possible extensions are discussed.

ACKNOWLEDGMENTS

I thank Professor Georgios Fainekos for being my research advisor. With his warm support and interest to my race, he leads me to this finish line of Ph.D. which was the running course of which the distance is unknown and where the finish line is also unrecognizable. I could see him at every water station, cheering me up. He was sometimes my pacemaker, helping me to control my breath properly. Yes, he motivated me whenever I passed my minute mile.

I remembered Shih-Kai Su yelling me that ”Almost done!” in the beginning of the race. The ”Dirty Video” we filmed with Hengyi Yang is still there. I could see Shashank Srinivas, Ramtin Kermani and Parth Pandya coming back to me at the 6 mile turning point. Yashwanth Annapureddy controlled the traffic as a volunteer of the race.

While following the course, I tried to chase Houssam Abbas, but when he sped up at the uphill, I missed him. It was okay because I still had Bardh Hoxha. We could run together, enjoying the nice view of the Sedona.

After the 2 mile point, Adel Dokanchi kept asking me how much left. Cumhur Erkan Tuncali was busy all the time talking through his phone call. It surprised me that Indranil Sur did not stop by water stations. Joe Cambell must be an angle. He was serving pancakes in the middle of the course. William Duong also helped Joe, putting toppings like maple syrup.

I understand Shakiva Yagoubi walked the rest of the course. Sai Krishna Bashetty drunk too much water. Rahul Thekkalore Srinivasa was out of shape while Xiaotong Zhang ran steadily. Wei Wei looked sleepy. Maybe his days and nights are mixed up. Mohammad Hekmatnejad looked happier because his wife also joined the race, too. Keyvan Majd provided me a cup of energy drink.

Yu Zhang took my photographs in the hot spot. In the water stations, Chitta

Baral gave me an orange, Joohyung Lee handed me an energy gel and Spring Berman gave me a banana.

Once I passed the finish line, I found my wife Soyoon Lee waiting for me after finishing her 6 mile race. Sang-Su Lee and Sung An gave me a finisher medal. Yeong-Hwan Tscha served me a bagel and Jennifer Wong gave me a protein shake. Then, finally my father Heesoo Kim and my mother Sanghee Yoon helped me find my drop bag.

This research was partially funded by NSF awards 1116136, 1350420, 1446730, 1361926 and the NSF I/UCRC Center for Embedded Systems. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, or the Center for Embedded Systems.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Page

LIST OF ALGORITHMS

Chapter 1

INTRODUCTION

1.1 Mission Planning

As robots become mechanically more capable, they are going to be more and more integrated into our daily lives. Non-expert users will have to communicate with the robots in a natural language setting and request a robot or a team of robots to accomplish complicated tasks. Therefore, we need methods that can capture the high-level user requirements, solve the planning problem and map the solution to low level continuous control actions. In addition, such frameworks must come with mathematical guarantees of safe and correct operation for the whole system and not just the high level planning or the low level continuous control.

Linear Temporal Logic (LTL) (see [1]) can provide the mathematical framework that can bridge the gap between

1. natural language and high-level planning algorithms (e.g., [2, 3]), and
2. high-level planning algorithms and control (e.g., [4–8]).

LTL has been utilized as a specification language in a wide range of robotics applications. Authors in [9] provide a recent survey and for a good coverage of the related research directions, the reader is referred to [4–8, 10–16] and the references therein. For instance, in [4], the authors present a framework for motion planning of a single mobile robot with second order dynamics. The problem of reactive planning and distributed controller synthesis for multiple robots is presented in [10] for a fragment of LTL (Generalized Reactivity 1 (GR1)). The authors in [7] present a method for incremental planning when the specifications are provided in the GR1 fragment of LTL. The papers [11, 13] address the problem of centralized control of multiple robots where the specifications are provided as LTL formulas. An application of LTL planning methods to humanoid robot dancing is presented in [15]. In [5], the authors convert the LTL planning problem into Mixed Integer Linear Programming (MILP) or Mixed Integer Quadratic Programming (MIQP) problems. The use of sampling-based methods for solving the LTL motion planning problem is explored in [6]. All the previous applications assume that the robots are autonomous agents with full control over their actions. An interesting different approach is taken in [12] where the agents move uncontrollably in the environment and the controller opens and closes gates in the environment.

From all the previous methods, we can have two questions. First, what if we relax the assumption that the LTL planning problem has a feasible solution? In real-life scenarios, it is to be expected that not all complex task requirements can be realized by a robot or a team of robots. In such failure cases, the robot needs to provide feedback to the non-expert user on why the specification failed. Furthermore, it would be desirable that the robot proposes a number of plans that can be realized by the robot and which are as "close" as possible to the initial user intent. Then, the user would be able to understand what are the limitations of the robot and, also, he/she would be able to choose among a number of possible feasible plans. In [17], the author made the first steps towards solving the debugging (i.e., why the planning failed) and revision (i.e., what the robot can actually do) problems for automata theoretic LTL planning ([18]).

Another important challenge is how to scale LTL mission planning methods to multiple robots. There is a lot of work discussing multi-agent problems under LTL mission specifications, e.g., [19–26]. However, if we focus on the number of robots in the existing temporal logic planning methods, we can see that they are typically applicable only to groups of 3 to 5 robots. In order to resolve this scalability issue for multi-agent LTL planning, we will focus on a specific subproblem not addressed in the literature. Namely, we will focus on the specific classes of LTL problems where the missions of each robot can be planned and executed independently. On the other hand, we will consider some other challenging assumptions like limited communication range and highly confined environments. Our solution builds upon methods from Cooperative Pathfinding from [27–29].

1.2 Motion Planning

Motion planning for autonomous vehicles has been actively studied for several decades now [30–32]. There are, however, still remaining issues in order to truly deploy autonomous vehicles in the wild. Autonomous vehicles should cooperate with other human drivers, motorcycles and pedestrians including bike riders and obey different traffic rules, operating in different road conditions and environments. Among them, this thesis focuses on vehicles operating in unstructured road environments, i.e., rural road networks with narrow roads and steep turns (see Fig. 1.1).

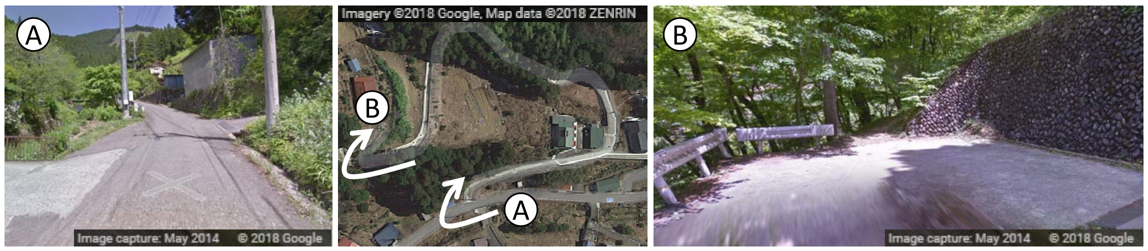

A
Image capture: May 2014 © 2018 Google
Imagery ©2018 Google, Map data ©2018 ZENRIN
B
A
B
Image capture: May 2014 © 2018 Google

Figure 1.1: Steep turns on rural roads that typically require complex maneuvers. A: the road width is approximately 1.5 cars; B: the road fits only one vehicle.

Such routes require complex motion planning maneuvers which must be computed in near real time. Clearly, no human passenger would be willing to use a vehicle that stays idle for 4-5 min while computing a feasible motion plan.

In order to archieve this challenging goal, this thesis utilizes a hierarchical approach for the planning [33–35]. The high level planner finds the path and generates a sequence of way points on the path. The low level planner generates a motion plan to follow the way points.

1.3 Literature Review

1.3.1 LTL Revision Problem

A related research problem is query checking [36, 37]. In query checking, given a model of the system and a temporal logic formula ϕ, some subformulas in ϕ are replaced with placeholders. Then, the problem is to determine a set of Boolean formulas such that if these formulas are placed into the placeholders, then ϕ holds on the model. The problem of revision as defined here is substantially different from query checking. For one, the user does not know where to position the placeholders in the formula when the planning fails.

The papers [38, 39] present a related problem. It is the problem of revising a system model such that it satisfies a temporal logic specification. Along the same lines, one can study the problem of maximally permissive controllers for automata specification [40]. Note that in this section, we are trying to solve the opposite problem, i.e., we are trying to relax the specification such that it can be realized on the system. The main motivation for our work is that the model of the system, i.e., the environment and the system dynamics, cannot be modified and, therefore, we need to understand what can be achieved with the current constraints.

Finding out why a specification is not satisfiable on a model is a problem that is very related to the problems of *vacuity* and *coverage* in model checking [41]. Another related problem is the detection of the causes of unrealizability in LTL games. In this case, a number of heuristics have been developed in order to localize the error and provide meaningful information to the user for debugging [42, 43]. Along these lines, LTLMop [44] was developed to debug unrealizable LTL specifications in reactive planning for robotic applications. In the paper [45], the authors provided an integrated system for non-expert users to control robots for high-level, reactive tasks through natural language. This system gives the user natural language feedback when the original intention is unsatisfiable. Given a unrealizable specification, the paper [46] characterizes unachievable cores, such as deadlock and livelock, from the specification. Then, it tries to find minimal cores of the unrealizable specification, providing it to the designer as a feedback. In [47], the authors investigated situations in which a planner-based agent cannot find a solution for a given planning task. They provided a formalization of coming up with "excuses" for not being able to find a plan and determined the computational complexity of finding excuses. On the practical side, they presented a method that is able to find good excuses in robotic application domains.

Over-Subscription Planning (OSP) [48] and Partial Satisfaction Planning (PSP) [49] are also very related problems. OSP finds an appropriate subset of an over-subscribed, conjunctive goal to meet the limitation of time and energy consumption. PSP explains the planning problem where the goal is regarded as soft constraints and trying to find a good quality plan for a subset of the goals. OSP and PSP have almost same definition, but there is also a difference. OSP regards the resource limitations as an important factor of partial goal to be satisfied, while PSP chooses a trade-off between the total action costs and the goal utilities where handling the plan quality.

Another related problem is the Minimum Constraint Removal problem (MCR) [50]. MCR concentrates on finding the smallest set of violated geometric constraints so that satisfaction in the specification can be achieved.

For the cases when the given specification is not satisfied, the authors in [51] introduce a non-monotonic temporal logic (N-LTL). This provides ways to elaborate the given specification for the revision. However, as a prerequisite, it needs the classifications of weak and strong exceptions for certain parts of the specification. On the other hand, the approach in this thesis does not require this classification. It revises an unsatisfiable specification by synthesizing it with the system and searching the relaxed, product graph in an algorithmic manner. In addition, it can return the specific atomic propositions which are not satisfiable.

The authors in [52] consider a number of high-level requirements in LTL which not all can be satisfied on the system. Each formula that is satisfied gains some reward. The goal of their algorithm is to maximize the rewards and, thus, maximize the number of requirements that can be satisfied on the system. Our problem definition is similar in spirit, but the problem goals are substantially different and the two approaches can be viewed as complementary. In [52], if a whole sub-specification cannot be realized, then it is aborted. In our case, we try to minimally revise the sub-specification so that it can be partially satisfied. Another substantial difference is that our proposed solutions can be incorporated directly within the control synthesis algorithm. Namely, as the algorithm searches for a satisfiable plan, it also creates the graph where the search for the revision will take place. In [52], the graph to be used for the revision must be constructed as a separate step. The problem of LTL planning with qualitative preferences has been studied in [53, 54] (see also the references therein for more research in this direction). As opposed to revision problem, planning with preferences is based on the fact that there are many satisfiable plans and, thus, the most preferable one should be selected.

1.3.2 Multi-agent Pathfinding Problem

Cooperative Pathfinding, a problem of computing non-conflicting paths for multiple mobile robots, can be addressed in one of two ways.

Firstly, in fully coupled approach, there are state-search algorithms such as A*. In [55], the authors introduced a technique to reduce the branching factors in order to resolve performance issue which occurs by its intractable nature. In [56], the authors provided how to connect the multi-agent planning problem to the network flow problem. However, this approach is unscalable due to the large state space.

Secondly, in the decoupled approach, [57] introduced the Hierarchical Cooperative A* (HCA*). This is a prioritized planning approach which applies a reservation table in order to respect the computed plans for robots of higher priorities. [57] also provided a Windowed HCA* approach (WHCA*) which limits the influence of the computed plans based on robots' current locations and within a fixed window size to access the reservation table. [58] extended this work, providing conflict-based reservation table (CO-WHCA*). All these decoupled approaches can find the solutions, but they are incomplete. In [28, 29], the authors provided completeness by relaxing optimality. However, their approaches depend on the availability of global information. For instance, they assume that each robot can always access individual plans for all robots. In a distributed system which has limited sensing and communication range, this information can not be accessed very often by all the other robots.

1.3.3 Motion Planning

For unknown semi-structured environments, [59] shows a robotic vehicle operation. Focusing on parking lot environments, they first get a solution with hybrid-state A*, and then they improve the solution quality through non-linear optimization. Their demonstration was conducted in relatively open spaces.

For generating way points, [34, 60] use similar approaches with ours. Authors in [34] focus on path smoothing before guiding to way points while we focus on sampling some of way points when we choose the local goals for planning the motion. Authors in [60] used a combined heuristic cost for their A* search in the unstructured environment. Their cost function combines the kinematic constraints of the vehicle with the Voronoi graph of the free space. We, on the other hand, use the mid path which is from the decomposed cells, not from the Voronoi graph.

Our hierarchical framework generates a local motion plan through lattice sets, following the guide lines of the path. This following method is similar with [34] and [60]. The method used in [34] adds mid points to the road map data, and then smooths the line which is a sequence of mid points before following it. The approach used in [60] is to build a Voronoi graph of the free space. Then, they use the mid points of the graph as a heuristic cost for their motion planning through A* search which also considers the kinematic constraints of the vehicle. On the other hands, we first decompose the roads and connects the mid points of the path which is a sequence of the decomposed road segments. Then, after sampling some of the points to target them as local goals, we generates motion plans through lattice sets similar to [61, 62].

To enable near real time motion planning for autonomous vehicles, we propose a hierarchical framework using lattice sets. Our method first decomposes the roads into segments and then it connects the mid points of each segment to form a road map. Next, it samples some of the points as local goals and it generates motion plans through lattice sets similar to [61, 62]. When our hierarchical framework generates a local motion plan through lattice sets, we compute the motion plan in order to follow the guide lines of the path. At a high level, our framework partially adapts techniques from [34, 60]. In detail, [34] adds mid points to the road map data, and then smoothens the resulting path, which is a sequence of mid points, before following it. Similarly, [60] builds a Voronoi graph of the free space. Then, it uses the mid points of the graph as a heuristic cost for their motion planning through A* search which also considers the kinematic constraints of the vehicle. Our line following method is similar; however, our end goal is to enable motions that require complex vehicle maneuvers.

When constructing lattice sets, we primarily follow the results in [61, 62]. The work in [62] regularly samples lattice sets in order to expand the search space. The approach is efficient for traveling in relatively open spaces, but it can fail in heavily constrained environments. In order to plan for steep turns and narrow passages, we choose densely covered lattice sets. In addition, [62] also proposes to build on-line the search graph while conducting A* search. In this process, a Heuristic Lookup Table (HLUT) is utilized which was introduced in [61]. However, when planning in a constraint environment, building the search graph on-line often leads to significant performance issues due to the frequent collision checks with the environment. Hence, we choose not to build the search graph on-line, but we search using pre-computed KD-Trees. In order to reduce planning time, [61, 62] manage the level of fidelity by varying the resolution of the grid for the lattice set (*graduated fidelity*). However, this is also only works for relatively open spaces. When following narrow and stiff curves, we have to consider the driving speed. Hence, our multi-resolution approach not only changes the level of fidelity, but also changes the level of velocity. This is close to the approach in [63] and the Adaptive Dimensionality (AD) in [64].

Among the works which are the closest related to ours are [65] and [66]. In [65], the authors developed a motion planning approach for narrow environments through the so-called RTR+TTS planner. Their approach does not consider optimal trajectories, but focuses on human like driving by reducing the number of cusps. The work in

[67] developed a similar approach for tight environments such as high density parking lots. First, they use HCS for the steering function of RRT*. Then, they tried to optimize the trajectory. In [66], the authors improved their previous work in [67] by introducing the maximum curvature, maximum curvature rate and maximum curvature acceleration. Their approach can generate the path in almost real time, but it is still needed to be improved. For example, their experiment results show that there are some failures even though RRT* provides probabilistic completeness. The experimental results in [67] show that their approach can generate paths in almost real time. However, it also shows that there are some failures. Since they ran the experiments in high density parking lots, failures may be unavoidable, but potentially they could avoid the failures if they relaxed the probabilistic completeness to resolution completeness.

1.4 Summary of Contributions

In this dissertation, we cover two major topics. In the rest of this chapter, a summary of contributions and publications of these topics are presented along with a reading guide.

Mission Planning When we specify robot missions, a formally structured language is required to be used. This language should be logically precise and consider the time for particular tasks to be done. In this setting, Linear Temporal Logic (LTL) is a good choice. Even though this logical formula can make the robot missions succeeded, when human designers use this framework, they can make some mistakes. Hence, it is necessary to introduce LTL Revision Problems.

- [68] ***Kangjin Kim***, *Georgios Fainekos and Sriram Sankaranarayanan,* ***On the Revision Problem of Specification Automata****, IEEE International Conference on Robotics and Automation, St. Paul, Minnesota, May 2012*

- [69] ***Kangjin Kim*** *and Georgios Fainekos,* ***Approximate Solutions for the Minimal Revision Problem of Specification Automata****, IEEE/RSJ International Conference on Intelligent Robots and Systems, Vilamoura Algarve, Portugal, Oct. 2012*

- [70] ***Kangjin Kim*** *and Georgios Fainekos,* ***Minimal Specification Revision for Weighted Transition Systems****, IEEE International Conference on Robotics and Automation, Karlsruhe, Germany, May 2013*

- [71] ***Kangjin Kim*** *and Georgios Fainekos,* ***Revision of Specification Automata under Quantitative Preferences****, IEEE International Conference on Robotics and Automation, Hong-Kong, June 2014*

- [72] ***Kangjin Kim****, Georgios Fainekos, and Sriram Sankaranarayanan,* ***On the Minimal Revision Problem of Specification Automata****, International Journal of Robotics Research (IJRR), 2015*

From the above work, parts of unsatisfiable robot missions can be indicated and corrected, guiding the human designers to fix the issue. In order for many more robots to be applied in this framework, we first investigated cooperative pathfinding. Along these lines, we developed the DisCoF framework:

- [27] *Yu Zhang,* ***Kangjin Kim*** *and Georgios Fainekos,* ***DisCoF: Cooperative Pathfinding in Distributed Systems with Limited Sensing and Communication Range****, International Symposium on Distributed Autonomous Robotic Systems, Daejeon, Korea, Nov 2014*

- [73] ***Kangjin Kim****, J. Campbell, W. Duong, Yu Zhang, Georgios Fainekos,* ***DisCoF+: Asynchronous DisCoF with flexible decoupling for cooperative pathfinding in distributed systems****, IEEE International Conference* *on Automation Science and Engineering (IEEE CASE), Gothenberg, Sweden, Aug 2015*

These works provide distributed framework while computing the path and resolving any intrinsic conflict among some agents. In a discrete world, such as a grid system, this approach returns a solution without difficulty.

Motion Planning Unlike the discrete world, the continuous world changes many things. Time should be considered in a continuous manner. Robots are in configuration space (a vector position and an orientation). Robots' rigid body should be considered. Following work introduces an online motion planner for this type of robots in somewhat challenging environment.

- [74] ***Kangjin Kim**, Yu Zhang, Georgios Fainekos, **Online Motion Planning for Autonomous Vehicles in Unstructured Road Networks**, under review, submitted in Fall 2019*

Even though the above approach enables to generate motion maneuver in the given world, resolving a conflict among agents in this setting is a different story. Utilizing the technique from [27, 73], a possible extension can be suggested.

Publications which do not appear in this thesis Even though related to this thesis topic, the following publications are not part of this thesis.

- [75] *S. Srinivas, R. Kermani, **K. Kim**, Y. Kobayashi, and G. Fainekos, **A Graphical Language for LTL Motion and Mission Planning**, In 2013 IEEE International Conference on Robotics and Biomimetics (ROBIO), pages 704–709, Dec 2013*

- [76] *Wei Wei, **Kangjin Kim**, and Georgios Fainekos, **Extended LTLvis Motion Planning Interface**, In 2016 IEEE International Conference on Systems,*

Man, and Cybernetics, SMC 2016 - Conference Proceedings, pages 4194–4199, United States, 2 2017. Institute of Electrical and Electronics Engineers Inc.

Chapter 2

MISSION PLANNING

2.1 Motivation

Even though the LTL Revision Problem was defined and studied in the past in various forms [17, 51], one fundamental question remained open: Can we efficiently compute a specification revision when the robot environment is known? This chapter develops polynomial time approximations and heuristics to this computationally hard problem for various versions of the problem: on weighted (Sec. 2.4) and unweighted transition systems (Sec. 2.3), and on requirements with (Sec. 2.5) and without preferences (Sec. 2.3). In addition, the chapter develops the theory needed to extend the specification revision problem to multi-robot systems where the robots have limited communication and sensing range (Sec. 2.6). This class of problems had not been considered before in the literature.

In this chapter, necessary background is built, the problems are defined, and solutions are provided, followed by potential future extensions and conclusions.

2.2 Preliminaries

In this section, we review some basic results on the automata theoretic planning and the specification revision problem from [4, 17].

Throughout this section, we will use the notation $\mathcal{P}(A)$ for representing the powerset of a set A, i.e., $\mathcal{P}(A) = \{B \mid B \subseteq A\}$. We also define a set difference as $A \setminus B = \{x \in A \mid x \notin B\}$.

2.2.1 LTL Planning

We assume that the combined actions of the robot/team of robots and their operating environment can be represented using an FSM. This is for discrete abstraction of the continuous robotic control system [4]. Each state of the Finite State Machine (FSM) $\mathcal{T}$ is labeled by a number of symbols from a set $\Pi = \{\pi_0, \pi_1, \dots, \pi_n\}$ that represent regions in the configuration space of the robot or, more generally, actions that can be performed by the robot.

Definition 1 (FSM)**.** *A Finite State Machine is a tuple $\mathcal{T} = (Q, Q_0, \rightarrow_{\mathcal{T}}, h_{\mathcal{T}}, w, \Pi)$ where: Q is a set of states; $Q_0 \subseteq Q$ is the set of possible initial states; $\rightarrow_{\mathcal{T}} \subseteq Q \times Q$ is the transition relation; $h_{\mathcal{T}} : Q \rightarrow \mathcal{P}(\Pi)$ maps each state q to the set of atomic propositions that are true on q; and $w : \rightarrow_{\mathcal{T}} \rightarrow \mathbb{R}_{\geq 0}$ returns the weight of each transition.*

We define a *path* on the FSM to be a sequence of states and a *trace* to be the corresponding sequence of sets of propositions. Formally, a path is a function $p : \mathbb{N} \rightarrow Q$ such that for each $i \in \mathbb{N}$ we have $p(i) \rightarrow_{\mathcal{T}} p(i+1)$ and the corresponding trace is the function composition $\bar{p} = h_{\mathcal{T}} \circ p : \mathbb{N} \rightarrow \mathcal{P}(\Pi)$. The language $\mathcal{L}(\mathcal{T})$ of $\mathcal{T}$ consists of all possible traces.

Assumption 1. *All the states on $\mathcal{T}$ are reachable.*

As a specification language, we will use LTL. Its syntax and semantics are as followings.

Definition 2 (LTL Syntax)**.** *The set $LTL(\Pi)$ of all LTL formulas built over a set of atomic propositions Π is defined recursively as*

$$\phi ::= \pi \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \mathbf{X}\phi_1 \mid \phi_1 \mathcal{U} \phi_2$$

for $\pi \in \Pi$ and ϕ_1, $\phi_2 \in LTL(\Pi)$, where $\neg$ is negation, $\vee$ is disjunction, $\mathbf{X}$ is "next", and $\mathcal{U}$ is strong "until".

From aforementioned operators, we can derive the following operators: $\wedge$ for conjunction, $\Rightarrow$ for implication, $\Leftrightarrow$ for equivalence, $\mathbf{F}$ for "eventually", $\mathbf{G}$ for "always" and $\mathcal{R}$ for weak "until". In the following, we let $(\bar{p}, i) \models \phi$ denote the satisfiability of an LTL formula ϕ over a trace $\bar{p}$ starting at time $i \in \mathbb{N}$. We define the language $\mathcal{L}(\phi)$ to be the set of all traces that satisfy ϕ at time 0, i.e., $\mathcal{L}(\phi) = \{\bar{p} \in \mathcal{P}(\Pi)^\omega \mid (\bar{p}, 0) \models \phi\}$.

Definition 3 (LTL Semantics)**.** *The semantics of any LTL formula $\phi \in LTL(\Pi)$ is defined as (for $i, j, k \in \mathbb{N}$):*

$$
\begin{array}{ll}
(\bar{p}, i) \models \top & \\
(\bar{p}, i) \not\models \bot & \\
(\bar{p}, i) \models \pi & \textit{iff } \pi \in \bar{p}(i) \\
(\bar{p}, i) \models \neg\phi_1 & \textit{iff } (\bar{p}, i) \not\models \phi_1 \\
(\bar{p}, i) \models \phi_1 \vee \phi_2 & \textit{iff } (\bar{p}, i) \models \phi_1 \textit{ or } (\bar{p}, i) \models \phi_2 \\
(\bar{p}, i) \models \mathbf{X}\phi_1 & \textit{iff } (\bar{p}, i+1) \models \phi_1 \\
(\bar{p}, i) \models \phi_1 \mathcal{U} \phi_2 & \textit{iff } \exists k \geq i \textit{ s.t. } (\bar{p}, k) \models \phi_2 \textit{ and } \forall i \leq j \leq k \ . \ (\bar{p}, j) \models \phi_1
\end{array}
$$

Intuitively, the formula $\mathbf{X}\phi_1$ means that ϕ_1 is true in the next time "step" (the next position in the trace $\bar{p}$). In addition, the formula $\phi_1 \mathcal{U} \phi_2$ means that ϕ_1 is true until ϕ_2 becomes true.

LTL formulas can be represented in the ω-automata which will impose certain requirements on the traces of $\mathcal{T}$. ω-automata differ from the classic finite automata in that they accept infinite strings (traces of $\mathcal{T}$ in our case).

Definition 4. *An automaton is a tuple $\mathcal{B} = (S_\mathcal{B}, s_0^\mathcal{B}, \mathcal{P}(\Pi), \rightarrow_\mathcal{B}, F_\mathcal{B}, \theta)$ where: $S_\mathcal{B}$ is a finite set of states; $s_0^\mathcal{B}$ is the initial state; $\mathcal{P}(\Pi)$ is an input alphabet; $\rightarrow_\mathcal{B} \subseteq S_\mathcal{B} \times \mathcal{P}(\Pi) \times S_\mathcal{B}$ is a transition relation; $F_\mathcal{B} \subseteq S_\mathcal{B}$ is a set of final states; and $\theta : \Pi \times S_\mathcal{B}^2 \rightarrow \mathbb{R}_{\geq 0}$ is a preference function.*

We also write $s \xrightarrow{l}_\mathcal{B} s'$ instead of $(s, l, s') \in \rightarrow_\mathcal{B}$. A *specification* automaton is an automaton with a Büchi acceptance condition where the input alphabet is the powerset of the set of labels, Π, of the system $\mathcal{T}$. A *run* r of a specification automaton $\mathcal{B}$ is a sequence of states $r : \mathbb{N} \rightarrow S_\mathcal{B}$ that occurs under an input trace $\bar{p}$ taking values in $\mathcal{P}(\Pi)$. That is, for $i = 0$ we have $r(0) = s_0^\mathcal{B}$ and for all $i \geq 0$ we have $r(i) \xrightarrow{\bar{p}(i)}_\mathcal{B} r(i+1)$. Let $\lim(\cdot)$ be the function that returns the set of states that are encountered infinitely often in the run r of $\mathcal{B}$. Then, a run r of an automaton $\mathcal{B}$ over an infinite trace $\bar{p}$ is *accepting* if and only if $\lim(r) \cap F_\mathcal{B} \neq \emptyset$. This is called a Büchi acceptance condition. Finally, we define the language $\mathcal{L}(\mathcal{B})$ of $\mathcal{B}$ to be the set of all traces $\bar{p}$ that have a run that is accepted by $\mathcal{B}$.

In order to simplify the discussion, we will make the following notations and assumption without loss of generality. We define

- the set $E_\mathcal{B} \subseteq S_\mathcal{B}^2$, such that $(s, s') \in E_\mathcal{B}$ iff $\exists l \in \mathcal{P}(\Pi), s \xrightarrow{l}_\mathcal{B} s'$; and,
- the function $\lambda_\mathcal{B} : S_\mathcal{B}^2 \rightarrow \mathcal{P}(\Pi)$ which maps a pair of states to the label of the corresponding transition.

That is, if $s \xrightarrow{l}_\mathcal{B} s'$, then $\lambda_\mathcal{B}(s, s') = l$; and if $(s, s') \notin E_\mathcal{B}$, then $\lambda(s, s') = \emptyset$.

Assumption 2. *Between any two states of the specification automaton there exists at most one transition.*

In brief, our goal is to generate paths on $\mathcal{T}$ that satisfy the specification $\mathcal{B}_\mathbf{s}$. In automata theoretic terms, we want to find the subset of the language $\mathcal{L}(\mathcal{T})$ which also belongs to the language $\mathcal{L}(\mathcal{B}_{\mathbf{s}})$. This subset is simply the intersection of the two languages $\mathcal{L}(\mathcal{T}) \cap \mathcal{L}(\mathcal{B}_{\mathbf{s}})$ and it can be constructed by taking the product $\mathcal{T} \times \mathcal{B}_{\mathbf{s}}$ of the FSM $\mathcal{T}$ and the specification automaton $\mathcal{B}_{\mathbf{s}}$. Informally, the automaton $\mathcal{B}_{\mathbf{s}}$ restricts the behavior of the system $\mathcal{T}$ by permitting only certain acceptable transitions. Then, given an initial state in the FSM $\mathcal{T}$, we can choose a particular trace from $\mathcal{L}(\mathcal{T}) \cap \mathcal{L}(\mathcal{B}_{\mathbf{s}})$ according to a preferred criterion.

Definition 5. *The product automaton $\mathcal{A} = \mathcal{T} \times \mathcal{B}_{\mathbf{s}}$ is the automaton $\mathcal{A} = (S_{\mathcal{A}}, s_0^{\mathcal{A}}, \mathcal{P}(\Pi), \rightarrow_{\mathcal{A}}, F_{\mathcal{A}})$ where: $\mathcal{S}_{\mathcal{A}} = Q \times S_{\mathcal{B}_{\mathbf{s}}}$; $s_0^{\mathcal{A}} = \{(q_0, s_0^{\mathcal{B}_{\mathbf{s}}}) \mid q_0 \in Q_0\}$; $\rightarrow_{\mathcal{A}} \subseteq S_{\mathcal{A}} \times \mathcal{P}(\Pi) \times S_{\mathcal{A}}$ s.t. $(q_i, s_i) \xrightarrow{l}_{\mathcal{A}} (q_j, s_j)$ iff $q_i \rightarrow_{\mathcal{T}} q_j$ and $s_i \xrightarrow{l}_{\mathcal{B}_{\mathbf{s}}} s_j$ with $l = h_{\mathcal{T}}(q_j)$, and $F_{\mathcal{A}} = Q \times F_{\mathcal{B}}$ is the set of accepting states.*

Note that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{T}) \cap \mathcal{L}(\mathcal{B}_{\mathbf{s}})$. We say that $\mathcal{B}_{\mathbf{s}}$ is *satisfiable* on $\mathcal{T}$ if $\mathcal{L}(\mathcal{A}) \neq \emptyset$.

Definition 6. *(ultimately periodic) Given a product automaton $\mathcal{A} = (S_{\mathcal{A}}, s_0^{\mathcal{A}}, \mathcal{P}(\Pi), \rightarrow_{\mathcal{A}}, F_{\mathcal{A}})$ and an infinite path $p \in \mathcal{L}(\mathcal{A})$, p is ultimately periodic if and only if $p = p_0 \ldots p_m \ldots p_{m+n} \ldots p_{m+2\times n} \ldots$ where $p_0 \in s_0^{\mathcal{A}}$, $p_m = p_{m+j\times n} \in F_{\mathcal{A}}$ and $m, n, j \in \mathbb{N}$.*

Definition 7. *(LTL Planning) Given $\mathcal{T}$, $\mathcal{B}_{\mathbf{s}}$, its product automaton $\mathcal{A} = \mathcal{T} \times \mathcal{B}_{\mathbf{s}} = (S_{\mathcal{A}}, s_0^{\mathcal{A}}, \mathcal{P}(\Pi), \rightarrow_{\mathcal{A}}, F_{\mathcal{A}})$ and $\mathcal{L}(\mathcal{A}) \neq \emptyset$, we define LTL planning as finding an ultimately periodic path p where $p \in \mathcal{L}(\mathcal{A})$ if $\mathcal{T}$ is a unweighted transition system.*

Finding an infinite (satisfiable) path on $\mathcal{T} \times \mathcal{B}_{\mathbf{s}}$ is an easy algorithmic problem (see [1]). First, we convert automaton $\mathcal{T} \times \mathcal{B}_{\mathbf{s}}$ to a directed graph and, then, we find the strongly connected components (SCC) in that graph.

If at least one SCC that contains a final state is reachable from an initial state, then there exist accepting (infinite) runs on $\mathcal{T} \times \mathcal{B}_{\mathbf{s}}$ that have a finite representation. Each such run consists of two parts: **prefix:** a part that is executed only once (from an initial state to a final state) and, **lasso:** a part that is repeated infinitely (from a final state back to itself). Note that if no final state is reachable from the initial or if no final state is within an SCC, then the language $\mathcal{L}(\mathcal{A})$ is empty and, hence, the high level synthesis problem does not have a solution. Namely, the synthesis phase has failed and we cannot find a system behavior that satisfies the specification. For this case, we shall introduce LTL Revision problem in Section 2.3.

In the next subsection, we will consider the case that $\mathcal{T}$ is a weighted transition system and we have to consider its optimal run.

We remark that the unweighted transition system is a transition system where every transition between any two states has same weight such as 1. This abstracted system is useful when we consider a simple environment for pebble motion or sliding puzzle, or when we only want to check if $\mathcal{L}(\mathcal{T} \times \mathcal{B}_{\mathbf{s}})$ is empty or not, without considering its optimal run.

2.2.2 Weighted LTL Planning

In this subsection, we will cover LTL planning under a weighted transition system. In order to plan on this setting, we convert it to a graph search problem. First, a product automaton of a system $\mathcal{T}$ and a specification $\mathcal{B}_{\mathbf{s}}$ can be converted to a directed graph. Second, finding the solution of the problem can be transfered to finding an acceptable path on the graph. Third, the cost of the path should be defined. Fourth, optimal solution can be chosen, comparing the cost of each path. First two steps are same as *LTL planning* in the previous section. Here, we will discuss on two cost functions $\mathcal{C}_1$ and $\mathcal{C}_2$ and their optimal solution.

Suppose that there are a system $\mathcal{T} = (Q, Q_0, \rightarrow_{\mathcal{T}}, h_{\mathcal{T}}, w, \Pi)$ and a specification $\mathcal{B}_\phi$ $= (S_{\mathcal{B}}, s_0^{\mathcal{B}}, \mathcal{P}(\Pi), \rightarrow_{\mathcal{B}}, F_{\mathcal{B}}, \theta)$ where $\phi = \mathbf{F}\pi$ and $\pi \in \Pi$. Formalizing a cross-product $\mathcal{A}$ $= (S_{\mathcal{A}}, s_0^{\mathcal{A}}, \rightarrow_{\mathcal{A}}, w, F_{\mathcal{A}})$ of $\mathcal{T}$ and $\mathcal{B}_\phi$ as a graph search problem, we can solve the LTL planning problem. Then, a possible solution can be an infinite path $p = v_0 \ldots v_m \ldots$

where $v_0 = s_0^{\mathcal{A}}$ and $v_m \in F_{\mathcal{A}}$. This is because when we focus on its prefix part, p eventually visits v_m where $v_m \in F_{\mathcal{A}}$ and $\bar{p}(m) \models \pi$. Note that if the LTL formula is more complex, then we have to consider the lasso part.

Suppose that $\phi = \mathbf{GF}\pi$.

More precisely, given an acceptable path $p \in \mathcal{L}(\mathcal{A})$, there exist constants $m, n \in \mathbb{N}$ s.t. for all $k \geq 0$, we have $\bar{p}(m+k) = \bar{p}(m + \mod(k,n))$, where $\mod$ is the modulo operation. In other words, an *ultimately periodic* trace consists of a *finite* (or *prefix*) part $\bar{p}(0)\bar{p}(1)\ldots\bar{p}(m-1)$ and a loop (or *lasso*) part $\bar{p}(m)\bar{p}(m+1)\ldots\bar{p}(m+n-1)$ repeated ad infinitum. Based on that observation, we can construct a cost function $\mathcal{C}_1$ that will help us compute a plan that satisfies our overall cost bound C:

$$\mathcal{C}_1(p) = \max\left\{\sum_{i=0}^{m-1}(w(p(i), p(1+i))), \sum_{i=0}^{n-1}(w(p(m+i), p(m+i+1))\right\}.$$

The first quantity is the cost of the path to get to the state $p(m)$, while the second is the cost of the steady state.

Based on the previous cost function, it is easy to compute the optimal path on $\mathcal{T}$ that satisfies $\mathcal{B}_{\mathbf{s}}$. We define a weight function $w_{\mathcal{A}}$ on the edges of $\mathcal{A}$ as follows: $\forall\ ((q_i, s_i), l, (q_j, s_j)) \in \rightarrow_{\mathcal{A}}$, $w_{\mathcal{A}}((q_i, s_i), (q_j, s_j)) = w(q_i, q_j)$. Then we run Dijkstra's shortest path algorithm [77] to compute the shortest path from the initial state to any accepting state in $F_{\mathcal{A}}$. Then, starting from each reachable accepting state, we compute the shortest path to get back to that state. Finally, we can select a path that minimizes $\mathcal{C}_1(p)$.

The procedure described above is meaningful only in certain motion planning scenarios. For example, such a planning framework can be useful in cases where the vehicle can continuously recharge, e.g., using solar panels, or regenerate energy while operating. In these cases, it is desirable that the cost of the periodic part of the plan as well as the transient behavior have cost less than cost budget C which could indicate depleted power sources.

A different LTL optimal planning framework has been developed in [13, 78]. In [78], the authors solve the optimal planning problem for specifications of the form

$$\phi := \varphi \wedge \Box\Diamond\psi, \tag{2.1}$$

where ψ is a Boolean combination of atomic propositions that must be satisfied infinitely often. For instance, we could set $\psi = \pi^\star$ where $\pi^\star$ is an atomic proposition indicating a recharging or time reseting operation in a particular location in the environment. Then, the optimal control framework attempts to compute paths that when passing through $\pi^\star$ have cost less than C.

In the following, we will assume that ψ is a single atomic proposition, i.e., $\psi = \pi^\star$. The discussion can be generalized to Boolean combinations of atomic propositions. Formally, given a path p, the function $\alpha_p^{\pi^\star} : \mathbb{N} \to \mathbb{N}$ returns the i-th appearance of $\pi^\star$ in $\bar{p}$. The cost $\mathcal{C}_2(p)$ of a path is defined as

$$\mathcal{C}_2(p) = \limsup_{i \to +\infty} \sum_{j=\alpha(i)}^{\alpha(i+1)} w(p(j), p(j+1))$$

$\mathcal{C}_2(p)$ is finite only if $\pi^\star$ occurs periodically in the trajectory. The main results from [78] can be summarized as follows.

Theorem 1. *There exists $\bar{p} \in \mathcal{L}(\mathcal{A})$ s.t. $\bar{p}$ is ultimately periodic and the corresponding path p minimizes the cost function $\mathcal{C}_2(p)$.*

The computation of a path that minimizes $\mathcal{C}_2$ is similar to the algorithm that produces plans that minimize $\mathcal{C}_1$, but with a major difference. There is a set of nodes R – potentially different from $F_{\mathcal{A}}$ – that are labeled by $\pi^\star$ and when visited they reset the cost of a path. A detailed description of the algorithm appears in [78].

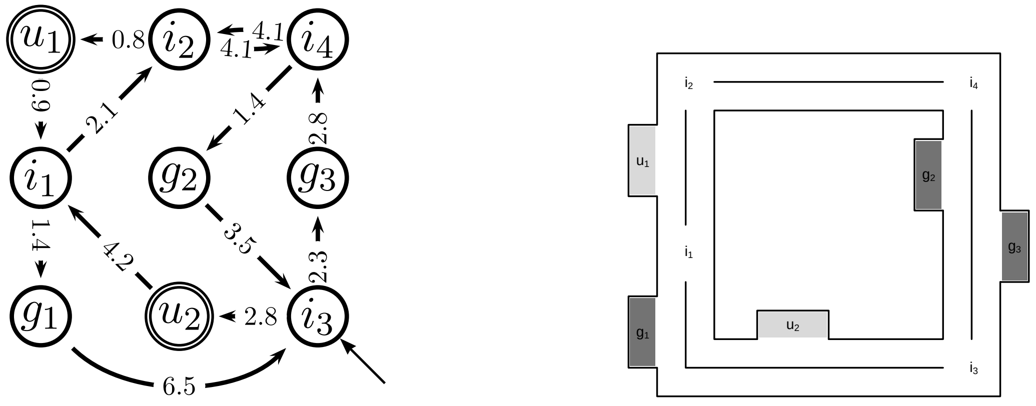


Figure 2.1: Illustration of the simple road network environment of Example 13. The robot is required to drive right-side of the road.

The paths computed with the above process are guaranteed to be the minimum cost paths. In realistic scenarios, it is to be expected to have hard constraints on the allowable worst case costs. The following example presents such a typical scenario for motion planing of a mobile robot in a road network inspired from [78].

Example 1. *(Robot Motion Planning) We consider a mobile robot which operates in a road network in Fig. 2.12. Initially, the robot is placed in the intersection labeled by i_3. The robot must accomplish the task: "Periodically visit gather locations g_1, g_2, or g_3 to gather data, and periodically visit upload locations u_1 or u_2 to upload the gathered data." The goal is to find an optimal path that minimizes the traveling time from one upload location to another or same upload location, while doing the mission. This natural language mission can be translated to a LTL formula $\phi := \Box\Diamond\varphi \wedge \Box\Diamond\pi$ where $\varphi := g_1 \vee g_2 \vee g_3$ and $\pi := u_1 \vee u_2$. Given a cost bound $C = 13$, cost reset nodes $R = \{u_1, u_2\}$ and a specification ϕ, an optimal run p is $i_3g_3i_4g_2i_3(u_2i_1i_2u_1i_1g_1i_3)^+$. In this trace, the cost for prefix part $i_3 \rightsquigarrow u_2$ is 12.8 and the costs for suffix part $u_2 \rightsquigarrow u_1$ and $u_1 \rightsquigarrow u_2$ are 7.1 and 11.6, respectively.*

In the next section, we will introduce two main problems. First one is about asking the case when a user's original intention is unrealizable on a given environment, and second one is about a scalability issue of multi-agent LTL planning.

2.3 LTL Revision Problem

Intuitively, a revised specification is one that can be satisfied on the discrete abstraction of the workspace of the robot. In order to search for a minimal revision, we need first to define an ordering relation on automata as well as a distance function between automata. Similar to the case of LTL formulas in [17], we do not want to consider the "space" of all possible automata, but rather the "space" of specification automata which are semantically close to the initial specification automaton $\mathcal{B}_{\mathbf{s}}$. The later will imply that we remain close to the initial intention of the designer. We propose that this space consists of all the automata that can be derived from $\mathcal{B}_{\mathbf{s}}$ by relaxing the restrictions for transitioning from one state to another. In other words, we introduce possible transitions between two states of the specification automaton. Our definition of the ordering relation between automata relies upon the previous assumptions.

Definition 8 (Relaxation)**.** *Let* $\mathcal{B}_1 = (S_{\mathcal{B}_1}, s_0^{\mathcal{B}_1}, \mathcal{P}(\Pi), \rightarrow_{\mathcal{B}_1}, F_{\mathcal{B}_1}, \theta_{\mathcal{B}_1})$ *and* $\mathcal{B}_2 = (S_{\mathcal{B}_2}, s_0^{\mathcal{B}_2}, \mathcal{P}(\Pi), \rightarrow_{\mathcal{B}_2}, F_{\mathcal{B}_2}, \theta_{\mathcal{B}_2})$ *be two specification automata. Then, we say that* $\mathcal{B}_2$ *is a relaxation of* $\mathcal{B}_1$ *and we write* $\mathcal{B}_1 \preceq \mathcal{B}_2$ *if and only if* $S_{\mathcal{B}_1} = S_{\mathcal{B}_2} = S$, $s_0^{\mathcal{B}_1} = s_0^{\mathcal{B}_2}$, $F_{\mathcal{B}_1} = F_{\mathcal{B}_2}$, $\theta_{\mathcal{B}_1} = \theta_{\mathcal{B}_2}$ *and*

1. $\forall (s, l, s') \in \rightarrow_{\mathcal{B}_1} - \rightarrow_{\mathcal{B}_2} \,.\, \exists l' \,.$

   $(s, l', s') \in \rightarrow_{\mathcal{B}_2} - \rightarrow_{\mathcal{B}_1}$ *and* $l' \subseteq l$.

2. $\forall (s, l, s') \in \rightarrow_{\mathcal{B}_2} - \rightarrow_{\mathcal{B}_1} \,.\, \exists l' \,.$

   $(s, l', s') \in \rightarrow_{\mathcal{B}_1} - \rightarrow_{\mathcal{B}_2}$ *and* $l \subseteq l'$.

We remark that $\preceq$ is a partial order over specification automata. Also, if $\mathcal{B}_1 \preceq \mathcal{B}_2$, then $\mathcal{L}(\mathcal{B}_1) \subseteq \mathcal{L}(\mathcal{B}_2)$ since the relaxed automaton allows more behaviors to occur. It is possible that two automata $\mathcal{B}_1$ and $\mathcal{B}_2$ cannot be compared under relation $\preceq$.

Example 2. *Consider the specification automaton $\mathcal{B}_\mathbf{s}$ and the automata $\mathcal{B}_1$-$\mathcal{B}_3$ in Fig. 2.2. Requirements 1 and 2 in Def. 16 specify that the two automata must have transitions between exactly the same states*[1] *. Moreover, if the label of a transition between the same pair of states on the two automata differs, then the label on the relaxed automaton must be a subset of the label of the original automaton. The latter means that we have relaxed the constraints that permit a transition on the specification automaton.*

We have that $\mathcal{B}_\mathbf{s} \preceq \mathcal{B}_1$, since

$$\begin{aligned}
\rightarrow_{\mathcal{B}_\mathbf{s}} =& \{(0, \{\pi_0^-\}0), (1, \{\pi_0^-\}, 1), (2, \{\pi_0^-\}, 2) \\
& (0, \{\pi_0^-, \pi_1\}, 1), (1, \{\pi_0^-, \pi_2\}, 2), (0, \{\pi_0^-, \pi_1, \pi_2\}, 2)\} \\
\rightarrow_{\mathcal{B}_1} =& \{(0, \{\pi_0^-\}0), (1, \emptyset, 1), (2, \{\pi_0^-\}, 2) \\
& (0, \{\pi_0^-, \pi_1\}, 1), (1, \{\pi_0^-, \pi_2\}, 2), (0, \{\pi_2\}, 2)\} \\
\rightarrow_{\mathcal{B}_\mathbf{s}} - \rightarrow_{\mathcal{B}_1} =& \{(1, \{\pi_0^-\}, 1), (0, \{\pi_0^-, \pi_1, \pi_2\}, 2)\} \\
\rightarrow_{\mathcal{B}_1} - \rightarrow_{\mathcal{B}_\mathbf{s}} =& \{(1, \emptyset, 1), (0, \{\pi_2\}, 2)\}
\end{aligned}$$

and $\emptyset \subseteq \{\pi_0^-\}$, $\{\pi_2\} \subseteq \{\pi_0^-, \pi_1, \pi_2\}$.

Similarly, we have $\mathcal{B}_\mathbf{s} \parallel \mathcal{B}_2$ since $\rightarrow_{\mathcal{B}_\mathbf{s}} - \rightarrow_{\mathcal{B}_2} = \{(0, \{\pi_0^-, \pi_1\}, 1)\}$ while $\rightarrow_{\mathcal{B}_2} - \rightarrow_{\mathcal{B}_\mathbf{s}} = \emptyset$, i.e., we have removed a transition between two states. We have $\mathcal{B}_\mathbf{s} \parallel \mathcal{B}_3$ since $\rightarrow_{\mathcal{B}_\mathbf{s}} - \rightarrow_{\mathcal{B}_3} = \emptyset$ while $\rightarrow_{\mathcal{B}_3} - \rightarrow_{\mathcal{B}_\mathbf{s}} = \{(2, \{\pi_2\}, 0)\}$, i.e., we have added a

[1]To keep the presentation simple, we do not extend the definition of the ordering relation to isomorphic automata. Also, this is not required in our technical results since we are actually going to construct automata which are relaxations of a specification automaton.

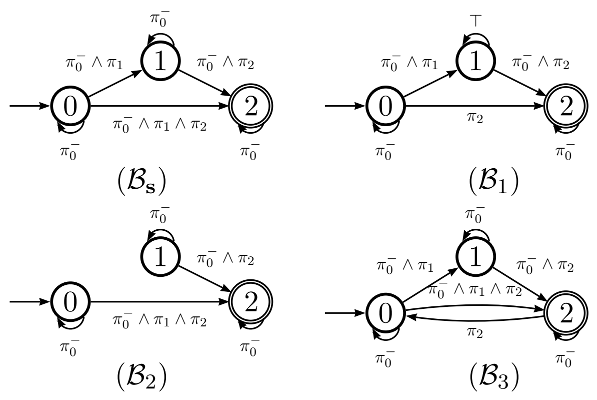


Figure 2.2: Example 2. $\mathcal{B}_\mathbf{s}$: the initial specification automaton, here $\pi_0^- \equiv \neg\pi_0$; $\mathcal{B}_\mathbf{s} \preceq \mathcal{B}_1$; $\mathcal{B}_\mathbf{s} \parallel \mathcal{B}_2$; $\mathcal{B}_\mathbf{s} \parallel \mathcal{B}_3$.

transition between two states. Recall that between any two states we may have only one transition. $\triangle$

We can now define the set of automata over which we will search for a revision.

Definition 9. *Given a system $\mathcal{T}$ and a specification automaton $\mathcal{B}_\mathbf{s}$, the set of valid relaxations of $\mathcal{B}_\mathbf{s}$ is defined as $\mathfrak{R}(\mathcal{B}_\mathbf{s}, \mathcal{T}) = \{\mathcal{B} \mid \mathcal{B}_\mathbf{s} \preceq \mathcal{B} \text{ and } \mathcal{L}(\mathcal{T} \times \mathcal{B}) \neq \emptyset\}$.*

We can now search for a solution in the set $\mathfrak{R}(\mathcal{B}_\mathbf{s}, \mathcal{T})$. Different solutions can be compared from their revision sets.

We can now search for a minimal solution in the set $\mathfrak{R}(\mathcal{B}_\mathbf{s}, \mathcal{T})$. That is, we can search for some $\mathcal{B} \in \mathfrak{R}(\mathcal{B}_\mathbf{s}, \mathcal{T})$ such that if for any other $\mathcal{B}' \in \mathfrak{R}(\mathcal{B}_\mathbf{s}, \mathcal{T})$, we have $\mathcal{B}' \preceq \mathcal{B}$, then $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{B}')$. However, this does not imply that a minimal solution semantically is minimal structurally as well. In other words, it could be the case that $\mathcal{B}_1$ and $\mathcal{B}_2$ are minimal relaxations of some $\mathcal{B}_\mathbf{s}$, but $\mathcal{B}_1 \parallel \mathcal{B}_2$ and, moreover, $\mathcal{B}_1$ requires the modification of only one transition while $\mathcal{B}_2$ requires the modification of two transitions. Therefore, we must define a metric on the set $\mathfrak{R}(\mathcal{B}_\mathbf{s}, \mathcal{T})$, which accounts for the number of changes from the initial specification automaton $\mathcal{B}_\mathbf{s}$.

Definition 10. *Given a system $\mathcal{T}$ and a specification automaton $\mathcal{B}_{\mathbf{s}}$, we define the distance of any $\mathcal{B} \in \mathfrak{R}(\mathcal{B}_{\mathbf{s}}, \mathcal{T})$ from $\mathcal{B}_{\mathbf{s}}$ to be $\mathbf{dist}_{\mathcal{B}_{\mathbf{s}}}(\mathcal{B}) = \sum_{(s,s') \in E_{\mathcal{B}_{\mathbf{s}}}} |\lambda_{\mathcal{B}_{\mathbf{s}}}(s,s') - \lambda_{\mathcal{B}}(s,s')|$ where $|\cdot|$ is the cardinality of the set.*

Problem 1 (LTL Revision Problem)**.** *Given a system $\mathcal{T}$ and a specification automaton $\mathcal{B}_{\mathbf{s}}$ such that $\mathcal{L}(\mathcal{T} \times \mathcal{B}_{\mathbf{s}}) = \emptyset$, find $\mathcal{B} \in \arg\min\{\mathbf{dist}_{\mathcal{B}_{\mathbf{s}}}(\mathcal{B}') \mid \mathcal{B}' \in \mathfrak{R}(\mathcal{B}_{\mathbf{s}}, \mathcal{T})\}$.*

2.3.1 LTL Revision as Graph Search Problems

In this section, we present how Problem 1 can be posed as shortest path problems on labeled graphs.

Without loss of generality, we will assume that for any given specification ϕ, each atomic proposition π in ϕ appears only once in ϕ. If this is not the case, then for each additional occurrence of π in ϕ, we can replace it with a new atomic proposition, add the proposition to Π and modify the map $h_{\mathcal{T}}$ accordingly. This change is necessary in order to uniquely identify in ϕ which propositions need to be replaced by $\top$.

Given a specification ϕ, we can construct the corresponding specification automaton $\mathcal{B}_{\mathbf{s}}$. Using the weighted labeled transition system $\mathcal{T}$ and the specification automaton $\mathcal{B}_{\mathbf{s}}$, we can construct a graph $G_{\mathcal{A}}$ which corresponds to the product automaton $\mathcal{A}$ while considering the effect of revisions and the weights.

Definition 11. *Given a system $\mathcal{T}$ and a specification automaton $\mathcal{B}_{\mathbf{s}}$, we define the graph $G_{\mathcal{A}} = (V, E, v_s, V_f, W, \Lambda, R, \theta)$, which corresponds to the product $\mathcal{A} = \mathcal{T} \times \mathcal{B}_{\mathbf{s}}$ as*

- *$V = \mathcal{S}_{\mathcal{A}}$ is the set of nodes*
- *$E = E_{\mathcal{A}} \cup E_D \subseteq \mathcal{S}_{\mathcal{A}} \times \mathcal{S}_{\mathcal{A}}$, where*
  - *$E_{\mathcal{A}}$ is the set of edges that correspond to transitions on $\mathcal{A}$, i.e., $((q,s),(q',s')) \in E_{\mathcal{A}}$ iff $\exists l \in \mathcal{P}(\Pi)$. $(q,s) \xrightarrow{l}_{\mathcal{A}} (q',s')$; and*

  - *E_D is the set of edges that correspond to disabled transitions, i.e., $((q,s),(q',s')) \in E_D$ iff $q \to_{\mathcal{T}} q'$ and $s \xrightarrow{l}_{\mathcal{B}_\mathbf{s}} s'$ with $l \cap (\Pi - h_{\mathcal{T}}(q')) \neq \emptyset$.*

- *$v_s = s_0^{\mathcal{A}}$ is the source node,*
- *$V_f = F_{\mathcal{A}}$ is the set of sinks,*
- *$W : E \to \mathbb{R}_{\geq 0}$ assigns a weight to each edge in E. If $e = ((q,s),(q',s')) \in E$, then $w(e) = w(q)$.*
- *$\Lambda : E \to \mathcal{P}(\overline{\Pi})$ maps each edge of the graph to the set of symbols that need to be removed in order to enable the edge in the product automaton $\mathcal{A}$. If $e = ((q,s),(q',s')) \in E$, then $\Lambda(e) = \{\langle l, (s,s') \rangle \mid l = \lambda_{\mathcal{B}_\mathbf{s}}(s,s') - h_{\mathcal{T}}(q')\}$.*
- *$R = \{(q,s) \in V \mid \pi^\star \in h_{\mathcal{T}}(q)\}$ is the set of "cost reset" nodes.*
- *$\theta : \overline{\Pi} \to \mathbb{R}_{\geq 0}$ is the preference function of $\mathcal{B}_\mathbf{s}$ restricted on $\overline{\Pi}$.*

We remark that the graph $G_{\mathcal{A}}$ is essentially the same graph as the graph of $\mathcal{A}$ with the addition of the disabled edges due to the specification constraints. Therefore, any path on the graph of $\mathcal{A}$ appears as a path on $G_{\mathcal{A}}$.

In Problem 1, we assume that the transition system $\mathcal{T}$ is unweighted. Hence, when we construct the graph $G_{\mathcal{A}}$ for Problem 1, we can set the same weight for every edge. i.e., the weight of every edge $e \in E$ of $G_{\mathcal{A}}$ is 1. The solution of this problem is *an acceptable path* $p = v_0, v_1, \ldots, v_m, v_{m+1}, \ldots, v_{m+n} \ldots$ where $v_0 = s_0^{\mathcal{A}}$ and $v_m = v_{m+k\times n} \in V_f$ where $m, n, k \in \mathbb{N}$. In the path p, $v_0 v_1 \ldots v_m$ is the prefix part and $v_m v_{m+1} \ldots v_{m+n}$ is the lasso part. Note that for revision problems, we assume that $\mathcal{L}(\mathcal{A}) = \emptyset$. Hence $\mathcal{L}(G_{\mathcal{A}}) = \emptyset$ only with edges in $E_{\mathcal{A}}$. In order to have an acceptable path $p \in \mathcal{L}(G_{\mathcal{A}})$, we have to add some edges in E_D to this path p. The cost function of this graph $G_{\mathcal{A}}$ is related with the set of atomic propositions on this edges and we use the function Λ of $G_{\mathcal{A}}$. Before defining the cost funciton, we define $|\langle l, (s, s') \rangle| = |l|$, i.e., the size of a tuple $\langle l, (s, s') \rangle \in \overline{\Pi} \times E$ is defined to be the size of the set l. *The cost of the path* p is defined as following. $Cost(p) = |\tilde{\Lambda}(p)|$. $\tilde{\Lambda}(p) = \bigcup_{i=0}^{m+n-1} \Lambda(v_i, v_{i+1})$.

Example 3. *Suppose that there are two edges $(v_i, v_j) = ((q_i, s_i), (q_j, s_j))$ and $(v_k, v_h) = ((q_k, s_k), (q_h, s_h))$. The union of $\Lambda(v_i, v_j)$ and $\Lambda(v_k, v_h)$ is $\{\langle l, (s_i, s_j) \rangle\} \cup \{\langle l', (s_k, s_h) \rangle\}$. Then, if $(s_i, s_j) = (s_k, s_h)$, it is $\{\langle l \cup l', (s_i, s_j) \rangle\}$. Otherwise, it is $\{\langle l, (s_i, s_j) \rangle, \langle l', (s_k, s_h) \rangle\}$.*

Example 4. *Consider the Example in Fig. 2.3. In the figure, we provide a partial description of an FSM $\mathcal{T}$, a specification automaton $\mathcal{B}_{\mathbf{s}}$ and the corresponding product automaton $\mathcal{A}$. The dashed edges indicate disabled edges which are labeled by the atomic propositions that must be removed from the specification in order to enable the transition on the system. In this example, we do not have to add any new nodes since we have only one conjunctive clause on the transition of the specification automaton. It is easy to see now that in order to enable the path $(q_0, s_1), (q_1, s_1), (q_3, s_1)$ on the product automaton, we need to replace π_0 and π_2 with $\top$ in $\Phi_{\mathcal{B}_{\mathbf{s}}}(s_1, s_1) = \pi_0 \wedge \pi_1 \wedge \pi_2 \wedge \pi_3$. On the graph $G_{\mathcal{A}}$, this path corresponds to $\widetilde{\Lambda}(e_1 e_3) = \{\langle \{\pi_0, \pi_2\}, (s_1, s_1), 0 \rangle\}$. Similarly, in order to enable the path $(q_0, s_1), (q_1, s_1), (q_2, s_1)$ on the product automaton, we need to replace π_0, π_2 and π_3 with $\top$ in $\Phi_{\mathcal{B}_{\mathbf{s}}}(s_1, s_1)$. On the graph $G_{\mathcal{A}}$, this path corresponds to $\widetilde{\Lambda}(e_1 e_2) = \{\langle \{\pi_0, \pi_2 \, \pi_3\}, (s_1, s_1), 0 \rangle\}$.*

Therefore, the path defined by edges e_1 and e_3 is preferable over the path defined by edges e_1 and e_2. In the first case, we have cost $C(e_1 e_3) = 2$ which corresponds to relaxing 2 requirements, i.e., π_0 and π_2, while in the latter case, we have cost $C(e_1 e_2) = 3$ which corresponds to relaxing 3 requirements, i.e., π_0, π_2 and π_3. $\triangle$

A valid relaxation $\mathcal{B}$ should produce a reachable $v_f \in V_f$ with prefix and lasso path such that $\mathcal{L}(\mathcal{T} \times \mathcal{B}) \neq \emptyset$. *Optimal solution* is the path p such that *cost*(p) is minimized. The next section provides an algorithmic solution to this problem.

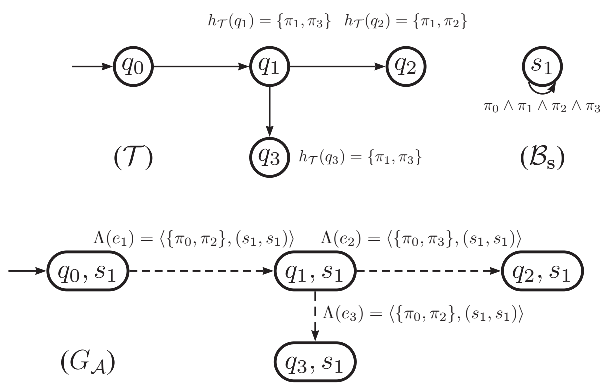

$h_{\mathcal{T}}(q_1) = \{\pi_1, \pi_3\}$
$h_{\mathcal{T}}(q_2) = \{\pi_1, \pi_2\}$
q_0
q_1
q_2
s_1
$\pi_0 \wedge \pi_1 \wedge \pi_2 \wedge \pi_3$
$(\mathcal{T})$
q_3
$h_{\mathcal{T}}(q_3) = \{\pi_1, \pi_3\}$
$(\mathcal{B}_\mathbf{s})$
$\Lambda(e_1) = \langle\{\pi_0, \pi_2\}, (s_1, s_1)\rangle$
$\Lambda(e_2) = \langle\{\pi_0, \pi_3\}, (s_1, s_1)\rangle$
q_0, s_1
q_1, s_1
q_2, s_1
$\Lambda(e_3) = \langle\{\pi_0, \pi_2\}, (s_1, s_1)\rangle$
$(G_{\mathcal{A}})$
q_3, s_1

Figure 2.3: Example 4. $\mathcal{T}$: part of the system; $\mathcal{B}_\mathbf{s}$: part of the specification automaton; $G_{\mathcal{A}}$: part of the graph that corresponds to the product automaton.

2.3.2 A Heuristic Algorithm for MRP

In this section, we present an approximation algorithm (AAMRP) for the Minimal Revision Problem (MRP). It is based on Dijkstra's shortest path algorithm [77]. The main difference from Dijkstra's algorithm is that instead of finding the minimum weight path to reach each node, AAMRP tracks the number of atomic propositions that must be removed from each edge on the paths of the graph $G_{\mathcal{A}}$.

The pseudocode for the AAMRP is presented in Algorithms 1 and 2. The main algorithm (Alg. 1) divides the problem into two tasks. First, in Line 4, it finds an approximation to the minimum number of atomic propositions from $\overline{\Pi}$ that must be removed to have a prefix path to each reachable sink (see Sec. 2.2.1). Then, in Line 9, it repeats the process from each reachable final state to find an approximation to the minimum number of atomic propositions that must be removed so that a lasso path is enabled. The combination of prefix/lasso that removes the minimal number of atomic propositions is returned to the user. We remark that from line 9, a set of atomic propositions found from prefix part is used when it starts searching for lasso path of every reachable $v_f \in \mathcal{V} \cap V_f$.

Algorithm 1: AAMRP($G_{\mathcal{A}}$)

```
Input: a graph $G_{\mathcal{A}} = (V, E, v_s, V_f, \overline{\Pi}, \Lambda)$
Output: the list $L$ of atomic propositions $\overline{\Pi}$ that must be removed from $\mathcal{B}_{\mathbf{s}}$
1  $L \leftarrow \overline{\Pi}$
2  $\mathcal{M}[:,:] \leftarrow (\overline{\Pi}, \infty)$
3  $\mathcal{M}[v_s,:] \leftarrow (\emptyset, 0)$                    ▷ Initialize the source node
4  $\langle \mathcal{M}, \mathbf{P}, \mathcal{V} \rangle \leftarrow$ FindMinPath($G_{\mathcal{A}}, \mathcal{M}, 0$)
5  Acceptable $\leftarrow$ $False$
6  for $v_f \in \mathcal{V} \cap V_f$ do
7      $L_p \leftarrow$ GetAPFromPath($v_s, v_f, \mathcal{M}, \mathbf{P}$)
8      $\mathcal{M}'[:,:] \leftarrow (\overline{\Pi}, \infty)$
9      $\mathcal{M}'[v_f,:] \leftarrow (L_p, |L_p|)$          ▷ Store APs of $v_s \rightsquigarrow v_f$ to $\mathcal{M}'[v_f,:]$
10     $G'_{\mathcal{A}} \leftarrow (V, E, v_f, \{v_f\}, \overline{\Pi}, \Lambda)$
11     $\langle \mathcal{M}', \mathbf{P}', \mathcal{V}' \rangle \leftarrow$ FindMinPath($G'_{\mathcal{A}}, \mathcal{M}', 1$)
12     if $v_f \in \mathcal{V}'$ then
13         $L' \leftarrow$ GetAPFromPath($v_f, v_f, \mathcal{M}', \mathbf{P}'$)      ▷ Get APs from $\mathcal{M}'[v_f,:]$
14         if $|L'| \leq |L|$ then
15             $L \leftarrow L'$
16         Acceptable $\leftarrow$ $True$
17 if $\neg$Acceptable then
18     $L \leftarrow \emptyset$
19 return $L$
```

Algorithm 2 follows closely Dijkstra's shortest path algorithm [77]. It maintains a list of visited nodes $\mathcal{V}$ and a table $\mathcal{M}$ indexed by the graph vertices which stores the set of atomic propositions that must be removed in order to reach a particular node on the graph. Given a node v, the size of the set $|\mathcal{M}[v, 1]|$ is an upper bound on the minimum number of atomic propositions that must be removed. That is, if we remove all $\overline{\pi} \in \mathcal{M}[v, 1]$ from $\mathcal{B}_{\mathbf{s}}$, then we enable a simple path (i.e., with no cycles) from a starting state to the state v. The size of $|\mathcal{M}[v, 1]|$ is stored in $\mathcal{M}[v, 2]$ which also indicates that the node v is reachable when $\mathcal{M}[v, 2] < \infty$.

The algorithm works by maintaining a queue with the unvisited nodes on the graph. Each node v in the queue has as key the number of atomic propositions that must be removed so that v becomes reachable on $\mathcal{A}$. The algorithm proceeds by

Algorithm 2: FindMinPath($G_{\mathcal{A}}$,$\mathcal{M}$,*lasso*)

```
Input: a graph $G_{\mathcal{A}} = (V, E, v_s, V_f, \overline{\Pi}, \Lambda)$, a table $\mathcal{M}$ and a flag $lasso$ on
       whether this is a lasso path search
Output: the tables $\mathcal{M}$ and $\mathbf{P}$ and the visited nodes $\mathcal{V}$
Variables: (a queue $\mathcal{Q}$, a set $\mathcal{V}$ of visited nodes and a table $\mathbf{P}$ indicating the
  parent of each node on a path)
1  $\mathcal{V} \leftarrow \{v_s\}$
2  $\mathbf{P}[:] \leftarrow \emptyset$                                  ▷ Each entry of $\mathbf{P}$ is set to $\emptyset$
3  $\mathcal{Q} \leftarrow V - \{v_s\}$
4  for $v \in V$ such that $(v_s, v) \in E$ and $v \neq v_s$ do
5      $\langle \mathcal{M}, \mathbf{P} \rangle \leftarrow$ Relax$((v_s, v), \mathcal{M}, \mathbf{P}, \Lambda)$
6  if $lasso = 1$ then
7      if $(v_s, v_s) \in E$ then
8          $\mathcal{M}[v_s, 1] \leftarrow \mathcal{M}[v_s, 1] \cup \Lambda(v_s, v_s)$
9          $\mathcal{M}[v_s, 2] \leftarrow |\mathcal{M}[v_s, 1] \cup \Lambda(v_s, v_s)|$
10         $\mathbf{P}[v_s] \leftarrow v_s$
11     else
12         $\mathcal{M}[v_s, :] \leftarrow (\overline{\Pi}, \infty)$
13 while $\mathcal{Q} \neq \emptyset$ do
14     $u \leftarrow$ ExtractMIN$(\mathcal{Q})$                         ▷ Get node $u$ with minimum $\mathcal{M}[u, 2]$
15     if $\mathcal{M}[u, 2] < \infty$ then
16         $\mathcal{V} \leftarrow \mathcal{V} \cup \{u\}$
17         for $v \in V$ such that $(u, v) \in E$ do
18             $\langle \mathcal{M}, \mathbf{P} \rangle \leftarrow$ Relax$((u, v), \mathcal{M}, \mathbf{P}, \Lambda)$
19 return $\mathcal{M}, \mathbf{P}, \mathcal{V}$
```

choosing the node with the minimum number of atomic propositions discovered so far (line 14). Then, this node is used in order to updated the estimates for the minimum number of atomic propositions needed in order to reach its neighbors (line 18). A notable difference of Alg. 2 from Dijkstra's shortest path algorithm is the check for lasso paths in lines 6-12. After the source node is used for updating the estimates of its neighbors, its own estimate for the minimum number of atomic propositions is updated either to the value indicated by the self loop or the maximum possible number of atomic propositions. This is required in order to compare the different paths that reach a node from itself.

Algorithm 3: RELAX((u, v),$\mathcal{M}$,$\mathbf{P}$,Λ)

```
  Input: an edge $(u, v)$, the tables $\mathcal{M}$ and $\mathbf{P}$ and the edge labeling function $\Lambda$
  Output: the tables $\mathcal{M}$ and $\mathbf{P}$
1 if $|\mathcal{M}[u, 1] \cup \Lambda(u, v)| < \mathcal{M}[v, 2]$ then
2     $\mathcal{M}[v, 1] \leftarrow \mathcal{M}[u, 1] \cup \Lambda(u, v)$
3     $\mathcal{M}[v, 2] \leftarrow |\mathcal{M}[u, 1] \cup \Lambda(u, v)|$
4     $\mathbf{P}[v] \leftarrow u$
5 return $\mathcal{M}$, $\mathbf{P}$
```

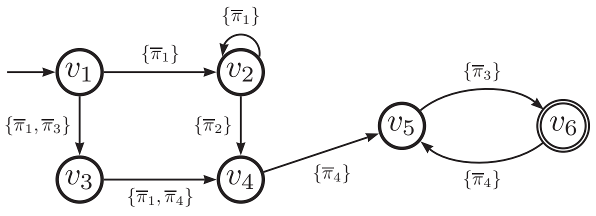


Figure 2.4: The graph of Example 5. The source $v_s = v_1$ is denoted by an arrow and the sink v_6 by double circle ($V_f = \{v_6\}$).

The following example demonstrates how the algorithm works and indicates the structural conditions on the graph that make the algorithm non-optimal.

Example 5. *Let us consider the graph in Fig. 2.4. The source node of this graph is $v_s = v_1$ and the set of sink nodes is $V_f = \{v_6\}$. The $\overline{\Pi}$ set of this graph is $\{\overline{\pi}_1, \ldots, \overline{\pi}_4\}$. Consider the first call of* FINDMINPATH *(line 4 of Alg. 1).*

- *Before the first execution of the while loop (line 13): The queue contains $\mathcal{Q} = \{v_2, \ldots, v_6\}$. The table $\mathcal{M}$ has the following entries: $\mathcal{M}[v_1, :] = \langle \emptyset, 0 \rangle$, $\mathcal{M}[v_2, :] = \langle \{\overline{\pi}_1\}, 1 \rangle$, $\mathcal{M}[v_3, :] = \langle \{\overline{\pi}_1, \overline{\pi}_3\}, 2 \rangle$, $\mathcal{M}[v_4, :] = \ldots = \mathcal{M}[v_6, :] = \langle \overline{\Pi}, \infty \rangle$.*

- *Before the second execution of the while loop (line 13): The node v_2 was popped from the queue since it had $\mathcal{M}[v_2, 2] = 1$. The queue now contains $\mathcal{Q} = \{v_3, \ldots, v_6\}$. The table $\mathcal{M}$ has the following rows: $\mathcal{M}[v_1, :] = \langle \emptyset, 1 \rangle$, $\mathcal{M}[v_2, :] = \langle \{\overline{\pi}_1\}, 1 \rangle$, $\mathcal{M}[v_3, :] = \langle \{\overline{\pi}_1, \overline{\pi}_3\}, 2 \rangle$, $\mathcal{M}[v_4] = \langle \{\overline{\pi}_1, \overline{\pi}_2\}, 2 \rangle$, $\mathcal{M}[v_5, :] = \mathcal{M}[v_6, :] = \langle \overline{\Pi}, \infty \rangle$.*

- *At the end of* FINDMINPATH *(line 19): The queue now is empty. The table* $\mathcal{M}$ *has the following rows:* $\mathcal{M}[v_1,:] = \langle \emptyset, 0 \rangle$, $\mathcal{M}[v_2,:] = \langle \{\overline{\pi}_1\}, 1 \rangle$, $\mathcal{M}[v_3,:] = \langle \{\overline{\pi}_1, \overline{\pi}_3\}, 2 \rangle$, $\mathcal{M}[v_4,:] = \langle \{\overline{\pi}_1, \overline{\pi}_2\}, 2 \rangle$, $\mathcal{M}[v_5,:] = \langle \{\overline{\pi}_1, \overline{\pi}_2, \overline{\pi}_4\}, 3 \rangle$, $\mathcal{M}[v_6,:] = \left\langle \overline{\Pi}, 4 \right\rangle$, *which corresponds to the path* v_1, v_2, v_4, v_5, v_6.

Note that algorithm returns a set of atomic propositions $L' = \overline{\Pi}$ *which is not optimal* $|L'| = 4$*). The path* v_1, v_3, v_4, v_5, v_6 *would return* $L' = \{\overline{\pi}_1, \overline{\pi}_3, \overline{\pi}_4\}$ *with* $|L'|$ $= 3$. $\triangle$

Correctness: The correctness of the algorithm AAMRP is based upon the fact that a node $v \in V$ is reachable on $G_{\mathcal{A}}$ if and only if $\mathcal{M}[v, 2] < \infty$. The argument for this claim is similar to the proof of correctness of Dijkstra's shortest path algorithm in [77]. If this algorithm returns a set of atomic propositions L which removed from $\mathcal{B}_{\mathbf{s}}$, then the language $\mathcal{L}(\mathcal{A})$ is non-empty. This is immediate by the construction of the graph $G_{\mathcal{A}}$ (Def. 19).

We remark that AAMRP does not solve Problem 1 exactly since MRP is NP-Complete. However, AAMRP guarantees that it returns a valid relaxation $\mathcal{B}$ where $\mathcal{B}_{\mathbf{s}} \preceq \mathcal{B}$.

Theorem 2. *If a valid relaxation exists, then AAMRP always returns a valid relaxation* $\mathcal{B}$ *of some initial* $\mathcal{B}_{\mathbf{s}}$ *such that* $\mathcal{L}(\mathcal{T} \times \mathcal{B}) \neq \emptyset$.

Proof. First, we will show that if AAMRP returns $\emptyset$, then there is no valid relaxation of $\mathcal{B}_{\mathbf{s}}$. AAMRP returns $\emptyset$ when there is no reachable $v_f \in V_f$ with prefix and lasso path or GETAPFROMPATH returns $\emptyset$. If there is no reachable v_f, then either the accepting state is not reachable on $\mathcal{B}_{\mathbf{s}}$ or on $\mathcal{T}$. Recall that the Def. 19 constructs a graph where all the transitions of $\mathcal{T}$ and $\mathcal{B}$ are possible. If it returns $\emptyset$ as a valid solution, then there is a path on the graph that does not utilize any labeled edge by Λ. Thus, $\mathcal{L}(\mathcal{T} \times \mathcal{B}_{\mathbf{s}}) \neq \emptyset$. Since we assume that $\mathcal{B}_{\mathbf{s}}$ is unsatisfiable on $\mathcal{T}$, this is contradiction.

Second, without loss of generality, suppose that AAMRP returns $\widetilde{\Lambda}(\mu)$. Using this $\widetilde{\Lambda}(\mu)$, we can build a relax specification automaton $\mathcal{B}$. Using each $\langle l, (s, s'), k\rangle \in \widetilde{\Lambda}(\mu)$ and for each $\pi \in l$, we add the indices of the literal ϕ_{ij} in $\Phi_{\mathcal{B}_{\mathbf{s}}(s,s')}$ that corresponds to π to the sets $\hat{D}_{ss'}$ and $\hat{C}^{i}_{ss'}$. The resulting substitution θ produces a relaxation. Moreover, it is a valid relaxation, because by removing the atomic propositions in θ from $\mathcal{B}_{\mathbf{s}}$, we get a path that satisfies the prefix and lasso components on the product automaton. $\square$

Running time: The running time analysis of the AAMRP is similar to that of Dijkstra's shortest path algorithm. In the following, we will abuse notation when we use the O notation and treat each set symbol S as its cardinality $|S|$.

First, we will consider FindMinPath. The fundamental difference of AAMRP over Dijkstra's algorithm is that we have set theoretic operations. We will assume that we are using a data structure for sets that supports $O(1)$ set cardinality quarries, $O(\log n)$ membership quarries and element insertions (see [77]) and $O(n)$ set up time. Under the assumption that $\mathcal{Q}$ is implemented in such a data structure, each ExtractMIN takes $O(\log V)$ time. Furthermore, we have $O(V)$ such operations (actually $|V| - 1$) for a total of $O(V \log V)$.

Setting up the data structure for $\mathcal{Q}$ will take $O(V)$ time. Furthermore, in the worst case, we have a set $\Lambda(e)$ for each edge $e \in E$ with set-up time $O(E\overline{\Pi})$. Note that the initialization of $\mathcal{M}[v, :]$ to $\left\langle\overline{\Pi}, \infty\right\rangle$ does not have to be implemented since we can have indicator variables indicating when a set is supposed to contain all the (known in advance) elements.

Assuming that E is stored in an adjacency list, the total number of calls to RELAX at lines 4 and 17 of Alg. 2 will be $O(E)$ times. Each call to RELAX will have to perform a union of two sets ($\mathcal{M}[u, 1]$ and $\Lambda(u, v)$). Assuming that both sets have in the worst case $|\overline{\Pi}|$ elements, each union will take $O(\overline{\Pi} \log \overline{\Pi})$ time. Finally, each set size quarry takes $O(1)$ time and updating the keys in $\mathcal{Q}$ takes $O(\log V)$ time. Therefore, the running time of FINDMINPATH is $O(V + E\overline{\Pi} + V \log V + E(\overline{\Pi} \log \overline{\Pi} + \log V))$.

Note that even if under Assumption 1 all nodes of $\mathcal{T}$ are reachable ($|V| < |E|$), the same property does not hold for the product automaton. (e.g, think of an environment $\mathcal{T}$ and a specification automaton whose graphs are Directed Acyclic Graphs (DAG). However, even in this case, we have ($|V| < |E|$). The running time of FINDMINPATH is $O(E(\overline{\Pi} \log \overline{\Pi} + \log V))$. Therefore, we observe that the running time also depends on the size of the set $\overline{\Pi}$. However, such a bound is very pessimistic since not all the edges will be disabled on $\mathcal{A}$ and, moreover, most edges will not have the whole set $\overline{\Pi}$ as candidates for removal.

Finally, we consider AAMRP. The loop at line 8 is going to be called $O(V_f)$ times. At each iteration, FINDMINPATH is called. Furthermore, each call to GETAPFROMPATH is going to take $O(V\overline{\Pi} \log \overline{\Pi})$ time (in the worst case we are going to have $|V|$ unions of sets of atomic propositions). Therefore, the running time of AAMRP is $O(V_f(V\overline{\Pi} \log \overline{\Pi} + E(\overline{\Pi} \log \overline{\Pi} + \log V))) = O(V_f E(\overline{\Pi} \log \overline{\Pi} + \log V))$ which is polynomial in the size of the input graph.

Approximation bound: AAMRP does not have a constant approximation ratio on arbitrary graphs.

Example 6 (Unbounded Approximation)**.** *The graph in Fig. 2.5 is the product of a specification automaton with a single state and a self transition with label* $\{\overline{\pi}_0, \overline{\pi}_1, \ldots, \overline{\pi}_m, \overline{\pi}_\bigstar, \overline{\pi}_\clubsuit\}$ *and an environment automaton with the same structure as the graph in Fig. 2.5 but with appropriately defined state labels. In this graph, AAMRP*

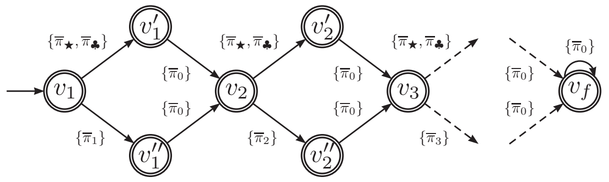


Figure 2.5: The graph of Example 6. The source $v_s = v_1$ is denoted by an arrow and the sink v_f by double circle ($V_f = \{v_f\}$).

will choose the path v_1,v_1'', v_2, v_2'', v_3, $\ldots$, v_f. *The corresponding revision will be the set of atomic propositions* $L_p = \{\overline{\pi}_0, \overline{\pi}_1, \overline{\pi}_2, \ldots, \overline{\pi}_m\}$ *with* $|L_p| = m + 1$. *This is because in* v_2, *AAMRP will choose the path through* v_1'' *rather than* v_1' *since the latter will produce a revision set of size* $|\{\overline{\pi}_0, \overline{\pi}_\bigstar, \overline{\pi}_\clubsuit\}| = 3$ *while the former a revision set of size* $|\{\overline{\pi}_0, \overline{\pi}_1\}| = 2$. *Similarly at the next junction node* v_3, *the two candidate revision sets* $\{\overline{\pi}_0, \overline{\pi}_1, \overline{\pi}_\bigstar, \overline{\pi}_\clubsuit\}$ *and* $\{\overline{\pi}_0, \overline{\pi}_1, \overline{\pi}_2\}$ *have sizes 4 and 3, respectively. Therefore, the algorithm will always choose the path through the nodes* v_i'' *rather than* v_i' *producing, thus, a solution of size* $m + 1$. *However, in this graph, the optimal revision would have been* $L_p = \{\overline{\pi}_0, \overline{\pi}_\bigstar, \overline{\pi}_\clubsuit\}$ *with* $|L_p| = 3$. *Hence, we can see that in this example for* $m > 2$ *AAMRP returns a solution which is* $m - 2$ *times bigger than the optimal solution.* $\triangle$

There is also a special case where AAMRP returns a solution whose size is at most twice the size of the optimal solution.

Theorem 3. *AAMRP on planar Directed Acyclic Graphs (DAG) where all the paths merge on the same node is a 2-approximation algorithm.*

The proof is provided in the Appendix B.

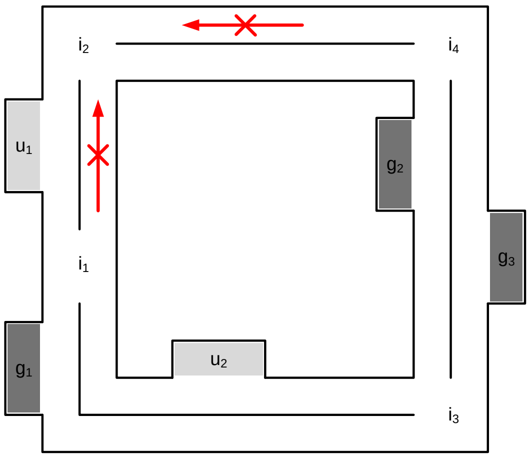


Figure 2.6: Schematic illustration of the simple road network environment of Example 13. The robot is required to drive right-side of the road.

2.3.3 Result

In this section, we present experimental results using our prototype implementation of AAMRP. The prototype implementation is in Python (see [79]). Therefore, we expect the running times to substantially improve with a C implementation using state-of-the-art data structure implementations.

We first present some examples and expand few more example scenarios. With our prototype implementation, we could expand our experiment to few more example scenarios introduced in [13, 80].

Example 7 (Single Robot Data Gathering Task)**.** *In this example, we use a simplified road network having three gathering locations and two upload locations with four intersections of the road. In Fig. 2.12, the data gather locations, which are labeled g_1, g_2, and g_3, are dark gray, the data upload locations, which are labeled u_1 and u_2, are light gray, and the intersections are labeled i_1 through i_4. In order to gather data and upload the gather-data persistently, the following LTL formula may be considered: ϕ_A $:= GF(\varphi_g) \wedge GF(\pi)$, where $\varphi_g := g_1 \vee g_2 \vee g_3$ and $\pi := u_1 \vee u_2$. The following formula* *can make the robot move from gather locations to upload locations after gathering data:* $\phi_G := G(\varphi_g \rightarrow X(\neg\varphi_g \mathcal{U} \pi)$*. In order for the robot to move to gather location after uploading, the following formula is needed:* $\phi_U := G(\pi \rightarrow X(\neg\pi \mathcal{U} \varphi_g)$*.*

Let us consider that some parts of road are not recommended to drive from gather locations, such as from i_4 *to* i_2 *and from* i_1 *to* i_2*. We can describe those constraints as following:* $\psi_1 := G(g_1 \rightarrow \neg(i_4 \wedge Xi_2)\mathcal{U}u_1)$ *and* $\psi_2 := G(g_2 \rightarrow \neg(i_1 \wedge Xi_2)\mathcal{U}u_2)$*. If the gathering task should have an order such as* g_3, g_1, g_2, g_3, g_1, g_2, $\ldots$*, then the following formula could be considered:* $\phi_O := ((\neg g_1 \wedge \neg g_2)\mathcal{U}g_3) \wedge G(g_3 \rightarrow X((\neg g_2 \wedge \neg g_3)\mathcal{U}g_1)) \wedge G(g_1 \rightarrow X((\neg g_1 \wedge \neg g_3)\mathcal{U}g_2)) \wedge G(g_2 \rightarrow X((\neg g_1 \wedge \neg g_2)\mathcal{U}g_3))$*. Now, we can informally describe the mission. The mission is "Always gather data from g3, g1, g2 in this order and upload the collected data to* u_1 *and* u_2*. Once data gathering is finished, do not visit gather locations until the data is uploaded. Once uploading is finished, do not visit upload locations until gathering data. You should always avoid the road from* i_4 *to* i_2 *when you head to* u_1 *from* g_1 *and the road from* i_1 *to* i_2 *when you head to* u_2 *from* g_2*". The following formula represents this mission:*

$$\phi_{single} := \phi_O \wedge \phi_G \wedge \phi_U \wedge \psi_1 \wedge \psi_2 \wedge\ GF(\pi).$$

Assume that initially, the robot is in i_3 *and all nodes are final nodes. When we made a cross product with the road and the specification, we could get 36824 states, 350114 edges, and 450 final states. Not removing some atomic propositions, the specification was not satisfiable. AAMRP took 15 min 34.572 seconds, and suggested removing* g_3*. Since the original specification has many* g_3 *in it, we had to trace which* g_3 *from the specification should be removed. Hence, we revised the LTL2BA in [81], indexing each atomic proposition on the transitions and states (see [82]).Two* g_3 *are mappped to the same transition on the specification automaton in* $(\neg g_1 \wedge \neg g_2)\mathcal{U}g_3$ *of* ϕ_O *and in* $\varphi_g := g_1 \vee g_2 \vee g_3$ *in* ϕ_U*.* △

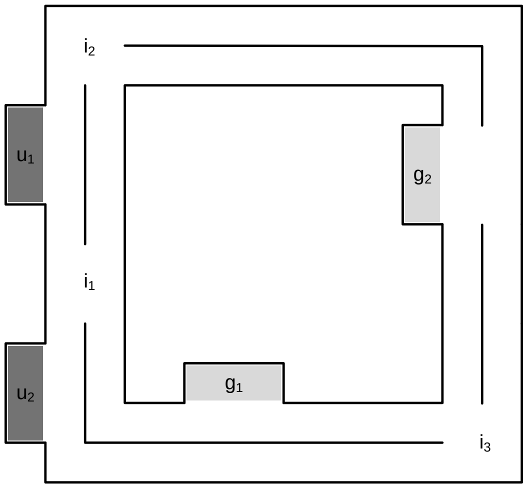


Figure 2.7: Schematic illustration of the simple road network environment of Example 8. The robots can stay upload locations u_1 and u_2 to recharge the battery.

The last example shows somewhat different missions with multiple robots. If the robots execute the gather and upload mission, persistently, we could assume that the battery in the robots should be recharged.

Example 8 (Charging while Uploading)**.** *In this exaple, we assume that robots can recharge their battery in upload locations so that robots are reqired to stay at the upload locations as much as possible. We also assume that each gathering localtion has a dedicated upload location such that g_1 has u_1 as an upload location, and g_2 has u_2 as an upload location. For this example, we revised the road network so that we remove the gather location g_3 and the intersection i_4 to make the network simpler for this mission. We also positioned the upload locations next to each other. We assume that the power source is shared and it has just two charging statations (see in Fig. 12). We can describe the mission as follows: "Once robot$_1$ finishes gathering data at g_1, robot$_1$ should not visit the gather locations until the data is uploaded at u_1. Once robot$_2$ fisniehs gathering data at g_2, robot$_2$ shoud not visit the gather locations until the data is uploaded at u_2. Once the data is uploaded at u_1 or u_2, robot$_1$ or robot$_2$ should stay* *there until a gather locaiton is not occupied. Persistently, gather data from g_1 and g_2, avoiding the road from g_2 to i_2." The following formula represents this mission:*

$$\begin{aligned}
\phi_{charging} := {} & G(g_{11} \rightarrow X(\neg g_{11} \wedge \neg g_{21})\ \mathcal{U} u_{11}) \wedge \\
& G(g_{22} \rightarrow X(\neg g_{22} \wedge \neg g_{12})\ \mathcal{U} u_{22}) \wedge \\
& G(u_{11} \rightarrow u_{11}\ \mathcal{U}\ \neg g_{22}) \wedge \\
& G(u_{22} \rightarrow u_{22}\ \mathcal{U}\ \neg g_{11}) \wedge \\
& GFg_{11} \wedge GFg_{22} \wedge \\
& G\neg(g_{21} \wedge Xi_{21}) \wedge \\
& G\neg(g_{22} \wedge Xi_{22}).
\end{aligned}$$

Assume that initially, $robot_1$ is in i_1, $robot_2$ is in i_2, and all nodes are final nodes. From the cross product with the road and the specification, there was 65966 states, 253882 transitions, and 504 final nodes. For this example, we computed a synchronized environtment for two robots, and in this environment, atomic propositions were duplicationed for each robot. For example, a gather location g_1 is duplicated to g_{11} for $robot_1$ and g_{12} for $robot_2$. With this synchronized environment, we could avoid robots to be colliding and to be in the same location at the same time. However, not removing some atomic propositions, the specification was unsatisfiable. AAMRP took 24 min 22.578 seconds, and suggested removing u_{22} from $robot_2$. The two occurances of u_{22} were in $G(g_{22} \rightarrow X(\neg g_{22} \wedge \neg g_{12})\ \mathcal{U} u_{22})$ and in the second u_{22} of $G(u_{22} \rightarrow u_{22}\ \mathcal{U}\ \neg g_{11})$ as indicated by our modified LTL2BA toolbox. The suggested path from AAMRP for each robot is as followings:

$$path_{robot_1} = i_{11}i_{21}u_{11}u_{11}i_{11}(i_{21}g_{21}i_{31}g_{11}i_{11}i_{21}u_{11}u_{11}u_{11}u_{11}u_{11}u_{11}u_{11}u_{11}u_{11}i_{11})^{+}$$

$$path_{robot_2} = i_{22}u_{12}i_{12}u_{22}u_{22}(u_{22}u_{22}u_{22}u_{22}u_{22}u_{22}u_{22}i_{32}g_{12}i_{12}i_{22}g_{22}i_{32}g_{12}i_{12}u_{22})^{+}$$

△

Nodes	BRUTE-FORCE SEARCH							AAMRP							RATIO		
	TIMES (SEC)			SOLUTIONS (SIZE)				TIMES (SEC)			SOLUTIONS (SIZE)						
	min	avg	max	min	avg	max	succ	min	avg	max	min	avg	max	succ	min	avg	max
9	0.037	0.104	1.91	1	1.97	5	200/200	0.022	0.061	1.17	1	1.975	5	200/200	1	1.0016	1.333
100	0.069	510.18	20786	1	3.277	13	198/200	0.038	0.076	0.179	1	3.395	15	200/200	1	1.0006	1.125
196	0.066	1025.44	25271	1	3.076	8	171/200	0.007	0.188	0.333	1	4.285	17	200/200	1	1	1
324	0.103	992.68	25437	1	2.379	6	158/200	0.129	0.669	1.591	1	4.155	20	200/200	1	1	1.2
400	0.087	1110.05	17685	1	2.692	6	143/200	0.15	0.669	1.591	1	5	24	200/200	1	1	1
529	0.14	2153.90	26895	1	2.591	5	137/200	0.382	1.88	4.705	1	5.115	30	200/200	1	1	1

Table 2.1: Numerical Experiments: Number of nodes versus the results of brute-force search and AAMRP. Under the brute-force search and AAMRP columns the numbers indicate computation times in sec. RATIO indicates the experimentally observed approximation ratio to the optimal solution.

For the experiments, we utilized the ASU super computing center which consists of clusters of Dual 4-core processors, 16 GB Intel(R) Xeon(R) CPU X5355 @2.66 Ghz. Our implementation does not utilize the parallel architecture. The clusters were used to run the many different test cases in parallel on a single core. The operating system is CentOS release 5.5.

In order to assess the experimental approximation ratio of AAMRP, we compared the solutions returned by AAMRP with the brute-force search. The brute-force search is guaranteed to return a minimal solution to the MRP problem.

We performed a large number of experimental comparisons on random benchmark instances of various sizes. We used the same instances which were presented in [68, 69]. The first experiment involved randomly generated DAGs. Each test case consisted of two randomly generated DAGs which represented an environment and a specification. Both graphs have self-loops on their leaves so that a feasible lasso path can be found. The number of atomic propositions in each instance was equal to four times the number of nodes in each acyclic graph. For example, in the benchmark where the

Nodes	AAMRP			
	TIMES			
	min	avg	max	succ
1024	0.125	0.23	0.325	9/10
10000	15.723	76.164	128.471	9/10
20164	50.325	570.737	1009.675	8/10
50176	425.362	1993.449	4013.717	3/10
60025	6734.133	6917.094	7100.055	2/10

Table 2.2: Numerical Experiments: Number of nodes versus the results of AAMRP. Under the TIMES columns the numbers indicate computation times in sec.

graph had 9 nodes, each DAG had 3 nodes, and the number of atomic propositions was 12. The final nodes are chosen randomly and they represent 5%-40% of the nodes. The number of edges in most instances were 2-3 times more than the number of nodes.

Table 2.3 compares the results of the brute-force search with the results of AAMRP on test cases of different sizes (total number of nodes). For each graph size, we performed 200 tests and we report minimum, average and maximum computation times in second and minimum, average and maximum numbers of atomic propositions for each instance solution. AAMRP was able to finish the computation and returned a minimal revision for all the test cases, but brute-force search was not able to finish all the computation within a 8 hours window.

Our brute-force search checks all the combinations of atomic propositions. For example, given n atomic propositions, it checks at most 2^n cases. It uses breath first search to check the reachability for the prefix and the lasso part. If it is reachable with the chosen atomic propositions, then it is finished. If it is not reachable, then it chooses another combination until it is reachable. Since brute-force search checks all the combinations of atomic propositions, the success mostly depends on the time limit of the test. We remark that the brute-force search was not able to provide an answer to all the test cases within a 8 hours window. The comparison for the approximation ratio was possible only for the test cases where brute-force search successfully completed the computation. Note that in the case of 529 Nodes, even though the maximum RATIO is 1, the maximum solution from brute-force does not match with the maximum solution from AAMRP. One is 5 and another is 30. This is because the number of success from brute-force search is 137 / 200 and only comparing this success with the ones from AAMRP, the maximum RATIO is still 1.

An interesting observation is that the maximum approximation ratio is experimentally determined to be less than 2. For the randomly generated graphs that we have constructed the bound apppears to be 1.333. However, as we showed in the example 6, it is not easy to construct random examples that produce higher approximation ratios. Such example scenarios must be carefully constructed in advance.

In the second numerical experiment, we attempted to determine the problem sizes that our prototype implementation of AAMRP in Python can handle. The results are presented in Table 2.2. We observe that approximately 60,025 nodes would be the limit of the AAMRP implementation in Python.

2.4 Weighted LTL Revision Problem

The specification revision problem concerns the search for one or more specifications which are related to the initial user requirement and, which, furthermore, can be satisfied on the weighted transition system under some hard cost constraints. One of the fundamental questions regards the form of the search space of the specifications. Since the initial user specification allows only system behaviors with higher cost than the constraint, it is natural to relax some of the requirements in order to permit more behaviors and, hopefully, find a behavior with lower cost.

The unconstrained LTL formula search space for a revised specification is

$$\mathfrak{F}_{\mathcal{D}}^{C} = \{\phi \in LTL(\Pi) \mid \exists \bar{p}.\bar{p} \in \mathcal{L}(\mathcal{D} \times \mathcal{B}_{\phi}) \wedge \mathcal{C}(p) \leq C\},$$

where $\mathcal{C}$ is $\mathcal{C}_1$ or $\mathcal{C}_2$. However, as we have demonstrated through examples in [17] for the unweighted version of the problem, $\mathfrak{F}_{\mathcal{D}}^{C}$ also contains specifications that from the user perspective cannot be considered valid specification revisions. Thus, we must impose some constraints on the search space. First, we define an ordering relation over the set of LTL formulas.

Definition 12. *Let ϕ, $\psi \in LTL(\Pi)$, then we define $\phi \preceq \psi$ if and only if $\mathcal{L}(\phi) \subseteq \mathcal{L}(\psi)$.*

We define the set of *ultra relaxations* as follows.

Definition 13 (Ultra Relaxation)**.** *Let $\phi \in LTL(\Pi)$, the set $\mathfrak{UR}(\phi)$ of all valid formula relaxations of ϕ can be constructed using the recursive operator $\mathbf{rel}(\phi)$ as follows:*

$$\mathbf{rel}(\pi) \in \{\pi, \top\}, \qquad OP_1\phi = \ OP_1\mathbf{rel}(\phi)$$
$$\mathbf{rel}(\phi_1 \ OP_2\phi_2) = \mathbf{rel}(\phi_1) \ OP_2\mathbf{rel}(\phi_2)$$

where OP_1 is any unary and OP_2 is any binary operator.

Informally, a valid formula relaxation is one that recursively relaxes each atomic proposition π of the initial specification ϕ. Then, the following result is immediate.

Corollary 1. *For any $\phi \in LTL(\Pi)$ and $\phi' \in \mathfrak{UR}(\phi)$, we have $\phi \preceq \phi'$.*

Problem 2 (Weighted LTL Revision Problem)**.** *Given a system $\mathcal{T}$, a specification ϕ and a cost $C \in \mathbb{R}_{\geq 0}$ such that $\phi \notin \mathfrak{F}_{\mathcal{D}}^{C}$, find $\varphi \in \mathfrak{F}_{\mathcal{D}}^{C} \cap \mathfrak{UR}(\phi)$ such that for any other $\psi \in \mathfrak{F}_{\mathcal{D}}^{C} \cap \mathfrak{UR}(\phi)$, we have $\psi \npreceq \varphi$.*

Obviously, with the aforementioned restrictions the WMRP is decidable. For a formula ϕ, there is a finite number of revisions in $\mathfrak{U}\mathfrak{R}(\phi)$ that we must consider. For each $\phi' \in \mathfrak{U}\mathfrak{R}(\phi)$, we can solve the optimal path planning problem and, then, choose the revised specification with the least modifications that produces optimal paths with cost less than the bound C. Nevertheless, typical examples of LTL specifications can have 10-30 occurrences of atomic propositions in a formula (see [78] for an interesting collection). This means that the optimal LTL planning problem (which includes the Büchi automaton synthesis for each new formula) must be solved anywhere from 1,000 times to 1 billion times. Next, we study the question whether the problem really requires exploring all the combinations for the optimal solution.

2.4.1 LTL Revision as a Shortest Path Problem

In this section, we present how Problem 2 can be posed as a shortest path problem on a weighted labeled graph.

Without loss of generality, we will assume that for any given specification ϕ, each atomic proposition π in ϕ appears only once in ϕ. If this is not the case, then for each additional occurrence of π in ϕ, we can replace it with a new atomic proposition, add the proposition to Π and modify the map $h_{\mathcal{T}}$ accordingly. This change is necessary in order to uniquely identify in ϕ which propositions need to be replaced by $\top$.

Given a specification ϕ, we can construct the corresponding specification automaton $\mathcal{B}_{\mathbf{s}}$. Using the weighted labeled transition system $\mathcal{T}$ and the specification automaton $\mathcal{B}_{\mathbf{s}}$, we can construct a graph $G_{\mathcal{A}}$ which corresponds to the product automaton $\mathcal{A}$ while considering the effect of revisions and the weights.

Definition 14. *Given a system $\mathcal{T}$ and a specification automaton $\mathcal{B}_{\mathbf{s}}$, we define the graph $G_{\mathcal{A}} = (V, E, v_s, V_f, W, L, R)$, which corresponds to the product $\mathcal{A} = \mathcal{T} \times \mathcal{B}_{\mathbf{s}}$ as*

- *$V = \mathcal{S}_{\mathcal{A}}$ is the set of nodes*

- *$E = E_{\mathcal{A}} \cup E_D \subseteq \mathcal{S}_{\mathcal{A}} \times \mathcal{S}_{\mathcal{A}}$, where*

  - *$E_{\mathcal{A}}$ is the set of edges that correspond to transitions on $\mathcal{A}$, i.e., $((q,s),(q',s')) \in E_{\mathcal{A}}$ iff $\exists l \in \mathcal{P}(\Pi)\ .\ (q,s) \xrightarrow{l}_{\mathcal{A}} (q',s')$; and*

  - *E_D is the set of edges that correspond to disabled transitions, i.e., $((q,s),(q',s')) \in E_D$ iff $q \rightarrow_{\mathcal{T}} q'$ and $s \xrightarrow{l}_{\mathcal{B}_\mathbf{s}} s'$ with $l \cap (\Pi - h_{\mathcal{T}}(q')) \neq \emptyset$.*

- *$v_s = s_0^{\mathcal{A}}$ is the source node,*

- *$V_f = F_{\mathcal{A}}$ is the set of sinks,*

- *$W : E \rightarrow \mathbb{R}_{\geq 0}$ assigns a weight to each edge in E. If $e = ((q,s),(q',s')) \in E$, then $w(e) = w(q)$.*

- *$L : E \rightarrow \mathcal{P}(\Pi)$ maps each edge of the graph to the set of symbols that need to be removed in order to enable the edge in the product automaton $\mathcal{A}$. If $e = ((q,s),(q',s')) \in E$, then $L(e) = \lambda_{\mathcal{B}_\mathbf{s}}(s,s') - h_{\mathcal{T}}(q')$.*

- *$R = \{(q,s) \in V \mid \pi^\star \in h_{\mathcal{T}}(q)\}$ is the set of "cost reset" nodes.*

We remark that the graph $G_{\mathcal{A}}$ is essentially the same graph as the graph of $\mathcal{A}$ with the addition of the disabled edges due to the specification constraints. Therefore, any path on the graph of $\mathcal{A}$ appears as a path on $G_{\mathcal{A}}$.

A path $\eta = v_0 v_1 v_2 \ldots v_n$ on $G_{\mathcal{A}}$ is a sequence of nodes that start from the source $v_0 = v_s$, follow the edges from E, i.e., for $0 \leq i < n$, $(v_i, v_{i+1}) \in E$, and end in one of the sinks $v_n \in V_f$. The cost of the corresponding path is defined as $C_{G_{\mathcal{A}}}(\eta) = \langle |L(\eta)|, W(\eta) \rangle$. $L(\eta) = \bigcup_{i=0}^{|\eta|-1} L(v_i, v_{i+1})$ is the set of symbols that need to be removed for the path to become enabled on $\mathcal{A}$. $W(\eta) = \max_{j=0,1,\ldots,k-1} \sum_{i=r_j}^{r_{j+1}} W(v_i, v_{i+1})$ is the weight of the path after $k-1$ cost resets by visiting a node in R. Here, $r_0 r_1 \dots r_k$ with $r_i \in \{0, 1, \dots, |\eta|-1\}$ is the sequence of indices such that $v_{r_i} \in R$, $r_0 = 0$ and $r_k = |\eta| - 1$. In the special case where there are no cost resets, i.e., $k = 1$, then $W(\eta) = \sum_{i=0}^{|\eta|-1} W(v_i, v_{i+1})$. Then, by construction, Problem 2 is reduced to solving a number of the following problems.

Problem 3. *Given a weighted labeled graph $G_{\mathcal{A}}$ as in Def. 19 and a cost bound $K \in \mathbb{R}_{\geq 0}$, find a path η on the graph such that $|L(\eta)|$ is minimum over all paths in $G_{\mathcal{A}}$ while $W(\eta) \leq K$.*

We refer to the last problem as Constrained Minimally Labeled Path (CMLP). It is easy to show that the corresponding decision problem is NP-complete.

Theorem 4. *Given an instance of the CMLP $(G_{\mathcal{A}}, K)$ and a bound Λ, the decision problem of whether there exists a path η such that $|L(\eta)| \leq \Lambda$ is NP-Complete.*

Sketch. Clearly, the problem is in NP since given a sequence of nodes η, we can verify in polynomial time that η is a path on $G_{\mathcal{A}}$, $W(\eta) \leq K$ and $|L(\eta)| \leq \Lambda$.

The problem is NP-hard since we can easily reduce the unweighed version of the problem (see Sec. 2.3) to this one by setting the weights of all edges equal to zero and $R = \emptyset$. $\square$

In other words, even for this simplified version of the specification revision problem, it is unlikely that there exists a polynomial time algorithm that can solve the problem. The best we can hope for is a polynomial time approximation algorithm as the one that we have presented in Sec. 2.3.2 for the unweighted version of the problem.

2.4.2 Brute-Force Search for WMRP

We first present a Brute-Force Search Algorithm for WMRP. With this algorithm, we shall compare the result of heuritics in Sec. 2.4.4. The pseudocodes are presented

Algorithm 4: BRUTE-FORCE($G_{\mathcal{A}}, k$)

```
  Input: a graph $G_{\mathcal{A}} = \langle V, E, v_s, V_f, R, W, \Lambda, \overline{\Pi} \rangle$, and a cost bound $k \in \mathbb{R}_{\geq 0}$
  Output: a set of atomic propositions $\overline{\Pi}' \subseteq \overline{\Pi}$
1 $Result \leftarrow$ REACHABLE($G_{\mathcal{A}}, \overline{\Pi}, k$)
2 if $Result$ then
3     for $i = 0$ to $|\overline{\Pi}| - 1$ do
4         for $\overline{\Pi}' \subset \overline{\Pi}$ such that $|\overline{\Pi}'| = i$ do
5             $Result \leftarrow$ REACHABLE($G_{\mathcal{A}}, \overline{\Pi}', k$)
6             if $Result$ then
7                 return $\overline{\Pi}'$
8 return $\overline{\Pi}$
```

in Algorithm 4 and 5.

Given a graph $G_{\mathcal{A}}$ and a cost bound k as inputs, the main algorithm (Alg. 4) checks reachablity with all possible subset of $\overline{\Pi}$, and outputs a set of atomic propositions $\overline{\Pi}' \subseteq \overline{\Pi}$. The output $\overline{\Pi}'$ has three different meaning when it is returned. First, if $\overline{\Pi}' = \emptyset$, then it means that the original speicification is satisfiable. This is because there exists a reachable path without any atomic propoisions to be removed. Second, if $\overline{\Pi}' = \overline{\Pi}$, then it means that there is no reachable path with all possible atomic proposition to be removed or all atomic proposions should be removed to make the specification satisfiable. However, note that the former means that k is not big enough to reach all the states due to Assumption 1. Also, note that the latter means that there remains nothing in the specification ϕ if removing all atomic propositions. Third, if $0 < |\overline{\Pi}'| < |\overline{\Pi}|$, then it means that there exists a reachable path when a proper subset of $\overline{\Pi}$ is given.

The sub algorithm (Alg. 5) mainly uses Bellman-Ford's shortest path algorithm [77]. Each BELLMAN-FORD outputs a tuple of $\langle D, P \rangle$: D is a distance table or matrix and P is a predecessor table or matrix. For each vertex $v, v' \in V$, $D[v]$ has the shortest distance to reach the vertex from v_s and $D[v, v']$ has the shortest distance from v to

Algorithm 5: REACHABLE($(G_{\mathcal{A}}, \overline{\Pi}', k)$

```
    Input: a graph $G_{\mathcal{A}} = (V, E, v_s, V_f, R, W, \Lambda, \overline{\Pi})$, a set of atomic propositions
           $\overline{\Pi}'$, and a cost bound $k$
    Output: a boolean {True, False}
1   Define a graph $G$ with $V$ and $E' = \{e \in E | \Lambda(e) \subseteq \overline{\Pi}'\}$
2   Compute shortest paths for $\{v_s\} \times V$ :
        $\langle D_{sr}, P_{sr} \rangle \leftarrow$ BALLMAN-FORD$(G, D_{sr}, \{v_s\})$.
3   Compute shortest paths for $R \times V$ :
        $\langle D_{rv}, P_{rv} \rangle \leftarrow$ BALLMAN-FORD$(G, D_{rv}, R)$.
4   Compute shortest paths for $V_f \times V$ :
        $\langle D_{fv}, P_{fv} \rangle \leftarrow$ BALLMAN-FORD$(G, D_{fv}, V_f)$, and
        set $D_{rv}[v, v] = 0$ and $D_{fv}[v, v] = 0$ for all $v \in R \cap V_f$.
5   For each triple $(r_1, f, r_2) \in R \times V_f \times R$, let $C(r_1, f, r_2)$
        be $max(D_{rv}[r_1, f], D_{fv}[f, r_2], D_{rv}[r_2, r_1], D_{sr}[r_1])$, if $r_1 \neq r_2 \neq f \wedge f \in R$,
        be $max(D_{rv}[r_1, f], D_{fv}[f, r_2], D_{rv}[r_2, r_1], D_{sr}[r2])$, if $r_1 = r_2 \neq f \wedge f \in R$,
        be $max(D_{rv}[r_1, f] + D_{fv}[f, r_2], D_{sr}[r1])$, if $r_1 = r_2 \wedge f \notin R$, and
        be $max(D_{rv}[r_1, f] + D_{fv}[f, r_2], D_{rv}[r_2, r_1], D_{sr}[r1])$, otherwise.
6   Find the triple $(r_1^*, f^*, r_2^*)$ that minimizes $C(r_1, f, r_2)$
7   If minimum cost is less than or equal to $k$, then output True which means
     "reachable". Otherwise, output False which means "not reachable".
```

v'. However, for unreachable path from v_s to v or v to v', the entry of distance table or matrix has ∞. In step 2, it computes shortest paths for $\{v_s\} \times V$, and gets the adjacency list D_{sr} and the predecessor list P_{sr}. Then, in step 3 it computes shortest paths for $R \times V$, and in step 4 for $V_f \times V$. In step 4, setting $D_{rv}(v, v) = 0$ and $D_{fv}(v, v) = 0$ for all $v \in R \cap V_f$ makes the cases when $r_1 = f$ or $f = r_2$ simple. In step 6, it finds a triple (r_1^*, f^*, r_2^*) in $R \times V_f \times R$ such that from r_1^* to f^* and from f^* to r_2^* and from r_2^* to r_1^*, all cumulative weight of each path should be less than or equal to k, which means reachable. It would return True if found one. Otherwise, it would return False.

We remark that in order to use Bellman-Ford search with the set of cost reset vertices R, the original RELAX function should be redefined as following: given an edge (u, v), two tables D, P, a set of cost reset vertices R, and a weight function w,

$$\langle D[v], P[v]\rangle := \begin{cases} \langle D[u] + w(u,v), u\rangle & \text{if } u \notin R \wedge D[v] > D[u] + w(u,v), \\ \langle w(u,v), u\rangle & \text{if } u \in R \wedge D[v] > w(u,v), \\ \langle D[v], P[v]\rangle & \text{otherwise.} \end{cases}$$

Instead of using Bellman-Ford search, Dijkstra's Shortest-Path algorithm can be used. However, we carefully redefine each graph, and use single pair shortest path search. We shall use Dijkstra's algorithm for heuristics in Sec., 2.4.3.

Correctness: In each step of Alg. 5, the correctness of Bellman-Ford's shortest path algorithm has proven in [77]. Hence, in step 6, if minimum cost of a triple (r_1^*, f^*, r_2^*) is less than or equal to k, then the cumulative weights of each path from r_1^* to f^*, from f^* to r_2^*, and from r_2^* to r_1^* are less than or equal to k so that there exists a reachable path with the cost bound k. In addition, the reachable path having the triple (r_1^*, f^*, r_2^*) always visits $v_r \in R$ and $v_f \in V_f$. In Alg. 4, if there is a set of optimal solutions, it would always return one of the solutions. The algorithm checks all possible subset of $\overline{\Pi}$ and it starts from $\emptyset$ and increases the size by 1 only if there is no reachable path with every subset having the same size. Hence, it guarrantees that if found a subset $\overline{\Pi}'$ which has a reachable path, then there is no solution which is less than $|\overline{\Pi}'|$.

Running time: Algorithm 5 runs in polynomial time. Step 1 takes $O(E\overline{\Pi})$. Step 2 runs Bellman-Ford's algorithm one time, so it takes $O(EV)$ (actually, $|E'|$ not $|E|$, but $|E'|$ varies, depending on $\overline{\Pi}'$, and $|E'| \leq |E|$). Step 3 runs Bellman-Ford's algorithm $|R|$ times, taking $O(REV)$, and step 4 takes $(V_f EV)$. Step 5 and 6 take $O(V_f R^2)$. Since $|E| \leq |V|^2$ and $|R|^2 \ll |V|^2$, the run time of Alg. 5 is given by $O((V_f + R)EV + E\overline{\Pi})$. In Alg. 4, Reachable is executed $2^{|\overline{\Pi}|}$ times at line 1 and 5. Therefore, the run time is $O(2^{|\overline{\Pi}|}((V_f + R)EV + E\overline{\Pi}))$, which is exponential in the size of the input graph.

2.4.3 Two heuristics for WMRP

In this section, we present two heuristics for the Weighted Minimal Revision Problem (WMRP). It is based on Dijkstra's shortest path algorithm [77]. Before introducing heuristics, we give definitions of a single pair graph G from $G_{\mathcal{A}}$ and the cost bound table C on the graph G which are used in all the heuristics. Single pair graph is a graph which has one source node and one sink node. Since the original graph $G_{\mathcal{A}}$ has a set of cost reset nodes R, in order to compute the cumulative weight on a path, we have to reset the weight when the path visits one of the reset nodes. Hence, in the original graph $G_{\mathcal{A}}$, it is necessary to compute the weight of paths from a node in R to another node in R. We define the reduced graph G from $G_{\mathcal{A}}$ and its cost bound table C on G to resolve this issue.

Given a graph $G_{\mathcal{A}} = (V, E, v_s, V_f, W, L, R)$, for a tuple $(v_s, v_r) \in \{v_s\} \times R$, let $V' = (V - R) \cup \{v_s, v_r\}$, $E' = \{(u, v) \in E | u, v \in V'\}$, and $G = (V', E', v_s, v_r, W)$. A path $\eta_{u,v} = (v_0 v_1 \dots v_n)$ on G is a sequence of nodes that start from a node $v_0 = u \in V'$, follow the edges in E', i.e., for $0 \leq i < n$, $(v_i, v_{i+1}) \in E'$, and end to a node $v_n = v \in V'$. A cumulative weight of the path $\eta_{u,v}$ is $w(\eta_{u,v}) = \sum_{i=0}^{n-1} W(v_i, v_{i+1})$, where $v_0 = u$, and $v_n = v$. $P_{u,v}$ is a set of all paths that start from $u \in V'$, follow the edges in E', and end to $v \in V'$. From a node $u \in V'$ to a node $v \in V'$, an optimal path is $\eta^*_{u,v}$ such that $w(\eta^*_{u,v}) \leq w(\eta'_{u,v})$ for any $\eta'_{u,v} \in P_{u,v} - \{\eta^*_{u,v}\}$. (In the case when $u = v \in V'$, $\eta^*_{u,v}$ is 0.) Now, it is ready to define the cost bound table C.

Definition 15. *Given a graph $G = (V', E', v_s, v_r, W)$, a cost bound k, a matrix η^* of optimal paths for $V' \times \{v_r\}$, a cost bound table C on G is as following. For any*

$v \in V'$,

$$C(v) := \begin{cases} 0 & \textit{if } v = v_s \neq v_r \wedge w(\eta^*_{v_s,v_r}) \leq k, \\ k - w(\eta^*_{v,v_r}) & \textit{otherwise.} \end{cases}$$

The cost bound table C has an interesting property when we check if a path is reachable under the cost bound k.

Proposition 1. *Given a graph $G = (V', E', u, v, W)$, a cost bound k, and a cost bound table C on G, there exists a path $\eta_{u,v} = v_0 v_1 \dots v_n$ where $v_0 = u$ and $v_n = v$ which has a cumulative weight under the cost bound k if and only if for every vertex h on $\eta_{u,v}$, the entry of the cost bound table C has nonnegative value.*

Proof. ($\rightarrow$) Suppose that there is a vertex u' on the path $\eta_{u,v}$ and $C(u') < 0$. Then, $\eta^*_{u',v} > k$ since $k - \eta^*_{u',v} < 0$. Hence, $\eta_{u,v} > k$ which is contradiction. Therefore, for every vertex h on $\eta_{u,v}$, the entry of the cost bound table C has nonnegative value.

($\leftarrow$) Suppose that there is no $\eta_{u,v}$ under the cost bound k. However, since $C(u) \geq 0$, $\eta^*_{u,v} \leq k$. Hence, $\eta^*_{u,v} = \eta_{u,v}$. Therefore, there exists a path $\eta_{u,v}$ under the cost bound k. $\square$

Heuristic1 for WMRP

Here, we present the first heuristic. The pseudocode of Heuristic1 is presented in Algorithms 6 to 9.

Given a graph $G_{\mathcal{A}}$ and a cost bound k as inputs, the main algorithm (Alg. 6) finds reachable paths for $\{v_s\} \times R$ and $R \times R$, and outputs a set of atomic propositions $\overline{\Pi}' \subseteq \overline{\Pi}$ from the paths, which has less number of elements than the other subsets. In step 1, it initializes distance tables D_s, D_r, D_{rr}, D_{sr}. All entries of the tables, except for each entry for source vertices, set to ∞, and entries for source vertices set to 0. Step

Algorithm 6: HEURISTIC1($G_{\mathcal{A}}$, k)

```
   Input: a graph $G_{\mathcal{A}} = (V, E, v_s, V_f, R, W, \Lambda, \overline{\Pi})$, and a cost bound k.
   Output: a set of atomic propositions $\overline{\Pi}'$
1  Initialize all entries of $D_s, D_r, D_{rr}, D_{sr}$
2  Compute paths from $v_s$ to each $v_r \in R$ :
       $\langle D_s, \mathcal{M}_s, R_s, F_s \rangle \leftarrow$ FINDPATH($G_{\mathcal{A}}, D_s, \{v_s\}, R, k$)
3  Compute paths between vertices in R:
       $\langle D_r, \mathcal{M}_r, R_r, F_r \rangle \leftarrow$ FINDPATH($G_{\mathcal{A}}, D_r, R, R, k$).
4  Define $G_R$ with $R$ and adjacency matrix $D_r$,$\mathcal{M}_r$,$R_r$,$F_r$.
5  Compute paths between vertices in R:
       $\langle D_{rr}, \mathcal{M}_{rr}, R_{rr}, F_{rr} \rangle \leftarrow$ FINDAPPATH($G_R, D_{rr}, R, R$).
6  Define $G_S$ with $R \cup \{v_s\}$ and adjacency list $D_s$, $\mathcal{M}_s$, $R_s$, $F_s$ and adjacency
     matrix $D_{rr}$, $\mathcal{M}_{rr}$, $R_{rr}$, $F_{rr}$.
7  Compute paths from $v_s$ to each $v_r \in R$ :
       $\langle D_{sr}, \mathcal{M}_{sr}, \_, \_ \rangle \leftarrow$ FINDAPPATH($G_S, D_{sr}, \{v_s\}, R$).
8  For each tuple $(r_1, r_2) \in R \times R$,
       $F_l \leftarrow F_{rr}[r_1, r_2] \cup F_{rr}[r_2, r_1]$,
       $R_l \leftarrow R_{rr}[r_1, r_2] \cup R_{rr}[r_2, r_1]$,
       $\mathcal{M}_l \leftarrow \mathcal{M}_{rr}[r_1, r_2][1] \cup \mathcal{M}_{rr}[r_2, r_1][1]$,
       $\mathcal{M}_{l \cup p} \leftarrow$ GETMINAP($\mathcal{M}_l, R_l, D_{sr}, \mathcal{M}_{sr}$),
       $D_{l \cup p} \leftarrow$ GETMAXDISTANCE($r_1, r_2, D_{rr}, D_{sr}$),
       $L(r_1, r_2) := \begin{cases} \mathcal{M}_{l \cup p} & \text{if } D_{l \cup p} < \infty \wedge |F_l| > 0, \\ \overline{\Pi} & \text{otherwise.} \end{cases}$
9  Find the tuple $(r_1^*, r_2^*)$ that minimizes $|L(r_1, r_2)|$, and set $\overline{\Pi}' = L(r_1^*, r_2^*)$.
10 return $\overline{\Pi}'$
```

2 finds minimum paths for $\{v_s\} \times R$, and gets a distance list D_s, an atomic propoistion list $\mathcal{M}_s$, two subsets R_s, F_s of R, V_f, respectively. Step 3 finds minimum paths for $R \times R$, and gets $D_r, \mathcal{M}_r, R_r, F_r$. In step 2 and 3, FINDPATH (Alg. 7) uses single pair shortest path algorithm, redefining the graph G_s and G_w with $V' = (V - R) \cup \{v_A, v_B\}$ and E' and $E'_{\leftarrow}$ for each pair of vertices for a source v_A and a sink v_B. This prevents that each path from v_A to v_B has any vertex in $R - \{v_A, v_B\}$. Hence, in step 4 of Alg. 6, in order to traverse all paths between vertices in R, it defines graph G_R with R and adjacency matrix tables $D_r, \mathcal{M}_r, R_r, F_r$, and computes shortest paths of the graph in step 5. Likewise, in step 6, it defines a graph G_S with $R \cup \{v_s\}$, adjacency matrix tables $D_s, \mathcal{M}_s, R_s, F_s$ for $\{v_s\} \times R$ and adjacency matrix tables $D_r, \mathcal{M}_r, R_r, F_r$

for $R \times R$, and computes shortest paths of the graph in step 7. In step 8, for paths $r_1 \rightsquigarrow r_2$ and $r_2 \rightsquigarrow r_1$ of each tuple $(r_1, r_2) \in R \times R$, it gets $F_l \subseteq V_f$, $R_l \subseteq R$, and $\mathcal{M}_l \subseteq \overline{\Pi}$. Then, it gets $\mathcal{M}_{l \cup p}$ from GetMinAP. GetMinAP finds minimum number of atomic propositions in $\mathcal{M}_l \cup \mathcal{M}_{sr}[v_s, r_i][1]$ for each $r_i \in R_l$ such that $D_{sr}[v_s, r_i] < \infty$. $D_{l \cup p}$ has maximum distance among $D_{rr}[r_1, r_2]$, $D_{rr}[r_2, r_1]$, and $D_{sr}[v_s, r_1]$. Then, it assigns $L(r_1, r_2)$ to $\mathcal{M}_{l \cup p}$ if the maximum distance is less than ∞ so that paths exist and either the path $r_1 \rightsquigarrow r_2$ or the path $r_2 \rightsquigarrow r_1$ visits at least one vertex in V_f, and to $\overline{\Pi}$ otherewise. Finally, it finds $L(r_1^*, r_2^*)$ which minimizes $|L(r_1, r_2)|$, and returns it.

We remark that we omit the pseudocode of FindAPPath because it is simular with Alg. 7 and Alg. 8. In order to relax the atomic proposition matrix $\mathcal{M}$, the Relax for FindAPPath checks the following condition: given a source vertex $v_A \in V_A$, an edge (u, v) and two atomic proposition matrix $\mathcal{M}$ and $\mathcal{M}'$,

$$\mathcal{M}[v_A, v][2] > |\mathcal{M}[v_A, u][1] \cup \mathcal{M}'[u, v][1]| \tag{2.2}$$

We note that each entry of $\mathcal{M}[v, v']$ where $v \neq v'$ consists of $\left\langle \overline{\Pi}', |\overline{\Pi}'| \right\rangle$ such that $\overline{\Pi}' \subseteq \overline{\Pi}$ and $|\overline{\Pi}'|$ is the number of elements in $\overline{\Pi}'$. Hence, $\mathcal{M}[v, v'][1]$ indicates $\overline{\Pi}'$ and $\mathcal{M}[v, v'][2]$ indicates $|\overline{\Pi}'|$. The Eq. (2.2) is to check if the currently computed number of atomic propositions from v_A to v in $\mathcal{M}$ is bigger then the number of atomic propositions computed from v_A to u in $\mathcal{M}$ and from u to v in $\mathcal{M}'$. If this condition is hold, we relax the current $\mathcal{M}[v_A, v][:]$ to the atomic propositions computed from v_A to u in $\mathcal{M}$ and from u to v in $\mathcal{M}'$. Likewise, if the condition is hold, we also relax the current $D[v_A, v], R[v_A, v], F[v_A, v]$ to $D[v_A, u] + D'[u, v], R[v_A, u] \cup R[u, v], F[v_A, u] \cup F'[u, v]$, respectively.

We also remark that from FindPath, each entry of D_s and D_r has the range from 0 to k if there exists a reachable path under the cost bound k, and have ∞ otherwise. From FindAPPath, each entry of D_{rr} and D_{sr} has the range from 0 to ∞ so that

```
Algorithm 7: FindPath($G_{\mathcal{A}}, D, A, B, k$)
   Input: a graph $G_{\mathcal{A}} = (V, E, v_s, V_f, R, W, \Lambda, \overline{\Pi})$, and a table $D$, two sets of
          vertices A, B, and a cost bound k.
   Output: four tables $D_{AB}, \mathcal{M}_{AB}, R_{AB}, F_{AB}$
 1 $\mathcal{M}'[:,:] \leftarrow \langle \overline{\Pi}, \infty \rangle$
 2 $R'[:,:] \leftarrow \emptyset$; $F'[:,:] \leftarrow \emptyset$
 3 for $(v_A, v_B) \in A \times B$ do
 4     $V' \leftarrow (V - R) \cup \{v_A, v_B\}$
 5     $E' \leftarrow \{(u, v) \in E \mid u, v \in V'\}$
 6     $E'_{\leftarrow} \leftarrow \{(v, u) \mid (u, v) \in E'\}$
 7     $W'_{\leftarrow} \leftarrow$ AssignWeight($E'_{\leftarrow}, E', W$)
 8     $G_w \leftarrow \langle V', E'_{\leftarrow}, W'_{\leftarrow} \rangle$
 9     $G_s \leftarrow \langle V', E', v_A, v_B, R, W, \Lambda, \overline{\Pi} \rangle$
10     $D[:] \leftarrow \infty$; $D[v_B] \leftarrow 0$
11     $\langle D, P \rangle \leftarrow$ ShortestPath($G_w, D, \{v_B\}, \{v_A\}$)
12     For each $v \in V'$,
           $C[v] := \begin{cases} 0 & \text{if } v = v_A \neq v_B \wedge D[v] \leq k, \\ k - D[v] & \text{otherwise.} \end{cases}$
13     $\langle \mathcal{M}, D', R', F' \rangle \leftarrow$ FindMinPath($G_s, C, V_f$)
14     $D_{AB}[v_A, v_B] \leftarrow D'$
15     $\mathcal{M}_{AB}[v_A, v_B] \leftarrow \mathcal{M}$
16     $R_{AB}[v_A, v_B] \leftarrow R'$
17     $F_{AB}[v_A, v_B] \leftarrow F'$
18 return $\langle D_{AB}, \mathcal{M}_{AB}, R_{AB}, F_{AB} \rangle$
```

only if the entry has the value less than ∞, there is a reachable path.

Algorithm 7 has a graph $G_{\mathcal{A}}$, a distance table D, two sets of vertices A, B and the cost bound k as inputs. It finds single pair shortest paths for each tuple $(v_A, v_B) \in A \times B$. It first finds the shortest distance from v_B to v_A, and finds the minimum number of atomic propositions from v_A to v_B. After finding every pair of shortest paths, it outputs a tuple of D_{AB}, $\mathcal{M}_{AB}$, R_{AB}, F_{AB}.

Proposition 2. Shortest-Path$(G, D, \{v_B\}, \{v_A\})$ *returns a shortest distance table D and the predecessor table P on the paths if there exist reachable paths.*

Proof. It is already proved in [77] through optimal substructure property. In short, optimal substructure property is a property that a shortest path between two vertices

```
Algorithm 8: FindMinPath($G_s, C, F$)
   Input: a graph $G_s = (V, E, v_s, v_f, R, W, \Lambda, \overline{\Pi})$, a table $C$ and a set of vertices
          F
   Output: a tuple $\mathcal{M}$, a distance $D' \in \mathbb{R}_{\geq 0}$ and two sets of vertices $R'$, $F'$
 1 $\mathcal{M}[:, :] \leftarrow \langle\overline{\Pi}, \infty, \infty\rangle$; $\mathcal{M}[v_s, :] \leftarrow \langle\emptyset, \infty, 0\rangle$
 2 $\mathcal{Q} \leftarrow V - \{v_s\}$; $P[:] \leftarrow \emptyset$; $R' \leftarrow \emptyset$; $F' \leftarrow \emptyset$; $\mathcal{V} \leftarrow \emptyset$
 3 for $v \in V$ such that $(v_s, v) \in E$ do
 4     $\langle\mathcal{M}, P\rangle \leftarrow$ RelaxAP($(v_s, v), \mathcal{M}, C, P, W, \Lambda$)
 5 $\mathcal{V} \leftarrow \mathcal{V} \cup \{v_s\}$
 6 while $\mathcal{Q} \neq \emptyset$ do
 7     $u \leftarrow$ ExtractMin($\mathcal{Q}$)
 8     if $\mathcal{M}[u, 2] < \infty$ then
 9         $\mathcal{V} \leftarrow \mathcal{V} \cup \{u\}$
10         for $v \in V$ such that $(u, v) \in E$ do
11             $\langle\mathcal{M}, P\rangle \leftarrow$ RelaxAP($(u, v), \mathcal{M}, C, P, W, \Lambda$)
12 if $v_s = v_f \wedge \mathcal{M}[v_f, 2] = \infty$ then
13     $\mathcal{M}' \leftarrow \langle\overline{\Pi}, \infty\rangle$; $D' \leftarrow \infty$
14 else
15     $\mathcal{M}' \leftarrow \langle\mathcal{M}[v_f, 1], \mathcal{M}[v_f, 2]\rangle$; $D' \leftarrow \mathcal{M}[v_f, 3]$
16 $R' \leftarrow$ ExtractSetR($P, R, v_f, v_s$)
17 $F' \leftarrow$ ExtractSetF($P, F, v_f, v_s$)
18 return $\langle\mathcal{M}', D', R', F'\rangle$
```

contains other shortest paths within it. □

Proposition 3. *Given each pair $(v_A, v_B) \in A \times B$, two graphs G_w and G_s, two distance tables D and D', for each $v \in V'$, $D'[v]$ has an individual cost bound k_v where $k_v \leq k$ for the path from v_A to v.*

Proof. We know from Proposition 2, for each $v \in V'$, $D[v]$ has the shortest distance on the path from v_B to v. Suppose there exists a reachable path $v_B \rightsquigarrow v \rightsquigarrow v_A$ under the cost bound k. Due to Subpath-Optimal property, $w(p_{v_B \rightsquigarrow v}) \leq w(p_{v_B \rightsquigarrow v \rightsquigarrow v_A}) \leq k$. Let $w(p_{v_B \rightsquigarrow v})$ be $k_{v_B \rightsquigarrow v}$. Since $w(p_{v_B \rightsquigarrow v \rightsquigarrow v_A}) = w(p_{v_B \rightsquigarrow v}) + w(p_{v \rightsquigarrow v_A}) \leq k$, $k_{v_B \rightsquigarrow v} + w(p_{v \rightsquigarrow v_A}) \leq k$. Then, the entry of $D[v]$ has $k_{v_B \rightsquigarrow v}$. □

Proposition 4. *Given a graph G_s = (V',E',v_A,v_B,R,W,Λ,$\overline{\Pi}$), a cost bound table*

Algorithm 9: RelaxAP$((u,v),\mathcal{M},C,P,W,\Lambda)$

```
Input: an edge $(u,v)$, three tables $\mathcal{M},C,P$, and two functions $W,\Lambda$
Output: a table $\mathcal{M}$ and $P$
1 if $|\mathcal{M}[u,1]\cup\Lambda((u,v))| < \mathcal{M}[v,2]$ and $C[v]-(\mathcal{M}[u,3]+W((u,v)))\geq 0$ then
2     $\mathcal{M}[v,1]\leftarrow\mathcal{M}[u,1]\cup\Lambda((u,v))$
3     $\mathcal{M}[v,2]\leftarrow|\mathcal{M}[u,1]\cup\Lambda((u,v))|$
4     $\mathcal{M}[v,3]\leftarrow\mathcal{M}[u,3]+W((u,v))$
5     $P[v]\leftarrow u$
6 return $\langle\mathcal{M},P\rangle$
```

D', and a distance table D from Shortest-Path*, any reachable path for v_A to each $v\in V'$ under the cost bound k has an individual cost bound.*

Proof. We know from Proposition 2, for each $v\in V'$, $D[v]$ has the shortest distance on the path from v_B to v. Suppose that there is a reachable path $v_0,v_1,v_2,\ldots,v_{i-1},v_i$ where $v\in\{v_0,\ldots,v_i\}$, $v_0=v_B$, and $v_i=v_A$ under the cost bound k. Then, $0=D[v_0]\leq D[v_1]\leq\cdots\leq D[v_{i-1}]\leq D[v_i]\leq k$. If $v_A=v_B$, then $D[v_A]=0$ since $D[v_B]=0$. If $v_A\neq v_B$, then $0\leq D[v_A]\leq k$. In step 12 of Alg. 7, $D'[v_A]$ is assigned to k if $v_A=v_B$ since $k-D[v_A]=k$. If $v_A\neq v_B$ and $D[v_A]\leq k$, then $D'[v_A]$ is assigned to 0. Hence, $0=D'[v_i]\leq D'[v_{i-1}]\leq\cdots\leq D'[v_1]\leq D'[v_0]=k$. □

Proposition 5. *Given a graph $G_s=(V',E',v_A,v_B,R,W,\Lambda,\overline{\Pi})$, a cost bound table D', and a distance table D from* Shortest-Path*, for any path for v_A to each $v\in V'-\{v_A\}$ over the cost bound k, the entry of $D'[v]$ has the value less than 0.*

Proof. Suppose that there is a path $v_0,v_1,\ldots,v_j,\ldots,v_{i-1},v_i$ where $v\in\{v_{j+1},\ldots,v_{i-1}\}$, $v_0=v_B$, and $v_i=v_A$ over the cost bound k. Then, $0\leq D[v_0]\leq\cdots\leq D[v_j]\leq k< D[v_{j+1}]\leq\cdots\leq D[v_i]$. If $v_A=v_B$, then $D'[v_A]=k$. In step 12 of Alg. 7, $D'[v_A]$ is assigned to k. However, $D'[v]>k$, so in step 12, $D'[v]$ is assigned to $k-D'[v]<0$. Hence, $D'[v_{i-1}]\leq\cdots\leq D'[v_{j+1}]<0$. If $v_A\neq v_B$, then $D[v_B]=0$, so $D'[v_B]=0$ and, $D[v_A]\geq D[v]>k$, so $D'[v_A]\leq D'[v]<0$. Hence, $D'[v_A]\leq\cdots\leq D'[v_{j+1}]<0$, and $0=D'[v_0]\leq D'[v_1]\leq\cdots\leq D'[v_j]\leq k$. □

Remark 1. *We remark that if $v_A = v$ and there exists a path from v_A to v over the cost bound k, it would be a self edge or a cycle. In this case, $D'[v]$ does not have the value less than 0 (the value is k). However, since the path is over the bound k, $W((v_A, v)) > k$ or $w(p_{v_A \rightsquigarrow v}) > k$. Both are over the individual cost bound $D'[v]$. Therefore, it is not reachable with the cost bound table D'.*

Algorithm 8 has a graph $G_s = (V, E, v_s, v_f, R, W, \Lambda, \overline{\Pi})$, a cost bound table C, and a set of vertices F as inputs. It is similar with Dijkstra's shortest path algorithm and AAMRP in Sec. 2.3.2, but instead of finding shortest path, it finds minimum number of atomic propositions on the path from v_s to v_f while checking the cost bound table C. Then, it outputs the distance from v_s to v_f, a set of atomic propositions, a set of vertices $R' \subseteq R$ visited on the path, and a set of vertices $F' \subseteq F$.

Algorithm 9 has an edge (u, v), three tables $\mathcal{M}, C, P$, and two functions W, Λ as inputs. It compares the number of atomic propositions on the path over the edge (u, v) with the number of atomic propositions on the previous path to v. If the new path has less number of atomic propositions and the cumulative weight is less than the cost bound $C[v]$, it relaxes the entries of two tables $\mathcal{M}$ and P, and outputs those tables.

Proposition 6. *For each pair $(v_A, v_B) \in A \times B$,* FindMinPath *searches all reachable paths from v_A to v_B under the cost bound k.*

Proof. From Proposition 4 and 5, the cost bound table D' has individual cost bound. RelaxAP does not check if the edge (u, v) has shorter distance than before, but check if the cumulative weight with the edge (u, v) is over than the inidividual cost bound. Therefore, all reachable paths under the cost bound are considered to be relaxed. □

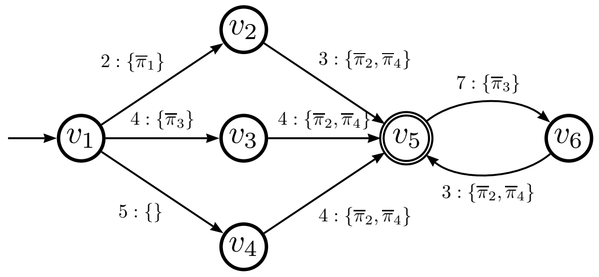

v_1
v_2
v_3
v_4
v_5
v_6
$2 : \{\overline{\pi}_1\}$
$4 : \{\overline{\pi}_3\}$
$5 : \{\}$
$3 : \{\overline{\pi}_2, \overline{\pi}_4\}$
$4 : \{\overline{\pi}_2, \overline{\pi}_4\}$
$4 : \{\overline{\pi}_2, \overline{\pi}_4\}$
$7 : \{\overline{\pi}_3\}$
$3 : \{\overline{\pi}_2, \overline{\pi}_4\}$

Figure 2.8: The graph of Example 9. The source $v_s = v_1$ is denoted by an arrow, the sink v_5 by double circle ($V_f = \{v_5\}$), and the cost reset vertex is v_6 ($R = \{v_6\}$).

Example 9. *Let us consider the graph in Fig. 2.8. The source vertex of this graph is $v_s = v_1$, the set of sink vertices is $V_f = \{v_5\}$, and the set of cost reset vertices is $R = \{v_6\}$. The $\overline{\Pi}$ set is $\{\overline{\pi}_1, \overline{\pi}_2, \overline{\pi}_3, \overline{\pi}_4\}$, and the cost bound is $k = 15$. Consider the first iteration of the loop at line 11 of Alg. 7.*

- *Before the execution of* ShortestPath*, the distance table is initialized: $D[:] = \infty$ and $D[v_6] = 0$.*
- *After the* ShortestPath*, the distance table has the following entries: $D[v_1] = 12, D[v_2] = 10, D[v_3] = 11, D[v_4] = 11, D[v_5] = 7, D[v_6] = 0$.*
- *After the step 12, the cost bound table Chas the following entries: $C[v_1] = 0, C[v_2] = 5, C[v_3] = 4, C[v_4] = 4, C[v_5] = 8, C[v_6] = 15$.*
- *After the* FindMinPath *at line 13, the set of atomic propositions returned would be $\mathcal{M}' = \langle\{\overline{\pi}_1, \overline{\pi}_2, \overline{\pi}_3, \overline{\pi}_4\}, 4\rangle$, which corresponds to the path v_1, v_2, v_5, v_6. This is because at the vertex v_5 both path from v_2 and from v_3 have same number of atomic propositions: $\{\overline{\pi}_1, \overline{\pi}_2, \overline{\pi}_4\}$, $\{\overline{\pi}_2, \overline{\pi}_3, \overline{\pi}_4\}$, respectively.*

Note that this heuristic returns a set of atomic propositions $\overline{\Pi}$ which is not optimal. The path v_1, v_3, v_5, v_6 would return $\{\overline{\pi}_2, \overline{\pi}_3, \overline{\pi}_4\}$. In addition, the path v_1, v_4, v_5, v_6

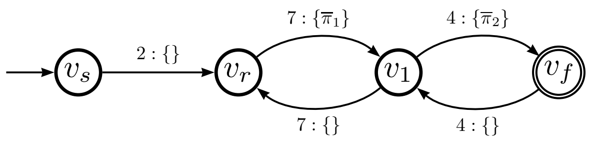


Figure 2.9: The graph of Example 10. The source v_s is denoted by an arrow, the sink v_f by double circle ($V_f = \{v_f\}$), and the cost reset vertex is v_r ($R = \{v_r\}$).

cannot be chosen because the cumulative weight of the path is 16 which is greater than the cost bound.

Example 10. *Let us consider the graph in Fig. 2.9. The source vertex of this graph is v_s, the set of sink is $V_f = \{v_f\}$, and the set of cost reset vertices is $R = \{v_r\}$. The $\overline{\Pi}$ set is $\{\overline{\pi}_1, \overline{\pi}_2\}$, and the cost bound is $k = 25$. Consider the first iteration of the loop at line 13 of Alg. 7.*

- *Before the first execution of the for loop, the table F_{AB} is initialized: $F_{AB}[:] = \emptyset$.*
- *After the* FindMinPath *at line 13, the visited set of final vertices is empty: $F' = \emptyset$. Hence, each entry of F_{AB} is still $\emptyset$.*

Note that in this example, the heuristic cannot return proper sets $F' = \{v_f\}$. This is because FindPath *in the step 3 of Alg. 6 tries to find the path having minimum number of atomic propositions between two vertices in R. Hence, after starting from v_r, when it is in v_1, it chooses the edge (v_1, v_r), not choosing the edge (v_1, v_f).*

We shall see how Heuristic 2 has resolved the issue in Example 10.

Correctness: We already show that Heuristic1 does not return optimal solution from Example 10. This is because Heuristic1 does not find every path for $R \times V_f$ and $V_F \times R$. However, we showed that Heuristic1 checks all reachable paths under the cost bound k for $R \times R$, and finds minimum number of atomic propositions on the path $r_1 \rightsquigarrow r_2 \rightsquigarrow r_1$ for each tuple $(r_1, r_2) \in R \times R$.

Running time: The running time analysis of Alg. 8 is similar to that of Dijkstra's shortest path algorithm and AAMRP in Sec. 2.3.2. We assume that we are using a data structure for sets that supports $O(1)$ set cardinality quarries, $O(\log n)$ membership quarries and element insertions (see [77]) and $O(n)$ set up time. Under the assumption that $\mathcal{Q}$ is implemented in such a data structure, each ExtractMin takes $O(\log V)$ time. Furthermore, we have $O(V)$ such operations (actually $|V| - 1$) for a total of $O(V \log V)$.

Setting up the data structure for $\mathcal{Q}$ will take $O(V)$ time. Furthermore, in the worst case, we have a set $\Lambda(e)$ for each edge $e \in E$ with set-up time $O(E\overline{\Pi})$. Note that the initialization of $\mathcal{M}[v, :]$ to $\langle \overline{\Pi}, \infty, \infty \rangle$ does not have to be implemented since we can have indicator variables indicating when a set is supposed to contain all the (known in advance) elements.

Assuming that E is stored in an adjacency list, the total number of calls to RelaxAP at lines 4 and 11 of Alg. 8 will be $O(E)$ times. Each call to RelaxAP will have to perform a union of two sets ($\mathcal{M}[u, 1]$ and $\Lambda(u, v)$). Assuming that both sets have in the worst case $|\overline{\Pi}|$ elements, each union will take $O(\overline{\Pi} \log \overline{\Pi})$ time. Each set size quarry takes $O(1)$ time and updating the keys in $\mathcal{Q}$ takes $O(\log V)$ time. ExtractSetR checks the path from v_f to v_s through the predecessor table P and extracts each vertex on the path if the vertex is in the set R. Hence, it takes $O(V \log R)$. Likewise, ExtractSetF takes $O(V \log V_f)$. Therefore, the running time of FindMinPath is $O(V + E\overline{\Pi} + V \log V + E(\overline{\Pi} \log \overline{\Pi} + \log V) + V \log R + V \log V_f)$.

Note that under Assumption 1 all nodes of $\mathcal{T}$ are reachable ($|V| < |E|$), the same property does not hold for the product automaton. (e.g, think of an environment $\mathcal{T}$ and a specification automaton whose graphs are Directed Acyclic Graphs (DAG). However, even in this case, we have ($|V| < |E|$). The running time of FindMinPath is $O(E(\overline{\Pi} \log \overline{\Pi} + \log V))$. Therefore, we observe that the running time also depends on the size of the set $\overline{\Pi}$. However, such a bound is very pessimistic since not all the edges will be disabled on $\mathcal{A}$ and, moreover, most edges will not have the whole set $\overline{\Pi}$ as candidates for removal.

Running time of Alg. 7 is as following. Step 1 takes $O(AB)$ times, and step 2 takes $O(AB)$ times. In each iteration of the for loop, making the set V' requires one subtraction $(V-R)$ and one union operation $(V-R)\cup\{v_A, v_B\}$. However, we can move the subtraction before the for loop so that each iteration can do one union operation. The subtraction takes $O(R\log V)$, and the union takes $O(\log V')$. Making the set of edges E' takes $O(E\log V')$. Making the set $E'_{\leftarrow}$ takes $O(E')$. Assigning the functon $W'_{\leftarrow}$ takes $O(E')$. Initializing the distance table D takes $O(V')$. ShortestPath takes $O(V'\log V' + E')$. Setting up the cost bound table C takes $O(V')$. FindMinPath takes $O(E'(\overline{\Pi}\log\overline{\Pi} + \log V'))$. Since, $O(E'(\overline{\Pi}\log\overline{\Pi} + \log V'))$ dominates others in the iteration, the running time of total for loop is $O(ABE'(\overline{\Pi}\log\overline{\Pi} + \log V'))$. Since $|E'| \leq |V'|^2 = |V-R|^2$, $O(AB(V-R)^2)$ also dominates $O(AB)$ and $O(R\log V)$. Therefore, the running time of Alg. 7 is $O(AB(V-R)^2(\overline{\Pi}\log\overline{\Pi} + \log(V-R))$.

Running time of Alg. 6 is as following. Initializing distance table D_{rr} takes $O(R^2)$. Computing paths from v_s to each $v_r \in R$ takes $O(R(V-R)^2(\overline{\Pi}\log\overline{\Pi} + \log(V-R))$. Computing paths between vertices in R takes $O(R^2(V-R)^2(\overline{\Pi}\log\overline{\Pi} + \log(V-R))$. FindAPPath for paths between vertices in R takes $O(R^4(\overline{\Pi}\log\overline{\Pi} + \log R))$. FindAPPath for paths from v_s to each vertex in R takes $O(R^3(\overline{\Pi}\log\overline{\Pi} + \log R))$. Each iteration of the for loop needs $O(V_f\log V_f)$ for union of two sets of F_{rr}, $O(R\log R)$ for union of two sets of R_{rr}, $O(\overline{\Pi}\log\overline{\Pi})$ for union of two sets of $\mathcal{M}_{rr}$, $O(R(\overline{\Pi}\log\overline{\Pi}))$ for GetMinAP, and $O(1)$ for GetMaxDistance. Since $O(R(\overline{\Pi}\log\overline{\Pi}))$ dominates others in the iteration, the total for loop takes $O(R^3(\overline{\Pi}\log\overline{\Pi}))$. To find the tuple (r_1^*, r_2^*) takes $O(R^2)$. However, $O(R^2(V-R)^2(\overline{\Pi}\log\overline{\Pi}+\log(V-R))$ dominates $O(R^4(\overline{\Pi}\log\overline{\Pi}+\log R)$. Therefore, the running time of Alg. 6 is $O(R^2(V-R)^2(\overline{\Pi}\log\overline{\Pi} + \log(V-R))$.

Algorithm 10: HEURISTIC2($G_{\mathcal{A}}, k$)

```
   Input: a graph $G_{\mathcal{A}} = (V, E, v_s, V_f, R, W, \Lambda, \overline{\Pi})$, and a cost bound k.
   Output: a set of atomic propositions $\overline{\Pi}'$
1  Same codes from step 1 to step 7 of HEURISTIC1
2  Initialize all entries of $D_{fr}, D_{rfr}$.
3  Compute paths from each $v_f \in V_f$ to each $v_r \in R$ :
       $\langle D_{fr}, \mathcal{M}_{fr}, R_{fr}, F_{fr}\rangle \leftarrow$ FINDPATH$(G_{\mathcal{A}}, D_{fr}, V_f, R, k)$.
4  Compute paths from each $v_r \in R$ to each $v_f \in V_f$ :
       $\langle D_{rfr}, \mathcal{M}_{rfr}, R_{rfr}, F_{rfr}\rangle \leftarrow$ FIND-RF-PATH$(G_{\mathcal{A}}, D_{rfr}, R, V_f, k, D_{fr})$
5  For each triple $(r_1, f, r_2) \in R \times V_f \times R$,
       $D_l \leftarrow$ MAXDISTFROMRFR$(r_1, f, r_2, D_{rfr}, D_{fr}, D_{rr})$,
       $R_l \leftarrow$ GETRFROMRFR$(r_1, f, r_2, R_{rfr}, R_{fr}, R_{rr})$,
       $\mathcal{M}_l \leftarrow$ GETAPFROMRFR$(r_1, f, r_2, \mathcal{M}_{rfr}, \mathcal{M}_{fr}, \mathcal{M}_{rr})$,
       $\mathcal{M}_{l\cup p} \leftarrow$ GETMINAP$(\mathcal{M}_l, R_l, D_{sr}, \mathcal{M}_{sr})$,
       $D_{l\cup p} \leftarrow max(D_l, D_{sr}[v_s, r_1])$,
       $L(r_1, f, r_2) := \begin{cases} \mathcal{M}_{l\cup p} & \text{if } D_{l\cup p} < \infty, \\ \overline{\Pi} & \text{otherwise.} \end{cases}$
6  Find the tuple $(r_1^*, f^*, r_2^*)$ that minimizes $|L(r_1, f, r_2)|$, and a set
   $\overline{\Pi}' = L(r_1^*, f^*, r_2^*)$.
7  return $\overline{\Pi}'$
```

Heuristic2 for WMRP

The second heuristic has more steps after the first heuristic. The pseudocode of Heuristic2 is presented in Alg. 10 and 11. The algorithm has seven steps. Step 1 is same as the HEURISTIC1. Step 2 initializes all entries of D_{fr}, D_{rfr}. In step 3, it computes path for $V_f \times R$. In step 4, it computes path for $R \times V_f$ with the distance table D_{fr} computed from step 3. In step 5, for paths $r_1 \rightsquigarrow f \rightsquigarrow r_2 \rightsquigarrow r_1$ of each triple $(r_1, f, r_2) \in R \times V_f \times R$, it gets the distance D_l, a set of cost reset vertices $R_l \subseteq R$, and atomic propositions $\mathcal{M}_l \subseteq \overline{\Pi}$. Then, it gets $\mathcal{M}_{l\cup p}$ from GETMINAP, and $D_{l\cup p}$. Then, it assigns $L(r_1, f, r_2)$ to $\mathcal{M}_{l\cup p}$ if the maximum distance is less than ∞ so that there exist the paths $r_1 \rightsquigarrow f \rightsquigarrow r_2 \rightsquigarrow r_1$ and $v_s \rightsquigarrow r_1$, and to $\overline{\Pi}$ otherewise. Finally, it finds the minimum number of atomic propositions, and returns them.

Algorithm 11 is similar with Alg. 7, except for the step 11 to the step 12. Instead

Algorithm 11: FIND-RF-PATH($G_{\mathcal{A}}, D_{rfr}, R, V_f, k, D_{fr}$)

```
Input: a graph $G_{\mathcal{A}} = (V, E, v_s, V_f, R, W, \Lambda, \overline{\Pi})$, two sets of vertices $R$, $V_f$, a
       cost bound $k$, and two tables $D_{rfr}, D_{fr}$
Output: two tables $D', \mathcal{M}'$ and two sets of vertices $R', F'$
1  $\mathcal{M}'[:,:,:] \leftarrow \langle \overline{\Pi}, \infty \rangle$; $D'[:,:,:] \leftarrow \infty$
2  $R'[:,:,:] \leftarrow \emptyset$; $F'[:,:,:] \leftarrow \emptyset$
3  for $(r_i, f, r_j) \in R \times V_f \times R$ do
4      $V' \leftarrow (V - R) \cup \{r_i, f\}$
5      $E' \leftarrow \{(u, v) \in E | u, v \in V'\}$
6      $E'_{\leftarrow} \leftarrow \{(v, u) | (u, v) \in E'\}$
7      $W'_{\leftarrow} \leftarrow$ ASSIGNWEIGHT($E'_{\leftarrow}, E', W$)
8      $G_w \leftarrow \langle V', E'_{\leftarrow}, W'_{\leftarrow} \rangle$
9      $G_s \leftarrow \langle V', E', r_i, \{f\}, R, W, \Lambda, \overline{\Pi} \rangle$
10     $D[:] \leftarrow \infty$
11     if $0 \leq D_{fr}[f, r_j] \leq k$ then
12         $D[f] \leftarrow -(D_{fr}[f, r_j] - k)$
13     $\langle D, P \rangle \leftarrow$ SHORTESTPATH($G_w, D, \{f\}, \{r_i\}$)
14     For each $v \in V'$,
           $C[v] := \begin{cases} 0 & \text{if } v = r_i \neq f \wedge D[v] \leq k, \\ k - D[v] & \text{otherwise.} \end{cases}$
15     $\langle \mathcal{M}, D'', R'', F'' \rangle \leftarrow$ FINDMINPATH($G_s, C, V_f$)
16     $\mathcal{M}'[r_i, f, r_j] \leftarrow \langle \mathcal{M}[f, 1], \mathcal{M}[f, 2] \rangle$
17     $D'[r_i, f, r_j] \leftarrow D''$
18     $R'[r_i, f, r_j] \leftarrow R''$
19     $F'[r_i, f, r_j] \leftarrow F''$
20 return $\langle D', \mathcal{M}', R', F' \rangle$
```

of initializing $D[f]$ to 0, it uses the distance of D_{fr}. After the for loop, it returns a tuple of $D', \mathcal{M}', R', F'$.

Proposition 7. HEURISTIC2 *checks every path for $R \times V_f$, $V_f \times R$, and $R \times R$ if there exists a reachable path under cost bound k.*

Proof. In step 3 and 5 of Alg. 6, checks every path for $R \times R$, and it is in step 1 of Alg. 10. In step 3 of Alg. 10, it checks every path for $V_f \times R$, and step 4 checks every path for $R \times V_f$. In order to keep the cost bound each path from $r_1 \rightsquigarrow f \rightsquigarrow r_2$, FIND-RF-PATH uses distance table D_{fr}. Hence, it checks all reachable path from $r_1 \rightsquigarrow f \rightsquigarrow r_2$ under the cost bound k. □

Example 11. *We revisit the Example 10. Since the fifth step computes the shortest path for $V_f \times R$, in the graph in Fig. 2.9, the path v_f, v_1, v_r is returned from the step. In sixth step, the path v_r, v_1, v_f is returned.*

Note that hueristic2 resolves the issue of Example 10. From this example, we can see that the lasso part is somewhat complicated. Since v_1 is visited twice, it is not a cycle, which is trail path.

Correctness: Even though Heuristic2 resolves the issue in Example 10, it returns same solution as Heuristic1 does for Example 9. However, we showed that heuristic2 checks all reachable paths under the cost bound k for $R \times V_f$, $V_f \times R$, and $R \times R$, and finds minimum number of atomic propositions on the path $r_1 \rightsquigarrow f \rightsquigarrow r_2 \rightsquigarrow r_1$ for each triple $(r_1, f, r_2) \in R \times V_f \times R$.

Running time: The running time analysis of Alg. 11 is similar to that of Alg. 7. Only difference is that the for loop is for each triple in $R \times V_f \times R$. Hence, running time of Alg. 11 is $O(R^2 V_f (V-R)^2 (\overline{\Pi} \log \overline{\Pi} + \log(V-R)))$.

The running time of Alg. 10 is as following. Since the first step has same codes from Alg. 6, it takes $O(R^2(V-R)^2(\overline{\Pi}\log\overline{\Pi} + \log(V-R))$. Initializing the distance table D_{rfr} takes $O(R^2 V_f)$. Computing paths from each $v_f \in V_f$ to each $v_r \in R$ takes $O(V_f R (V-R)^2 (\overline{\Pi}\log\overline{\Pi} + \log(V-R))$. Find-RF-Path for each $v_r \in R$ to each $v_f \in V_f$ takes $O(R^2 V_f (V-R)^2 (\overline{\Pi}\log\overline{\Pi} + \log(V-R)))$. Each iteration of the for loop needs $O(1)$ for MaxDistFromRFR, $O(R\log R)$ for union of three sets of R_{rfr}, R_{fr}, R_{rr}, $O(\overline{\Pi}\log\overline{\Pi})$ for union of three sets of $\mathcal{M}_{rfr}, \mathcal{M}_{fr}, \mathcal{M}_{rr}$, $O(R\overline{\Pi}\log\overline{\Pi})$ for GetMinAP, $O(1)$ for max operation and for seting up $L(r_1, f, r_2)$. Since $O(R\log R)$ and $O(R\overline{\Pi}\log\overline{\Pi})$ dominates others, the total for loop takes $O(R^3 V_f(\log R + \overline{\Pi}\log\overline{\Pi}))$. Finding the triple (L_1^*, f^*, L_2^*) takes $O(R^2 V_f)$. However, $O(R^2 V_f (V-R)^2 (\overline{\Pi}\log\overline{\Pi} + \log(V-R)))$ dominates $O(R^3 V_f(\log R + \overline{\Pi}\log\overline{\Pi}))$. Therefore, the running time of Alg. 10 is $O(R^2 V_f (V-R)^2 (\overline{\Pi}\log\overline{\Pi} + \log(V-R)))$.

2.4.4 *Result*

In this section, we present experimental result using our propotype implementation of heuristics and brute-force search. The propotype implementation is written in Python. Therefore, we expect the running times to substantially improve with a C implementation using state-of-the-art data structure implementations.

For the experiments, we utilized the ASU super computing center which consists of clusters of Dual 4-core processors, 16 GB Intel(R) Xeon(R) CPU X5355 @2.66 Ghz. Our implementation does not utilize the parallel architecture. The clusters were used to run the many different test cases in parallel on a single core. The operating system is CentOS release 5.9.

In order to assess the experimental approximation ratio of two heuristics, we compared the solutions returned by heuristics with the Brute-force search. The Brute-force search is guaranteed to return a minimal solution to the WMRP problem.

We performed a large number of experimental comparisons on random benchmark instances of various sizes. Each test case consisted of two randomly generated DAGs which represented an environment and a specification. Both graphs have self-loops on their leaves so that a feasible lasso path can be found. The number of atomic propositions in each instance was equal to four times the number of nodes in each acyclic graph. For example, in the benchmark where the graph had 9 nodes, each DAG had 3 nodes, and the number of atomic propositions was 12. The final nodes are chosen randomly and they represent 5%-40% of the nodes. The number of edges in most instances were 2-3 times more than the number of nodes.

Table 2.3 compares the results of the Brute-force search with the results of two heuristics on test cases of different sizes (total number of nodes). For each graph size, we performed 200 tests and we report minimum, average and maximum computation times in sec and minimum, average and maximum ratio for two heuristics comparing with the result of Brute-force search.

As shown in Alg. 4, our Brute-force search starts choosing atomic propositions to be removed from small number, and checks if it is reachable so that there exists a lasso and prefix path. We remark that even though each instance was tested for 8 hours window, the Brute-force search was not able to provide an answer to all the test cases within the time limit. The numbers under the column of succ for Brute-force search show that as the number of nodes increases the number of success decreases.

The comparison for the approximation ratio was possible only for the test cases where Brute-force search successfully completed the computation.

There are few interesting observations. First, the maximum approximation ratio for both Heuristic1 and Heustic2 is experimentally determined to be 2. For the randomly generated graphs that we have constructed the bound appears to be 2. However, it is not easy to construct random examples that produce higher approximation ratios. Such example scenarios must be carefully constructed in advance. Second, the ratio was same between Heuristic1 and Heuristic2 for the 529 nodes' test even though Heuristic2 took almost 12 times longer than Heuristic1 for the average time.

2.5 LTL Revision Problem under Preferences

When choosing an alternative plan, each user can have different preferences. Suppose that users can assign some preference level to each proposition labeling the specification automaton through the preference function θ. When preference level is 0, it is least preferred, and the greater preference level is, the more preferred it is. However, preference level cannot be ∞. We remark that each occurrence of an atomic proposition over different transitions can have different preference levels. Therefore, taking transitions on the cross-product automaton $\mathcal{A}$, we can get as a reward preference

	BRUTER-FORCE SEARCH				HEURISTIC1							HEURISTIC2						
	TIME				TIME			RATIO				TIME			RATIO			
Nodes	min	avg	max	succ	min	avg	max	min	avg	max	succ	min	avg	max	min	avg	max	succ
9	0.079	0.266	1.561	182	0.047	0.292	5.58	1	1	1	182	0.041	0.114	1.529	1	1.005	2	182
100	0.903	12.95	326.7	198	1.095	1.825	4.66	1	1.002	1.5	198	1.846	6.751	18.03	1	1.002	1.5	198
196	3.271	856.5	25935	199	5.367	8.412	23.7	1	1.012	2	199	8.9	40.03	135.3	1	1.001	1.166	199
324	12.33	1574	27170	174	17.27	27.52	87.32	1	1.002	1.5	199	19.92	215.3	844.9	1	1.002	1.5	199
400	16.53	2612	26947	165	25.03	46.31	193.2	1	1.012	2	200	36.15	451.3	1498	1	1.014	2	200
529	33.56	3154	25265	125	60.8	101.9	142.4	1	1.0036	1.25	200	118.4	1231	5074	1	1.0036	1.25	198

Table 2.3: Numerical Experiments: Number of nodes versus the results of Brute-force search and Heuristic1 and Heuristic2. The numbers under the TIME columns indicate computation times in sec. RATIO indicates the experimentally observed approximation ratio to the optimal solution. The numbers under the succ columns indicates the number of success results among 200 instances.

levels of elements in Π on the transitions.

A revised specification is one that can be satisfied on the discrete abstraction of the workspace or the configuration space of the robot. In order to search for a minimal revision, we need first to define an ordering relation on automata as well as a distance function between automata. We do not want to consider the "space" of all possible automata, but rather the "space" of specification automata which are semantically close to the initial specification automaton $\mathcal{B}_\mathbf{s}$. The later will imply that we remain close to the initial intention of the designer. We propose that this space consists of all the automata that can be derived from $\mathcal{B}_\mathbf{s}$ by removing symbols from the transitions. Our definition of the ordering relation between automata relies upon the previous assumption.

Definition 16 (Relaxation). *Let* $\mathcal{B}_1 = (S_{\mathcal{B}_1},\ s_0^{\mathcal{B}_1},\ \mathcal{P}(\Pi),\ \rightarrow_{\mathcal{B}_1},\ F_{\mathcal{B}_1},\ \theta_{\mathcal{B}_1})$ *and* $\mathcal{B}_2 = (S_{\mathcal{B}_2}, s_0^{\mathcal{B}_2}, \mathcal{P}(\Pi), \rightarrow_{\mathcal{B}_2}, F_{\mathcal{B}_2}, \theta_{\mathcal{B}_2})$ *be two specification automata having the same preference levels for* $\mathcal{P}(\Pi)$*. Then, we say that* $\mathcal{B}_2$ *is a relaxation of* $\mathcal{B}_1$ *and we write* $\mathcal{B}_1 \preceq \mathcal{B}_2$ *if* *and only if* $S_{\mathcal{B}_1} = S_{\mathcal{B}_2} = S$, $s_0^{\mathcal{B}_1} = s_0^{\mathcal{B}_2}$, $F_{\mathcal{B}_1} = F_{\mathcal{B}_2}$, $\theta_{\mathcal{B}_1} = \theta_{\mathcal{B}_2}$ *and*

1. $\forall(s, l, s') \in \rightarrow_{\mathcal{B}_1} - \rightarrow_{\mathcal{B}_2} \,.\, \exists l' \,.$

   $(s, l', s') \in \rightarrow_{\mathcal{B}_2} - \rightarrow_{\mathcal{B}_1}$ *and* $l' \subseteq l$.

2. $\forall(s, l, s') \in \rightarrow_{\mathcal{B}_2} - \rightarrow_{\mathcal{B}_1} \,.\, \exists l' \,.$

   $(s, l', s') \in \rightarrow_{\mathcal{B}_1} - \rightarrow_{\mathcal{B}_2}$ *and* $l \subseteq l'$.

We remark that if $\mathcal{B}_1 \preceq \mathcal{B}_2$, then $\mathcal{L}(\mathcal{B}_1) \subseteq \mathcal{L}(\mathcal{B}_2)$ since the relaxed automaton allows more behaviors to occur.

We can now define the set of automata over which we will search for a revision.

Definition 17. *Given a system* $\mathcal{T}$ *and and a specification automaton* $\mathcal{B}_{\mathbf{s}}$*, the set of valid relaxations of* $\mathcal{B}_{\mathbf{s}}$ *is defined as* $\mathfrak{R}(\mathcal{B}_{\mathbf{s}}, \mathcal{T}) = \{\mathcal{B} \mid \mathcal{B}_{\mathbf{s}} \preceq \mathcal{B}$ *and* $\mathcal{L}(\mathcal{T} \times \mathcal{B}) \neq \emptyset\}$.

We can now search for a solution in the set $\mathfrak{R}(\mathcal{B}_{\mathbf{s}}, \mathcal{T})$. Different solutions can be compared from their revision sets.

Definition 18 (Revision Set)**.** *Given a specification automaton* $\mathcal{B}_{\mathbf{s}}$ *and a* $\mathcal{B} \in \mathfrak{R}(\mathcal{B}_{\mathbf{s}}, \mathcal{T})$*, the revision set is defined as* $R(\mathcal{B}_{\mathbf{s}}, \mathcal{B}) = \{(\pi, s, s') \mid \pi \in (\lambda_{\mathcal{B}_{\mathbf{s}}}(s, s') - \lambda_{\mathcal{B}}(s, s'))\}$.

We define two different revision problems.

Problem 4 (Min-Sum Revision)**.** *Given a system* $\mathcal{T}$ *and a specification automaton* $\mathcal{B}_{\mathbf{s}}$*, if the specification* $\mathcal{B}_{\mathbf{s}}$ *is not satisfiable on* $\mathcal{T}$*, then find a revision set* R *such that* $\sum_{\rho \in R} \theta(\rho)$ *is minimized.*

Problem 5 (Min-Max Revision)**.** *Given a system* $\mathcal{T}$ *and a specification automaton* $\mathcal{B}_{\mathbf{s}}$*, if the specification* $\mathcal{B}_{\mathbf{s}}$ *is not satisfiable on* $\mathcal{T}$*, then find a revision set* R *such that* $\max_{\rho \in R} \theta(\rho)$ *is minimized.*

The edges of $G_{\mathcal{A}}$ are labeled by the set of symbols which if removed from the corresponding transition on $\mathcal{B}_{\mathbf{s}}$, they will enable the transition on $\mathcal{A}$. The overall problem then becomes one of finding the least number of symbols to be removed in order for the product graph to have an accepting run.

Definition 19. *Given a system $\mathcal{T}$ and a specification automaton $\mathcal{B}_{\mathbf{s}}$, we define the graph $G_{\mathcal{A}} = (V, E, v_s, V_f, \overline{\Pi}, \Lambda, p)$, which corresponds to the product $\mathcal{A} = \mathcal{T} \times \mathcal{B}_{\mathbf{s}}$ as follows*

- $V = \mathcal{S}$ *is the set of nodes*
- $E = E_{\mathcal{A}} \cup E_D \subseteq \mathcal{S} \times \mathcal{S}$, *where $E_{\mathcal{A}}$ is the set of edges that correspond to transitions on $\mathcal{A}$, i.e., $((q,s),(q',s')) \in E_{\mathcal{A}}$ iff $\exists l \in \mathcal{P}(\Pi)$. $(q,s) \xrightarrow{l}_{\mathcal{A}} (q',s')$; and E_D is the set of edges that correspond to disabled transitions, i.e., $((q,s),(q',s')) \in E_D$ iff $q \rightarrow_{\mathcal{T}} q'$ and $s \xrightarrow{l}_{\mathcal{B}_{\mathbf{s}}} s'$ with $l \cap (\Pi - h_{\mathcal{T}}(q')) \neq \emptyset$*
- $v_s = s_0^{\mathcal{A}}$ *is the source node*
- $V_f = F_{\mathcal{A}}$ *is the set of sinks*
- $\overline{\Pi} = \{\langle \pi, (s,s') \rangle \mid \pi \in \Pi, (s,s') \in E_{\mathcal{B}_{\mathbf{s}}}\}$
- $\Lambda : E \rightarrow \mathcal{P}(\overline{\Pi})$ *is the edge labeling function such that if $e = ((q,s),(q',s'))$, then*

$$\Lambda(e) = \{\langle \pi, (s,s') \rangle \mid \pi \in (\lambda_{\mathcal{B}_{\mathbf{s}}}(s,s') - h_{\mathcal{T}}(q'))\}.$$

- $\theta : \overline{\Pi} \rightarrow \mathbb{R}_{\geq 0}$ *is the preference function of $\mathcal{B}_{\mathbf{s}}$ restricted on $\overline{\Pi}$.*

If $\Lambda(e) \neq \emptyset$, then it specifies those atomic propositions in $\lambda_{\mathcal{B}_{\mathbf{s}}}(s,s')$ that need to be removed in order to enable the edge in $\mathcal{A}$. Again, note that the labels of the edges of $G_{\mathcal{A}}$ are subsets of $\overline{\Pi}$ rather than Π. This is due to the fact that we are looking into

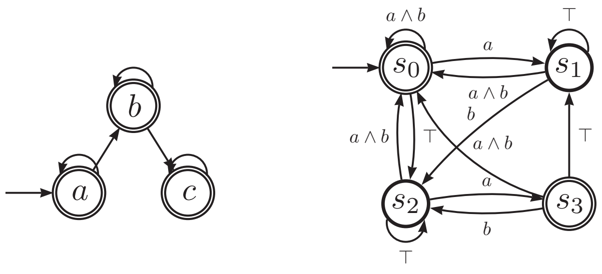


Figure 2.10: The system $\mathcal{T}$ and the specification $\mathcal{B}_{\mathbf{s}}$ of Example 12. The LTL formula of $\mathcal{B}_{\mathbf{s}}$ is $GF(a \wedge Fb)$.

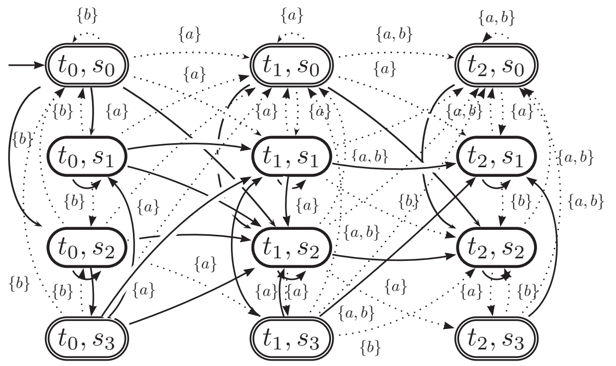


Figure 2.11: The cross-product automaton $\mathcal{T} \times \mathcal{B}_{\mathbf{s}}$ with relaxations. Solid transition are for valid transitions and dotted transitions are for relaxed transitions.

removing an atomic proposition π from a specific transition (s, l, s') of $\mathcal{B}_{\mathbf{s}}$ rather than all occurrences of π in $\mathcal{B}_{\mathbf{s}}$.

Consider now a path that reaches an accept state and then can loop back to the same accept state. The set of labels of the path is a revision set R that corresponds to some $\mathcal{B} \in \mathfrak{R}(\mathcal{B}_{\mathbf{s}}, \mathcal{T})$. This is immediate by the definition of the graph $G_{\mathcal{A}}$. Thus, our goal is to solve the Min-Sum and Min-Max revision problems on this graph.

Example 12. *Let us consider the system $\mathcal{T}$ in Fig 2.10. The LTL formula of the specification $\mathcal{B}_{\mathbf{s}}$ in Fig. 2.10 is $GF(a \wedge Fb)$[2]. Informally, the specification is 'Infinitely often visit a and then visit b'. Fig. 2.11 is the cross-product automaton*

[2]For LTL semantics, see Def. 3 in Sec. 2.2.1.

$\mathcal{T} \times \mathcal{B}_\mathbf{s}$. The initial state of the cross-product automaton is (t_0, s_0). The final states are $(t_0, s_0), (t_1, s_0), (t_2, s_0), (t_0, s_3), (t_1, s_3), (t_2, s_3)$. $\mathcal{B}_\mathbf{s}$ is not satisfiable on $\mathcal{T}$ so that there is no reachable path from the state (t_0, s_0) to one of the finals and from one of the final states to back to itself. In this example, the set of atomic propositions is $\Pi = \{a, b, c\}$. Suppose that the preference levels of the atomic propositions are $\theta((s_i, s_j), \{a\}) = 3$, $\theta((s_i, s_j), \{b\}) = 5$, $\theta((s_i, s_j), \{c\}) = 4$ where $\forall s_i, s_j \in S_\mathcal{B}$. Then from valid relaxations of $\mathcal{B}_\mathbf{s}$, we can find acceptable paths as follows: $p_1 = \langle(((t_0, s_0),\ \{b\},\ (t_0, s_0))\ ((t_0, s_0),\ \{b\},\ (t_0, s_0))\ \ldots\rangle$, $p_2 = \langle(((t_0, s_0),\ \emptyset,\ (t_0, s_1))\ ((t_0, s_1),\ \{b\},\ (t_0, s_0))\ ((t_0, s_0),\ \emptyset,\ (t_0, s_1))\ \ldots\rangle$, $p_3 = \langle(((t_0, s_0),\ \{a\},\ (t_1, s_0))\ ((t_1, s_0),\ \{a\},\ (t_1, s_0))\ ((t_1, s_0),\ \{a\},\ (t_1, s_0))\ldots\rangle$, $p_4 = \langle(((t_0, s_0),\ \{a\},\ (t_1, s_0))\ ((t_1, s_0),\ \{a\},\ (t_1, s_0))\ ((t_1, s_0),\ \{a, b\},\ (t_2, s_0))\ ((t_2, s_0),\ \{a, b\},\ (t_2, s_0))\ldots\rangle$, etc. The sum of preference levels of each path are 5, 5, 3, 8, respectively. The max of preference levels of each path are 5, 5, 3, 5. Therefore, among the above paths, the path having atomic propositions that minimize the sum of preference levels is p_3. It has only $\{a\}$ on the transitions, so the sum of preference level of the path is 3 and the max of preference level of the path is also 3. $\triangle$

First, we study the computational complexity of the two problems by restricting the search problem only to paths from source (initial state) to sink (accept state). Let $Paths(G_\mathcal{A})$ denote all such paths on $G_\mathcal{A}$. We indicate that the graph search equivalent problem of Problem 5 is in P. Given a path $p = v_s v_1 v_2 \ldots v_f$ on $G_\mathcal{A}$ with $v_f \in V_f$, we define the max-preference level of the path to be:

$$\theta_{\max}(p) = \max_{(v_i, v_{i+1}) \in p} \theta(\Lambda(v_i, v_{i+1}))$$

Note that this is the same as the original cost function in Problem 5 since clearly $\max_{(v_i, v_{i+1}) \in p} \theta(\Lambda(v_i, v_{i+1})) = \max_{\rho \in R} \theta(\rho)$ where $R = \cup_{(v_i, v_{i+1}) \in p} \Lambda(v_i, v_{i+1})$. Thus, Problem 5 is converted into the following optimization problem:

$$p^* = \arg \min_{p \in Paths(G_\mathcal{A})} \theta(p) \tag{2.3}$$

And, thus, the revision will be $R = \cup_{(v_i,v_{i+1})\in p^*}\Lambda(v_i, v_{i+1})$. Now, we recall the weak optimality principle [83].

Definition 20 (Weak optimality principle)**.** *There is an optimal path formed by optimal subpaths.*

Proposition 8. *The graph search equivalent of Problem 5 satisfies the weak optimality principle.*

Proof. Let p^* be an optimal path under the cost function $\theta_{\max}$, that is, for any other path p, we have $\theta_{\max}(p) \geq \theta_{\max}(p*)$. We assume that p^* is a loopless path. Notice if a loop exists, then it can be removed without affecting the cost of the path. Let p^* have a subpath $p_s = v_1v_2 \dots v_{i-1}v_i$ which is not optimal, that is $p* = p_1 \circ p_s \circ p_2$. We use here the notation $p_1 \circ p_2$ to indicate that the last vertex of p_1 and the first vertex of p_2 are the same and are going to be merged. Now assume that there is another subpath $p'_s = v_1v'_2 \dots v'_{j-1}v_i$ such that $\theta_{\max}(p_s) > \theta_{\max}(p'_s)$. Note that $\theta_{\max}(p_s) \leq \theta_{\max}(p_1)$ and $\theta_{\max}(p_s) \leq \theta_{\max}(p_2)$ otherwise p^* would not be optimal. We have $\theta_{\max}(p_s) = \max(\theta_{\max}(p_1), \theta_{\max}(p_s), \theta_{\max}(p_2)) = \max(\theta_{\max}(p_1), \theta_{\max}(p'_s), \theta_{\max}(p_2)) = \theta_{\max}(p_1 \circ p'_s \circ p_2)$. Hence, the path $p_1 \circ p'_s \circ p_2$ is also optimal. If this process is repeated, we can construct an optimal path p^{**} that contains only optimal subpaths. □

The importance of the weak optimality principle being satisfied is that label correcting and label setting algorithms can be applied to such problems [83]. Dijkstra's algorithm is such an algorithm [84] and, thus, it can provide an exact solution to the problem.

Now, we proceed to the Min-Sum preference problem. Given a path $p = v_sv_1v_2 \dots v_f$ on $G_{\mathcal{A}}$ with $v_f \in V_f$, we define the sum-preference level of the path to be:

$$\theta_+(p) = \sum\{\theta(\rho) \mid \rho \in \cup_{(v_i,v_{i+1})\in p}\Lambda(v_i, v_{i+1})\}$$

and if we are directly provided with a revision set R, then

$$\theta_+(R) = \sum_{\rho \in R} \theta(\rho)$$

Problem 6. *Labeled Path under Additive Preferences (LPAP).*

- INPUTS: *A graph $G_{\mathcal{A}} = (V, E, v_s, V_f, \overline{\Pi}, \Lambda, \theta)$, and a preference bound $K \in \mathbb{N}$.*
- OUTPUT: *a set $R \subseteq \overline{\Pi}$ such that removing all elements in R from edges in E enables a path from v_s to some final vertex $v_f \in V_f$ and $\theta_+(R) \leq K$.*

We can show that the corresponding decision problem is NP-Complete.

Theorem 5. *Given an instance of the LPAP $(G_{\mathcal{A}}, K)$, the decision problem of whether there exists a path p such that $\theta_+(p) \leq K$ is NP-Complete.*

Proof (Sketch): Clearly, the problem is in NP since given a sequence of nodes p, we can verify in polynomial time that p is a path on $G_{\mathcal{A}}$ and $\theta_+(p) \leq K$.

The problem is NP-hard since we can easily reduce the revision problem without preferences (see Sec. 2.3) to this one by setting the preference levels of all atomic propositions equal to 1. Then, since all atomic propositions have the same preference level which is 1, it becomes the problem to find the minimal number of atomic propositions of the graph. □

2.5.1 Algorithms for the Revision Problem with Preferences

In this section, we present Algorithms for the Revision Problem with Preferences (ARPP). It is based on the Approximation Algorithm of the Minimal Revision Problem (AAMRP) in Sec. 2.3.2 which is in turn based on Dijkstra's shortest path algorithm [84]. The main difference from AAMRP is that instead of finding the minimum number of atomic propositions that must be removed from each edge on the paths of the graph

$G_{\mathcal{A}}$, ARPP tracks paths having atomic propositions that minimize the preference level from each edge on the paths of the graph $G_{\mathcal{A}}$.

Here, we present the pseudocode for ARPP. ARPP is similar to AAMRP in Sec. 2.3.2. The difference is that AARP uses PREF function instead of using cardinality of the set. For Min-Sum Revision, the function PREF: $\overline{\Pi} \rightarrow \mathbb{R}_{\geq 0}$ is defined as following: given a set of label $R \subseteq \overline{\Pi}$ and the preference function $\theta_+ : \overline{\Pi} \rightarrow \mathbb{R}_{\geq 0}$,

$$\textsc{Pref}(R) = \theta_+(R).$$

The Min-Sum ARPP is denoted by $ARPP_+$.

For Min-Max Revision, the function PREF: $\overline{\Pi} \rightarrow \mathbb{R}_{\geq 0}$ is defined as following: given a set of label $R \subseteq \overline{\Pi}$ and the preference function $\theta : \overline{\Pi} \rightarrow \mathbb{R}_{\geq 0}$,

$$\textsc{Pref}(R) = \max_{\rho \in R} \theta(\rho).$$

The Min-Max ARPP is denoted by $ARPP_{max}$.

The main algorithm (Alg. 12) divides the problem into two tasks. First, in line 4, it finds an approximation to the minimum preference level of atomic propositions from $\overline{\Pi}$ that must be removed to have a prefix path to each reachable sink (see Def. 6 and 7 in Sec. 2.2.1). Then, in line 9, it repeats the process from each reachable final state to find an approximation to the minimum preference level of atomic propositions from $\overline{\Pi}$ that must be removed so that a lasso path is enabled. The combination of prefix/lasso that removes the least preferable atomic propositions is returned to the user.

Algorithm 13 follows closely Dijkstra's shortest path algorithm [77]. It maintains a list of visited nodes $\mathcal{V}$ and a table $\mathcal{M}$ indexed by the graph vertices which stores the set of atomic propositions that must be removed in order to reach a particular node on the graph. Given a node v, the preference level of the set $\mathcal{M}[v, 1]$ is an upper bound on the minimum preference level of atomic propositions that must be removed.

```
Algorithm 12: ARPP($G_{\mathcal{A}}$)
   Input: a graph $g_{\mathcal{A}} = (v, e, v_s, v_f, \overline{\pi}, \lambda, p)$.
   Output: the list $L$ of symboles from $\overline{\Pi}$ that must be removed from $\mathcal{B}_{\mathbf{s}}$.
 1 $L \leftarrow \overline{\Pi}$
 2 $\mathcal{M}[:,:] \leftarrow (\overline{\Pi}, \infty)$                ▷ Each row is set to $(\overline{\Pi}, \infty)$
 3 $\mathcal{M}[v_s,:] \leftarrow (\emptyset, 0)$                  ▷ Initialize the source node
 4 $\langle \mathcal{M}, \mathbf{P}, \mathcal{V} \rangle \leftarrow$ FindMinPath($G_{\mathcal{A}}, \mathcal{M}, 0$)
 5 if $\mathcal{V} \cap V_f = \emptyset$ then
 6     $L \leftarrow \emptyset$
 7 else
 8     for $v_f \in \mathcal{V} \cap V_f$ do
 9         $L_p \leftarrow$ GetAPFromPath($v_s, v_f, \mathcal{M}, \mathbf{P}$)
10         $\mathcal{M}'[:,:] \leftarrow (\overline{\Pi}, \infty)$
11         $\mathcal{M}'[v_f,:] \leftarrow \mathcal{M}[v_f,:]$
12         $G'_{\mathcal{A}} \leftarrow (V, E, v_f, \{v_f\}, \overline{\Pi}, L)$
13         $\langle \mathcal{M}', \mathbf{P}', \mathcal{V}' \rangle \leftarrow$ FindMinPath($G'_{\mathcal{A}}, \mathcal{M}', 1$)
14         if $v_f \in \mathcal{V}'$ then
15             $L_l \leftarrow$ GetAPFromPath($v_f, v_f, \mathcal{M}', \mathbf{P}'$)
16             if Pref($L_p \cup L_l$) $\leq$ Pref($L$) then
17                 $L \leftarrow L_p \cup L_l$
18 return $L$
```

The function GetAPFromPath($(v_s, v_f, \mathcal{M}, \mathbf{P})$) returns the atomic propositions that must be removed from $\mathcal{B}_{\mathbf{s}}$ in order to enable a path on $\mathcal{A}$ from a starting state v_s to a final state v_f given the tables $\mathcal{M}$ and $\mathbf{P}$.

That is, if we remove all $\overline{\pi} \in \mathcal{M}[v, 1]$ from $\mathcal{B}_{\mathbf{s}}$, then we enable a simple path (i.e., with no cycles) from a starting state to the state v. The preference level of $|\mathcal{M}[v, 1]|$ is stored in $\mathcal{M}[v, 2]$ which also indicates that the node v is reachable when $\mathcal{M}[v, 2] < \infty$.

The algorithm works by maintaining a queue with the unvisited nodes on the graph. Each node v in the queue has as key the summed preference level of atomic propositions that must be removed so that v becomes reachable on $\mathcal{A}$. The algorithm proceeds by choosing the node with the minimally summed preference level of atomic propositions discovered so far (line 14). Then, this node is used in order to updated the estimates for the minimum preference level of atomic propositions needed in order to reach its neighbors (line 18). A notable difference of Alg. 13 from Dijkstra's shortest

Algorithm 13: FindMinPath($G_{\mathcal{A}}$,$\mathcal{M}$,*lasso*)

```
Input: a graph $G_{\mathcal{A}} = (V, E, v_s, V_f, \overline{\Pi}, \Lambda, p)$, a table $\mathcal{M}$ and a flag $lasso$ on
       whether this is a lasso path search.
Output: the tables $\mathcal{M}$ and $\mathbf{P}$ and the visited nodes $\mathcal{V}$
Variables: (a queue $\mathcal{Q}$, a set $\mathcal{V}$ of visited nodes and a table $\mathbf{P}$ indicating the
  parent of each node on a path.)
1  $\mathcal{V} \leftarrow \{v_s\}$
2  $\mathbf{P}[:] \leftarrow \emptyset$                                   ▷ Each entry of $\mathbf{P}$ is set to $\emptyset$
3  $\mathcal{Q} \leftarrow V - \{v_s\}$
4  for $v \in V$ such that $(v_s, v) \in E$ and $v \neq v_s$ do
5      $\langle \mathcal{M}, \mathbf{P} \rangle \leftarrow$ Relax($(v_s, v), \mathcal{M}, \mathbf{P}, \Lambda$)
6  if $lasso = 1$ then
7      if $(v_s, v_s) \in E$ then
8          $\mathcal{M}[v_s, 1] \leftarrow \mathcal{M}[v_s, 1] \cup \Lambda(v_s, v_s)$
9          $\mathcal{M}[v_s, 2] \leftarrow$ Pref($\mathcal{M}[v_s, 1] \cup \Lambda(v_s, v_s)$)
10         $\mathbf{P}[v_s] \leftarrow v_s$
11     else
12         $\mathcal{M}[v_s, :] \leftarrow (\overline{\Pi}, \infty)$
13 while $\mathcal{Q} \neq \emptyset$ do
14     $u \leftarrow$ ExtractMIN($\mathcal{Q}$)                        ▷ Get node $u$ with minimum $\mathcal{M}[u, 2]$
15     if $\mathcal{M}[u, 2] < \infty$ then
16         $\mathcal{V} \leftarrow \mathcal{V} \cup \{u\}$
17         for $v \in V$ such that $(u, v) \in E$ do
18             $\langle \mathcal{M}, \mathbf{P} \rangle \leftarrow$ Relax($(u, v), \mathcal{M}, \mathbf{P}, \Lambda$)
19 return $\mathcal{M}$, $\mathbf{P}$, $\mathcal{V}$
```

path algorithm is the check for lasso paths in lines 6-12. After the source node is used for updating the estimates of its neighbors, its own estimate for the minimum preference level of atomic propositions is updated either to the value indicated by the self loop or the maximum possible preference level of atomic propositions. This is required in order to compare the different paths that reach a node from itself.

Correctness: The correctness of the algorithm ARPP is based upon the fact that a node $v \in V$ is reachable on $G_{\mathcal{A}}$ if and only if $\mathcal{M}[v, 2] < \infty$. The argument for this claim is similar to the proof of correctness of Dijkstra's shortest path algorithm in [77]. If this algorithm returns a set of atomic propositions L which removed from $\mathcal{B}_{\mathbf{s}}$,

```
Algorithm 14: Relax($(u,v)$,$\mathcal{M}$,$\mathbf{P}$,$\Lambda$)
  Input: an edge $(u,v)$, the tables $\mathcal{M}$ and $\mathbf{P}$ and the edge labeling function $\Lambda$
  Output: the tables $\mathcal{M}$ and $\mathbf{P}$
1 if Pref($\mathcal{M}[u,1] \cup \Lambda(u,v)$) $< \mathcal{M}[v,2]$ then
2     $\mathcal{M}[v,1] \leftarrow \mathcal{M}[u,1] \cup \Lambda(u,v)$
3     $\mathcal{M}[v,2] \leftarrow$ Pref($\mathcal{M}[u,1] \cup \Lambda(u,v)$)
4     $\mathbf{P}[v] \leftarrow u$
5 return $\mathcal{M}$, $\mathbf{P}$
```

then the language $\mathcal{L}(\mathcal{A})$ is non-empty. This is immediate by the construction of the graph $G_{\mathcal{A}}$ (Def. 19).

Running time: The analysis of the algorithm ARPP follows closely the analysis of AAMRP in Sec. 2.3.2. The only difference in the time complexity is that ARPP uses Pref function in order to compute preference levels of all elements in $\overline{\Pi}$. Both Min-Sum Revision and Min-Max Revision take $O(\overline{\Pi})$ since at most they compute preference levels of all elements in $\overline{\Pi}$. Hence, the running time of FindMinPath is $O(E(\overline{\Pi}^2 \log \overline{\Pi} + \log V))$. Therefore, the running time of ARPP is $O(V_f(V\overline{\Pi}\log\overline{\Pi} + E(\overline{\Pi}^2 \log \overline{\Pi} + \log V))) = O(V_f E(\overline{\Pi}^2 \log \overline{\Pi} + \log V))$ which is polynomial in the size of the input graph.

2.5.2 Result

In this section, we present an example scenario and experimental results using our prototype implementation of algorithms and brute-force search.

In the following example, we will be using LTL as a specification language. We remark that the results presented here can be easily extended to LTL formulas by renaming repeated occurrences of atomic propositions in the specification and adding them on the transition system (for details, see [82]).

The following example scenario was inspired by [13, 80], and we will be using LTL as a specification language.

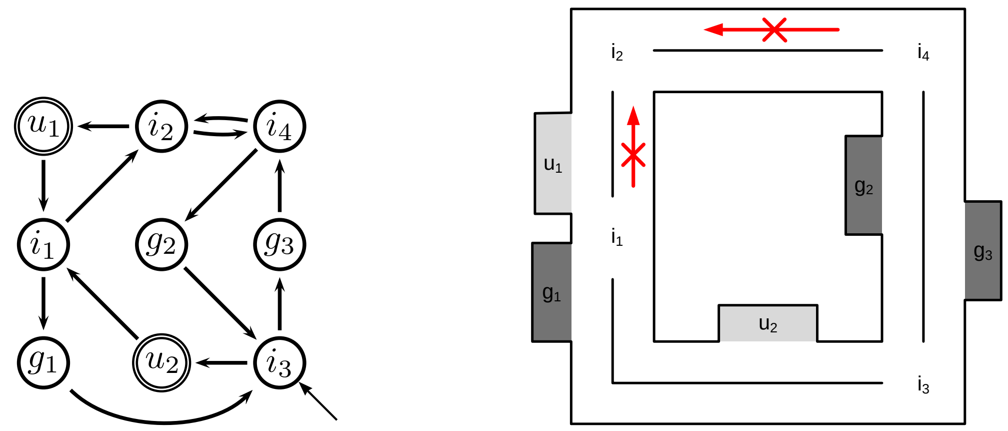


Figure 2.12: Illustration of the simple road network environment of Example 13. The robot is required to drive right-side of the road.

Example 13 (Single Robot Data Gathering Task)**.** *In this example, we use a simplified road network having three gathering locations and two upload locations with four intersections of the road. In Fig. 2.12, the data gather locations, which are labeled g_1, g_2, and g_3, are dark gray, the data upload locations, which are labeled u_1 and u_2, are light gray, and the intersections are labeled i_1 through i_4. In order to gather data and upload the gather-data persistently, the following LTL formula may be considered: $\phi_A := GF(\varphi) \wedge GF(\pi)$, where $\varphi := g_1 \vee g_2 \vee g_3$ and $\pi := u_1 \vee u_2$. The following formula can make the robot move from gather locations to upload locations after gathering data: $\phi_G := G(\varphi \rightarrow X(\neg\varphi \mathcal{U} \pi))$. In order for the robot to move to gather location after uploading, the following formula is needed: $\phi_U := G(\pi \rightarrow X(\neg\pi \mathcal{U} \varphi))$.*

Let us consider that some parts of road are not recommended to drive from gather locations, such as from i_4 to i_2 and from i_1 to i_2. We can describe those constraints as follows: $\psi_1 := G(g_1 \rightarrow \neg(i_4 \wedge Xi_2)\mathcal{U}u_1)$ and $\psi_2 := G(g_2 \rightarrow \neg(i_1 \wedge Xi_2)\mathcal{U}u_2)$. If the gathering task should have an order such as $g_3, g_1, g_2, g_3, g_1, g_2, \ldots$, then the following formula could be considered: $\phi_O := ((\neg g_1 \wedge \neg g_2)\mathcal{U}g_3) \wedge G(g_3 \rightarrow X((\neg g_2 \wedge \neg g_3)\mathcal{U}g_1)) \wedge G(g_1 \rightarrow X((\neg g_1 \wedge \neg g_3)\mathcal{U}g_2)) \wedge G(g_2 \rightarrow X((\neg g_1 \wedge \neg g_2)\mathcal{U}g_3))$. Now, *we can informally describe the mission. The mission is "Always gather data from* g_3, g_1, g_2 *in this order and upload the collected data to* u_1 *and* u_2. *Once data gathering is finished, do not visit gather locations until the data is uploaded. Once uploading is finished, do not visit upload locations until gathering data. You should always avoid the road from* i_4 *to* i_2 *when you head to* u_1 *from* g_1 *and from* i_1 *to* i_2 *when you head to* u_2 *from* g_2*". The following formula represents this mission:*

$$\phi_{single} := \phi_O \wedge \phi_G \wedge \phi_U \wedge \psi_1 \wedge \psi_2 \wedge \; GF(\pi).$$

Assume that initially, the robot is in i_3 *and final nodes are* u_1 *and* u_2. *When we made a cross product with the road and the specification, we could get 36824 states, 350114 transitions and 100 final states. Not removing some atomic propositions, the specification was not satisfiable.*

We tested two different preference levels. For clarity in presentation, we omit for presenting preference levels on each transition since we set for all the occurances of the same symbols the same preference level, we abuse notation and write $\theta(\pi)$ *instead of* $\theta(\pi, (s_i, s_j))$. *However, the revision is for specification transitions. First, the preference level of the symbols are as follows: for* g_1, g_2, g_3, u_1, u_2, i_1, i_2, i_3, i_4, *the preference levels are 3, 4, 5, 20, 20, 1, 1, 1, 1, respectively, and for* $\neg g_1$, $\neg g_2$, $\neg g_3$, $\neg u_1$, $\neg u_2$, $\neg i_1$, $\neg i_2$, $\neg i_3$, $\neg i_4$, *the preference levels are 3, 4, 5, 20, 20, 1, 1, 1, 1, respectively. ARPP for Min-Sum Revision took 210.979 seconds, and suggested removing* $\neg g_1$ *and* $\neg i_4$. *The total returned preference was 4 since* $\theta(\neg g_1) = 3$ *and* $\theta(\neg i_4) = 1$. *The sequence of the locations suggested by ARPP is* $i_3 g_3 i_2 u_1 (i_1 g_1 i_3 u_2 i_1 i_2 i_4 g_2 i_3 u_2 i_1 g_1 i_3 g_3 i_4 i_2 u_1)^+$. *We can check that* $\neg g_1$ *is from* $G(g_2 \rightarrow X((\neg g_1 \wedge \neg g_2)\mathcal{U} g_3))$ *of the formula* ϕ_O *and from* $\neg\varphi = \neg(g_1 \vee g_2 \vee g_3)$ *of the formula* $\phi_G = G(\varphi \rightarrow (\neg\varphi \mathcal{U} \pi))$, *and* $\neg i_4$ *is from* $G(g_1 \rightarrow \neg(i_4 \wedge X i_2)\mathcal{U} u_1)$ *of the formula* ψ_1. *AARP for Min-Max Revision took 239 seconds, and returned* g_1, $\neg g_1$, $\neg i_1$, *and* $\neg i_4$. *The maximum returned preference was 3*

Nodes	Brute-Force				Min-Sum Revision				RATIO		
	min	avg	max	succ	min	avg	max	succ	min	avg	max
9	0.033	0.0921	0.945	200/200	0.019	0.183	0.874	200/200	1	1	1
100	0.065	0.3707	3.997	200/200	0.065	0.1598	2.66	200/200	1	1.003	1.619
196	0.278	303.55	11974	199/200	0.137	0.4927	12.057	200/200	1	1.0014	1.1475

Table 2.4: Numerical Experiments: Number of nodes versus the results of Brute-Force Search Algorithm and ARPP for Min-Sum Revision. Under the Brute-Force and Min-Sum Revision columns the numbers indicate computation times in sec. RATIO indicates the experimentally observed approximation ratio to the optimal solution.

since $\theta(g_1) = 3$ *and* $\theta(\neg g_1) = 3$.

In the second case, the preference level of the positive atomic propositions are same as the first test, and the preference level of the negative atomic propositions are as follow: for $\neg g_1$, $\neg g_2$, $\neg g_3$, $\neg u_1$, $\neg u_2$, $\neg i_1$, $\neg i_2$, $\neg i_3$, $\neg i_4$, *the preference levels are 3, 4, 5, 20, 20, 10, 10, 10, 10, respectively. In this case, ARPP for Min-Sum Revision took 207.885 seconds, and suggested removing* g_3. *The total returned preference was 5 since* $\theta(g_3) = 5$. *The sequence of the locations suggested by ARPP is* $i_3 g_3 i_4 i_2 u_1 (i_1 g_1 i_3 u_2 i_1 i_2 i_4 g_2 i_3 u_2 i_1 i_2 u_1)^+$. *We can check that* g_3 *is from* $G(g_3 \rightarrow X((\neg g_2 \wedge \neg g_3)\mathcal{U} g1))$ *of the formula* ϕ_O *and from* $\varphi = (g_1 \vee g_2 \vee g_3)$ *of the formula* $\phi_U = G(\phi \rightarrow X(\neg\phi\mathcal{U}\varphi)$. *ARPP for Min-Max Revision took 214.322 seconds, and returned* g_1 *and* $\neg g_1$. *The maximum preference was 3 since* $\theta(g_1) = 3$ *and* $\theta(\neg g_1) = 3$. $\triangle$

Now, we present some experimental results. The propotype implementation is written in Python. For the experiments, we utilized the ASU super computing center which consists of clusters of Dual 4-core processors, 16 GB Intel(R) Xeon(R) CPU X5355 @2.66 Ghz. Our implementation does not utilize the parallel architecture. The clusters were used to run the many different test cases in parallel on a single core.

Nodes	Min-Sum Revision ($ARPP_+$)				Min-Max Revision ($ARPP_{max}$)				RATIO1			RATIO2		
	min	avg	max	succ	min	avg	max	succ	min	avg	max	min	avg	max
9	0.019	0.183	0.874	200/200	0.02	0.0508	0.66	200/200	1	1.2677	3.4	1	1.0007	1.1428
100	0.065	0.1598	2.66	200/200	0.061	0.1258	0.471	200/200	1	1.441	5.97	1	1.0264	1.3928
196	0.137	0.4927	12.057	200/200	0.139	0.29824	0.74	200/200	1	1.4876	5.634	1	1.0389	2.1904

Table 2.5: Numerical Experiments: For each graph $G_{\mathcal{A}}$, Number of nodes versus the results of ARPP for Min-Sum Revision ($ARPP_+$) and ARPP for Min-Max Revision ($ARPP_{max}$). Under the Min-Sum Revision and Min-Max Revision columns the numbers indicate computation times in sec. RATIO1 indicates $\sum(\theta(ARRP_{max}(G_{\mathcal{A}})))/\sum(\theta(ARPP_+(G_{\mathcal{A}})))$. RATIO2 indicates $max(\theta(ARRP_+(G_{\mathcal{A}}))/max(\theta(ARPP_{max}(G_{\mathcal{A}})))$.

The operating system is CentOS release 5.9.

In order to assess the experimental approximation ratio of the heuristic (Min-Sum Revision), we compared the solutions returned by the heuristic with Brute-force search algorithm. The Brute-force search is guaranteed to return a minimal solution to the Min-Sum Revision problem.

We performed a large number of experimental comparisons on random benchmark instances of various sizes. Each test case consisted of two randomly generated DAGs which represented an environment and a specification. Both graphs have self-loops on their leaf nodes so that a feasible lasso path can be found. The number of atomic propositions in each instance was equal to four times the number of nodes in each acyclic graph. For example, in the benchmark where the graph had 9 nodes, each DAG had 3 nodes, and the number of atomic propositions was 12. The final nodes are chosen randomly and they represent 5%-40% of the nodes. The number of edges in most instances were 2-3 times more than the number of nodes.

Table 2.4 compares the results of the Brute-Force Search Algorithm with the results of ARPP for Min-Sum Revision on test cases of different sizes (total number of nodes). For each graph size, we performed 200 tests and we report minimum, average, and maximum computation times in sec. Both algorithms were able to finish the computation and return a minimal revision for instances having 9 nodes and 100 nodes. However, for instances having 196 nodes, the Brute-Force Search Algorithm had one failed instance which exceeded the 2 hrs window limit. In the large problem instances, ARPP for Min-Sum Revision achieved a 600 time speed-up on the average running time.

In Table 2.5, we present two ratios. RATIO1 captures the ratios between the sum of preference levels of the set returned by $ARPP_{max}$ over the sum of preference levels of the set returned by $ARPP_{+}$. On the other hand, RATIO2 captures the ratios between the max of preference levels of the set returned by $ARPP_{+}$ over the max of preference levels of the set returned by $ARPP_{max}$. If the $ARPP_{+}$ was always returning the optimal solution, then RATIO1 should always be greater than 1. We observe on the random graph instances that the result also holds for this particular class of random graphs. Moreover, there were graph instances where $ARPP_{+}$ returned much smaller total preference sum then $ARPP_{max}$. Importantly, when received the results for RATIO2, we observe that there exist graph instances where $ARPP_{max}$ returned a revision set with maximum much less then the maximum preference in the set returned by $ARPP_{+}$. Thus, depending on the user application it could be desirable to utilize either revision criterion.

Table 2.6 shows the comparison between the number of atomic propositions of the set returned from ARPP for Min-Sum Revision ($ARPP_{+}$) and the number of atomic propositions of the set returned from ARPP for Min-Max Revision ($ARPP_{max}$). The columns under the avg columns of Min-Sum and Min-Max indicate the average number of atomic propositions of the set returned from $ARPP_{+}$ and $ARPP_{max}$ for graph

Nodes	Min-Sum	Min-Max		RATIO	
	avg	avg	min	avg	max
9	1.305	1.785	0.66	1.423	5
100	1.95	3.215	1	1.8056	6
196	2.305	3.84	1	1.7793	8

Table 2.6: Numerical Experiments: Number of nodes versus the results of ARPP for Min-Sum Revision ($ARPP_{+}$) and ARPP for Min-Max Revision ($ARPP_{max}$).

instances having 9 nodes, 100 nodes, and 196 nodes. The RATIO captures the ratios between the number of atomic propositions of the set returned by $ARPP_{max}$ over the number of atomic propositions of the set returned by $ARPP_{+}$. Even though Min-Sum Revision and Min-Max Revision do not count the number of atomic propositions while relaxing, this result shows readers how many atomic propositions each algorithm returns. From the fact that the avg of the RATIO for all random graph instances is greater than 1, we observe that the set returned from Min-Max Revision in general has more number of atomic propositions than the set returned from Min-Sum Revision.

2.6 Multi-agent LTL Planning Problem

In this section, we will introduce Multi-agent LTL Planning Problem. Throughout this section, we define the following:

Definition 21. *(Multi-agent LTL Planning)*

- *Given that there are h robots, $\mathcal{R} = \{1, 2, \ldots, h\} \subset \mathbb{N}$ is a set of indices for h robots.*
- *For each robot $i \in \mathcal{R}$, $\mathcal{B}_i = (S_{\mathcal{B}_i}, s_0^{\mathcal{B}_i}, \mathcal{P}(\Pi), \rightarrow_{\mathcal{B}_i}, F_{\mathcal{B}_i})$ is the corresponding specification automaton.*
- *A system $\mathcal{T} = (Q, Q_0, \rightarrow_{\mathcal{T}}, h_{\mathcal{T}}, \Pi)$ (as defined in Def. 1).*

- *For each robot $i \in \mathcal{R}$, given $\mathcal{T}$ and $\mathcal{B}_i$, $\mathcal{A}_i = \mathcal{T} \times \mathcal{B}_i = (S_{\mathcal{A}_i}, s_0^{\mathcal{A}_i}, \mathcal{P}(\Pi), \rightarrow_{\mathcal{A}_i}, F_{\mathcal{A}_i})$ is a product automaton (as defined in Def. 5).*

- *For each robot $i \in \mathcal{R}$, given $\mathcal{A}_i$, an non-empty accepting path $p[i]$ can be computed where $p[i] \in \mathcal{L}(\mathcal{A}_i)$ is ultimately periodic (as defined in Def. 6).*

We remark that a system $\mathcal{T}$ is the original environment.

We also remark that a robot's path can be indexed by its time step. For example, for a robot $i \in \mathcal{R}$ and a time step k, $p_k[i]$ indicates a state on the path $p[i]$ at the time step k. In addition, $p_{k,j}[i]$ indicates a set of states on the path $p[i]$ from the step k to the step j.

Given a path $p[i] \in \mathcal{L}(\mathcal{A}_i)$ for some robot $i \in \mathcal{R}$, the path $p[i] = p_0, p_1, \ldots$ is a sequence of states in $\mathcal{A}_i$. Since $\mathcal{A}_i = \mathcal{T} \times \mathcal{B}_i$, this path also can be represented by a corresponding sequence of states in Q of $\mathcal{T}$. Now, we can define a function to show the progression of robots in states of its system $\mathcal{T}$.

Definition 22. *Given $i \in \mathcal{R}$, $\mathcal{T}$, $\mathcal{A}_i$, a path $p[i] \in \mathcal{L}(\mathcal{A}_i)$ and a time step $k \in \mathbb{N}$, we define the location function $S_k : \mathcal{R} \rightarrow Q$.*

We remark that since $p[i]$ is *ultimately periodic*, every $k \in \mathbb{N}$ can be mapped to some $p_k[i]$ and for this $p_k[i]$ there exist corresponding states in both $\mathcal{T}$ and $\mathcal{B}_i$. In addition, each robot has a planner that can compute a shortest path, $P(u, v)$ that moves a robot from vertex u to v. The length of $P(u, v)$ is denoted as $\mathcal{C}(u, v)$, i.e., $\mathcal{C}(u, v) = |P(u, v)|$. We have the following assumptions:

Assumption 3. *(Assumptions of Cooperative pathfinding robots)*

- *Robots are homogeneous and have the same sensing and communication range.*

- *Robots are equipped with a communication protocol that allows them to efficiently relay messages.*

- *Time steps are synchronized.*

- *Each robot has full knowledge of the environment, i.e.* $\mathcal{T} = (Q, Q_0, \rightarrow_{\mathcal{T}}, h_{\mathcal{T}}, \Pi)$.

- *Each robot has a different start location and a different goal location.*

We also assume that each robot mission is independent. Hence, there is no cooperative task between any robots.[3]

A *conflict* happens at time step k, if two robots are in the same location, or their locations at $k-1$ are exchanged.

Formally,

$$\mathcal{S}_k[i] = \mathcal{S}_k[j] \vee (\mathcal{S}_k[i] = \mathcal{S}_{k-1}[j] \wedge \mathcal{S}_{k-1}[i] = \mathcal{S}_k[j]) \tag{2.4}$$

where $i, j \in \mathcal{R}$, $i \neq j$ and $k \in \mathbb{N}$. Then, given a time step k and two different robots $i, j \in \mathcal{R}$, *Conflict* function can be defined as following:

$$Conflict(k, i, j) := \begin{cases} \top & \text{if Eq. (2.4) is true} \\ \bot & \text{otherwise.} \end{cases}$$

Example 14 (Possible Conflict). *Consider a set of robots $r_1, \ldots, r_4$ in Fig. 2.13 which is represented to $\mathcal{R} = \{1, 2, 3, 4\}$. Suppose that now there are at the time step $k-1$, and they are following their own path $p_{k-1,k}[i]$ where $i \in \{1, 2, 3, 4\}$. Then, robot r_3 and robot r_4 have a conflict plan at the next time step which is time step k. This is because $S_k[3] = S_k[4]$. Then, $Conflict(k, 3, 4) = \top$.*

Problem 7. *Given a system $\mathcal{T}$, a set of robots $\mathcal{R}$, for each robot $i \in \mathcal{R}$, its corresponding specification automaton $\mathcal{B}_i$ and a product automaton $\mathcal{A}_i = \mathcal{T} \times \mathcal{B}_i$, find a set of paths p such that*

[3]We explain how each agent coordinates collaboratively in this environment later in this section while covering pessimistic decoupling (Sec. 2.6.1).

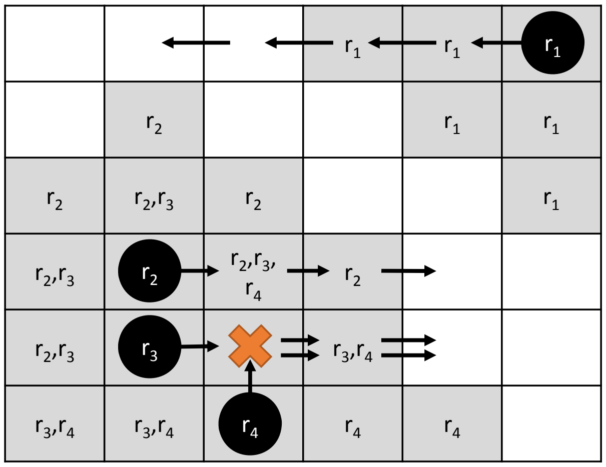


Figure 2.13: This shows a possible conflict between robot r_3 and r_4 at the next time step. Black arrows represent each robot's plan. The r_i in gray cells represents the communication and sensing range of each robot r_i. This limited range makes r_2, r_3 and r_4 being separated with r_1. Hence, these two groups cannot recognize each other group.

- $\forall i \in \mathcal{R}, \exists p[i] \in \mathcal{L}(\mathcal{A}_i)$ *s.t.* $p[i]$ *is ultimately perodic, and*
- $\forall i, j \in \mathcal{R}$ *where* $i \neq j$, $\forall k \in \mathbb{N}$, $\neg \mathit{Conflict}(k, i, j)$.

If we consider a centralized approach in order to solve Problem 7 and we utilize the same approach for the LTL planning as we covered in Sec. 2.2.1, we have to have a combined $\mathcal{T}$ and a carefully designed LTL formula for each robot. For the 4 robots in Fig. 2.13, the combined $\mathcal{T}$ can be constructed by $\mathcal{T}_{r_1} \times \mathcal{T}_{r_2} \times \mathcal{T}_{r_3} \times \mathcal{T}_{r_4}$. With LTL formulas for these robots, there should be additional constraints to avoid a possible collision among these robots. Getting more robots in the system $\mathcal{T}$, the planning is computationally difficult. In general, for multi-robots planning, the complexity is PSPACE-hard even for relatively simple settings [85]. Therefore, a decentralized and on-line approach should be considered.

2.6.1 DisCoF

We introduce a window-based approach, called DisCoF, for cooperative pathfinding in distributed systems with limited sensing and communication range. In DisCoF, the window size corresponds to the sensing range of the robots. Robots can commmunicate with each other either directly if in range or indirectly if out of range. In the latter case, it is still possible to communicate indirectly through other robots using a communication relay protocol. This allows for coordination beyond a single robot's sensor range. To ensure completeness, DisCoF uses a flexible approach to decoupling robots such that they can transition from optimistic to pessimistic decoupling when necessary.

Optimistic Decoupling

In order to reduce communication overhead, a robot is only allowed to communicate with other robots when it can sense them. However, robots that cannot sense each other can communicate using the message relay protocol through other robots. A closure of the set of robots that can communicate (directly or via message relay) in order to coordinate is called an *outer closure* (OC). In an OC, there can be multiple predictable conflicts. A closure that contains agents with potential conflicts is the *inner closure* (IC) of the OC. Figure 2.13 shows an example of OC and IC.

In DisCoF, decoupling is optimistic initially, and gradually becomes more pessimistic when necessary. Given an OC with predicted conflicts, in *optimistic decoupling* DisCoF updates the individual plans of robots to proactively resolve these conflicts, while avoiding introducing new conflicts within a finite horizon (which is specified by a parameter in DisCoF). Therefore, this "optimistic" decoupling allows to have another recurring conflict with the robots once they progress over the horizon. This finite horizon, however, is key to efficiency since the resolution for conflicts in the far future is likely to waste computation efforts given the incomplete information (e.g., the positions of other robots in the environment). Note that the window size, i.e., sensing range, in DisCoF represents a horizon for detecting conflicts.

To ensure that robots are jointly making progress towards their goals, DisCoF uses the notion of *contribution value*. In order to resolve conflicts, plans are updated in a process known as conflict resolution. In this process, each robot is associated with a contribution value when using optimistic decoupling. If this process is successful, robots continue as fully decoupled. The contribution value is also used to determine cases when optimistic decoupling is insufficient. That is, when the resolution process would fail due to potential live-locks. When there are no potential live-locks, it is shown that optimistic decoupling is sufficient for robots to converge to their goals. Otherwise, robots within the OC use the following *pessimistic decoupling* process.

Pessimistic Decoupling

In DisCoF, when there are potential live-locks, robots within an OC transition to *pessimistic decoupling* by remaining within each other's communication range (whether direct or indirect). These robots are referred to as a *coupling group*. This coupling group moves as a group until all the members finally reach their goal location, decoupling each member at its own goal location. In this way, the coupling group is "pessimistically" decoupled. In order for the coupling group to move in a group, it executes a process known as push and pull[4]. This process allows it to merge with other groups and robots. Thus, the level of coupling gradually increases. In this way, DisCoF can naturally transition robots to be fully coupled when necessary.

In push and pull, robots move to goals one at a time according to the priorities

[4]For this process, there are more details with figures later in this section.

of subproblems (first introduced in [28]). However, due to the incompleteness of information in distributed systems, the priorities will not be fully known. As a result, DisCoF employs the following process. At time step k, for each coupling group that has been formed, DisCoF will:

1. Maintain robots in the group within each other's communication range.
2. Move robots to goals one at a time based on a relaxed version of the priority ordering which is consistent to that in [28].
3. Add other robots or merge with other groups that introduce potential conflicts with robots in the current group as they move to their goals.

Unless there are potential conflicts, each coupling group progresses independently of other robots and coupling groups. These processes are described in Alg. 15:

Algorithm 15: $\textsc{PessimisticDecoupling}(G := (V, E, \omega, \mathcal{S}, \mathcal{G}), k, \mathcal{P})$

```
Input: a coupling group $\omega$, the current time step $k \in \mathbb{N}$, the environment
       $G := (V, E, \omega, \mathcal{S}, \mathcal{G})$ and a set of initial plans $\mathcal{P}$ for $\omega$
Output: a set of conflict free plans $\mathcal{P}$
1  $r \leftarrow \perp$                                          ▷ Initialize the leader robot $r$
2  while $\exists i \in \omega\ s.t.\ \mathcal{S}_k[i] \neq \mathcal{G}[i]$ do
3      $\langle \psi, \phi \rangle \leftarrow \textsc{SenseConflict}(\mathcal{P}, \omega, \mathcal{S}, k, \mathcal{W})$
4      if $\psi = \emptyset$ then
5          $k \leftarrow k + 1$                                  ▷ Increase the time step by 1
6          $G' \leftarrow (V, E, \omega, S, \mathcal{G})$
7          $\langle \mathcal{S}, \mathcal{P}, \mathcal{W} \rangle \leftarrow \textsc{ProceedOneStep}(G', \mathcal{P}, k)$
8      else
9          $\omega \leftarrow \omega \cup \psi$                  ▷ Merge conflict robots with $\omega$
10         $\langle f, D \rangle \leftarrow \textsc{AssignAgentsToSubP}(G, \omega, \mathcal{S}, \mathcal{G})$
11         $H \leftarrow \textsc{ComputePriority}(G, \omega, f, D, \mathcal{S}, \mathcal{G})$
12         $r \leftarrow \perp$
13     if $r = \perp \vee \mathcal{S}_k[r] = \mathcal{G}[r]$ then
14         $r \leftarrow \textsc{RemoveFromQueue}(H)$
15     $G' \leftarrow (V, E, \omega, \mathcal{S}, \mathcal{G})$
16     $\mathcal{P}' \leftarrow \textsc{PushAndPull}(G', r)$
17     $\mathcal{P} \leftarrow \mathcal{P}[0 : k] + \mathcal{P}'[:]$    ▷ Update a set of plans for $\omega$
```

Alg. 15 continues until all members in a coupling group ω reach their final goals. The termination condition is checked in line 2. As long as there exists a robot that has not reached its goal, the algorithm will continue with push and pull. In Alg. 15, r represents the leader of the group ω. We remark that there can be cases in which a robot that has already reached its goal may block the path of the leader r. In this case, push and pull will swap or rotate (similar to the operators in [28]) robots that have not reached their goals with this blocking robot in order to progress. Push and pull also ensures that this blocking robot moves back to its goal afterwards.

We remark that the graph G in Alg. 15 is a common graph space for the coupling group ω. Recall from Def. 21, Def. 22 and Assumption 3 that each robot $i \in \omega$ can have the product automaton $\mathcal{A}_i = \mathcal{T} \times \mathcal{B}_i = (S_{\mathcal{A}_i}, s_0^{\mathcal{A}_i}, \mathcal{P}(\Pi), \rightarrow_{\mathcal{A}_i}, F_{\mathcal{A}_i})$. In order to cooperatively find the path, we define that the common graph space G for the coupling group ω is as following:

- $V := \bigcap_{i \in \omega} \textsc{ProjNodes}(\mathcal{A}_i, \mathcal{T})$
- $E := \bigcap_{i \in \omega} \textsc{ProjEdges}(\mathcal{A}_i, V)$

Here, ProjNodes represents a set of observed nodes of Q in $\mathcal{T}$ for a specific state $s \in S_{\mathcal{A}_i}$ for each robot i and ProjEdges represents a set of observed edges in $V \times V$. The resulting graph $G = (V, E)$ is the maximally common local environment graph for all the robots which will be used for local planning while still the robots are guaranteed to satisfy their individual requirements.

The combination of optimistic and pessimistic decoupling in DisCoF guarantees completeness.[5]

[5] DisCoF is complete for the class of cooperative pathfinding problems in which there are two or more unoccupied vertices in each connected component. Later in this section, we show the proof of

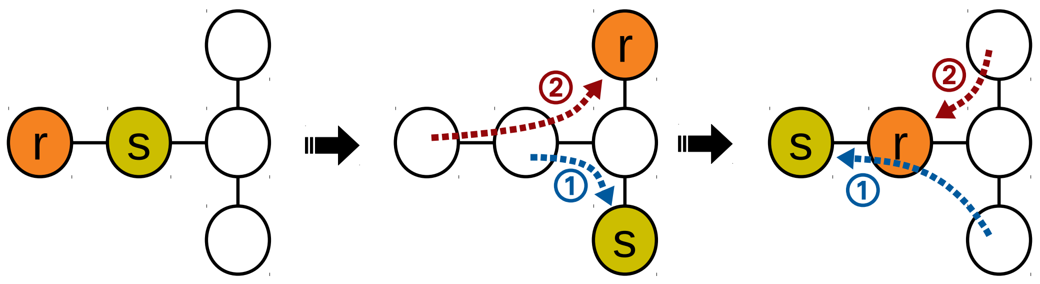
It assumes that an initial plan for each robot is given. This initial plan is computed only for each robot. At this time, all the robots are fully decoupled. This individually computed plan, however, does not consider other robots' plans. Thus, while the robots are progressing through the plan, they may encounter possible collisions. If a predictable conflict is sensed by a group of robots that are near (more precisely within the communication or sensing range), then they try to resolve the conflict by coupling together.

Example 15. *Fig. 2.13 shows that robot r_3 and r_4 have a predictable conflict at the next time step. Robot r_2 is also a neighbor, but it is not directly related with the conflict. When robot r_3 and r_4 are re-planning, r_2 can be also involved if two of them cannot have a proper new plan due to r_2. Robot r_1 is not sensible by r_2, r_3 and r_4 due to its limited sensing range.*

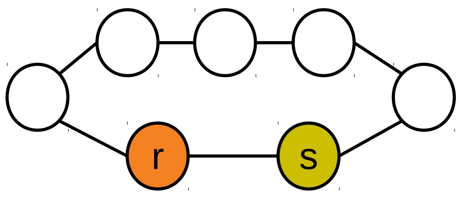
We call this stage of resolving procedure CONVERGENCE stage. In this stage, related robots or near robots are coupled together in order to re-plan together. Once they can have a new plan to avoid this collision, they are decoupled. However, due to the huge number of robots involved in the collision or its complicated environment, it may be impossible to resolve its conflict in a given time. In this case, it regards this conflict as a livelock.

In order to resolve this livelock situation, the group of robots follows a pre-defined rule based operations. We call this PUSHANDPULL. PUSHANDPULL is a sequence of rule based conflict resolution process among a group of conflicted robots. It, first, elects a leader of the group. Then, all the other members in the group respect the leader's moves to its goal. In this cooperative moves, the leader seems to 'push' the members blocking its way, but this is the members' moves prior to the leader's moves

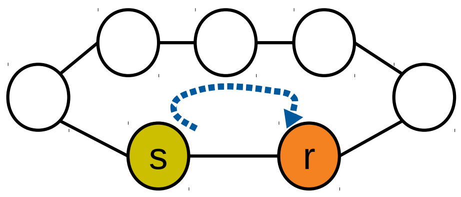
the PUSHANDPULL part which is an extension of results in [28].



(a) Swap operation



(b) Unswappable case



(c) Simple rotate operation

Figure 2.14: (a) shows a sequence of steps how the agent r and the agent s exchange their position through the swap operation. (b) is the case when two agents r and s cannot exchange their position through the swap operation. (c) shows that a simple rotate operation can exchange their position.

in order to make the way clean. Due to the limited sensing and communication range, the other members should follow the leader once it finishes moving. This also seems for the leader to 'pull' the other members, but again this is the members' moves after the leader's moves. In some case, this push operation is not feasible due to the environment condition, such as dead-end or lack of unoccupied further vertices. Then, the leader swaps the vertices with the member that blocks its way (see Fig. 2.14(a)). This raises more numbers of cooperative actions among the members. In some more special situation, this swap operation is also infeasible (see Fig. 2.14(b)). In that case, rotate operation is executed among the members (see Fig. 2.14(c) and Fig. 2.15).

Example 16. *Fig. 2.16 shows an overall picture how the* PushAndPull *operation*

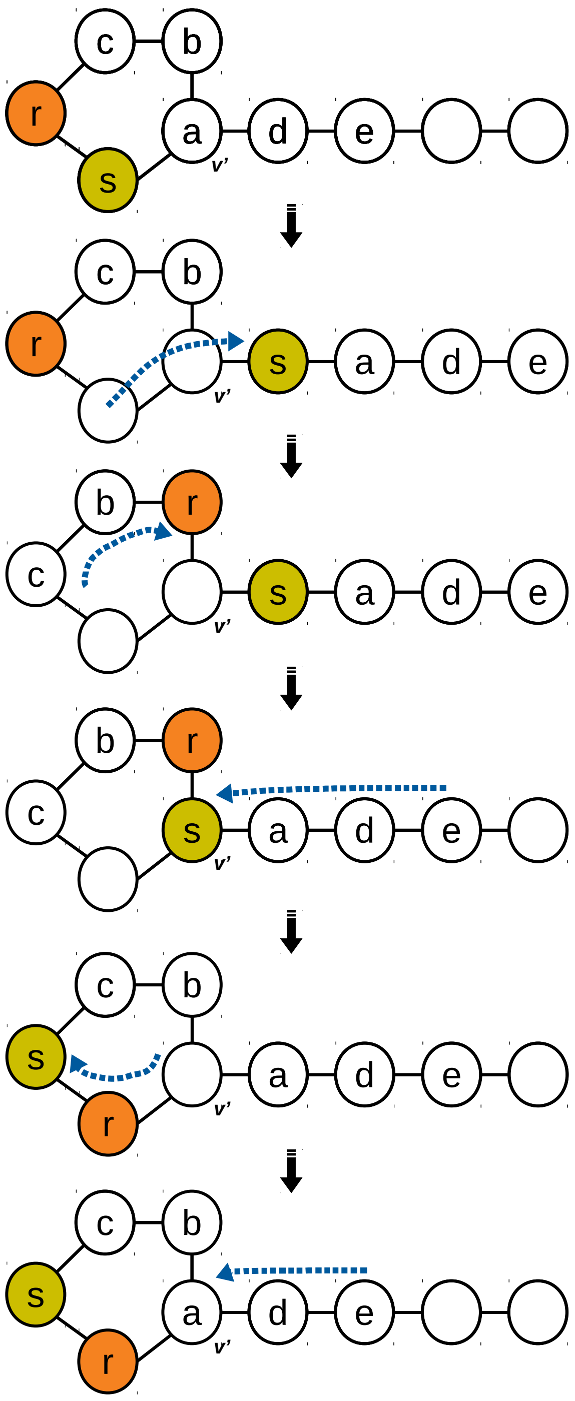


Figure 2.15: Even in the case when the cycle is fully occupied, it can exchange the position of the agent r and the agent s through rotate and swap operations if this cycle has at least one vertex v' having degree ≥ 3 and the component including the cycle and other part of the graph which is connected by v' has two or more unoccupied vertices.

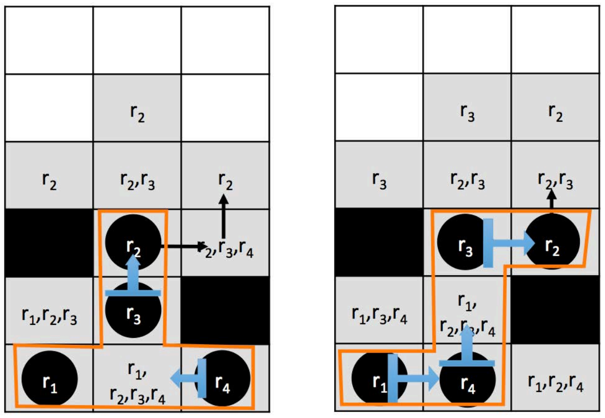


Figure 2.16: This shows how Pull operation works. Robot r_2 is a planning robot, so it moves to its goal. Robot r_3, r_4 and r_1 are being pulled. In the left figure, robot r_2 will move to its one cell right, r_3 will be pulled to its one cell up and r_4 will move to its one cell left. The right figure shows the robots' movements at the one step later.

resolves a livelock. In this scenario, robot r_2 is moving to its goal and robot r_3, r_4 and r_1 are being pulled.

This livelock resolution process makes the involved robots being coupled until all they reach their goal. We call this being partially coupled. This is because robots that are not related in this livelock still remain decoupled. In this way, this approach provides an interesting property to move fully coupled planning to partially (or fully depending on its problem instance) coupled planning.

In the next section, we extend the above approach. First, we relax the synchronous time assumption. Thus, for Fig. 2.13, the extended approach regards that the first group of robots r_2, r_3, r_4 and the second group of robot r_1 act independently and asynchronously. Second, we introduce a decoupling strategy. In the previous approach, members of a coupling group can be decoupled only if they reached their goals. On the other hand, in the extended approach, each member checks whether it can be decoupled while executing PUSHANDPULL.

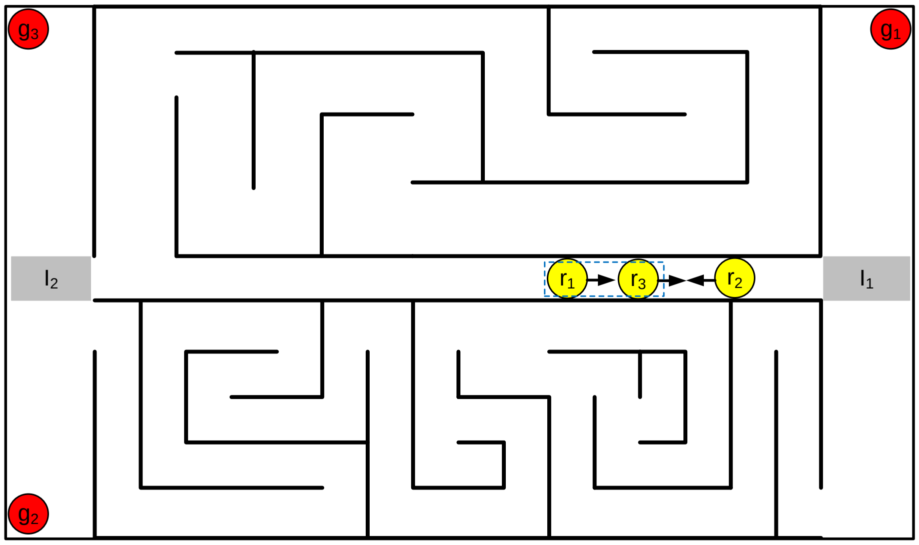


Figure 2.17: Yellow circles are robots and red circles are their goal locations. Blue dashed square represents a coupling group of robot r_1 and r_3. This group meets another robot r_2 moving in the opposite direction. Gray cells represent the intersections of corridors.

Example 17. *In Fig. 2.17, a group of robots r_1, r_2, r_3 moves to the intersection I_1 together. Then, robot r_1 is decoupled from robots r_2 and r_3, heading to g_1. Robots r_2 and r_3 are still remained in the coupling group until they reach the intersection I_2.*

Theorem 6. PushAndPull *is complete for the class of Multi-agent Pathfinding problems with at least two empty vertices.*

Proof (Sketch): In this proof, we focus on pull operation. The completeness of push and rotate (including swap) is already proved by [86]. The pull operation is an additional post process after each push & rotate operation. Hence, it does not influence the other operations. Besides, this operation sequence such as push & rotate then pull continues until the leader of a group reaches its goal. Then, the next leader follows the same operation sequences. In this way, eventually all the members in the group reach their goal. That is, the conflict is resolved. □

2.6.2 Asynchronous DisCoF

In this section, we discuss the extensions to DisCoF that are made in the new approach named DisCoF$^+$. First, we relax the assumption that robots synchronize at every time step (or plan step). Note that even though robots in different OCs cannot communicate in DisCoF, it is assumed that robots act in synchronized time steps. That is, robots are given a fixed amount of time to finish planning and execute a single action at every time step. The relaxation of this synchronization is necessary for implementation in a real distributed system because we cannot always assume the existence of a global clock and a fixed amount of time for each time step (e.g., the time required for planning for each robot may be arbitrarily different).

We remark that each robot can still access the entire map. We can assume that this information is static such that it is initially given and does not change.[6] However, each robot cannot recognize where other robots are if they are out of (indirect) communication and sensing range. This information is dynamic such that it changes arbitrarily.

Furthermore, we introduce a new decoupling strategy such that robots are also allowed to decouple after they form a coupling group (i.e., executing push and pull); thus, transitioning back to optimistic decoupling from pessimistic decoupling. This strategy makes DisCoF$^+$ more computationally efficient while achieving higher quality plans that require fewer steps.[7]

[6] Our replanning framework can be extended to partially known environments with unknown static obstacles.

[7] How efficient this strategy is depends on the problem instance. Robots in a denser environment may need frequent coupling and decoupling, thus increasing the computation overhead. This is discussed in Sec. 2.6.3 through simulation experiments.

Asynchronous Time Steps

Unlike DisCoF, DisCoF$^+$ allows robots in different OCs to proceed independently and asynchronously. However, robots within the same OC are assumed to still have synchronized plan steps. This is a reasonable assumption because these robots communicate to coordinate with each other. As a result of this assumption, robots who finish their current plan step must wait until all others in the OC also finish theirs. Afterwards, all members of the group start the next plan step at the same time in order to avoid unnecessary conflicts. We remark that since we assume homogeneous robots, the waiting time at each time step is not significant.[8]

We now explain Alg. 16. First, all variables including a set of current locations $\mathcal{S}$ and a set of current local window $\mathcal{W}$ are initialized and updated until line 6. Then, while progressing its own plan $\mathcal{P}$, it senses a conflict at line 8. If a conflict is not detected, then it progresses the next step at line 11. If a conflict is detected, then it resolves the conflict and updates the current plan $\mathcal{P}$ with the new plan $\mathcal{P}'$ from line 15 to line 25. If a conflict is detected such that the IC ψ is not empty, robot i tries to resolve the conflict after checking if it is already involved in any conflicts at line 15. If ω is not empty at line 15, it means that from the previous iterations, ω has been already assigned. Then, at the current iteration, another conflict is detected. That is, a coupling group meets another coupling group while resolving its conflict. Then, it merges the ω with an current OC ϕ and begins PushAndPull in order to resolve it through pessimistic decoupling process.[9] If ω is empty, then it means that it hasn't

[8] Heterogeneous robots may have different speed, sensing & communication range, size and etc. Considering these issues and resolving them are beyond the scope of this thesis topic.

[9] We remark that our description of PushAndPull in Alg. 16 is simplified to show the overall process. Once PushAndPull returns a new plan $\mathcal{P}'$ in Alg. 16, it contains the individual plans for the coupling group ω to move from their locations at the time step k to their goals.

```
Algorithm 16: DisCoF⁺($G := (V, E, \omega, \mathcal{S}, \mathcal{G}), \mathcal{P}, \gamma$)
Input: an asynchronous time step $k$ for a robot $i \in \mathcal{R}$, given environment
       $G := (V, E, \omega, \mathcal{I}[i], \mathcal{G}[i])$, its initial location $\mathcal{I}[i]$, final destination $\mathcal{G}[i]$
       and initial plan $\mathcal{P}[i]$ from $\mathcal{I}[i]$ to $\mathcal{G}[i]$
Output: a set of updated plans $\mathcal{P}$ for the coupling group $\omega$
Variables: (a local window $\mathcal{W}$, a group $\omega$ and a contribution value $\gamma \in \mathbb{N}_{\geq 0}$)
1  $\langle \psi, \phi, \omega, \mathcal{S}[:], \mathcal{W}, \gamma[:], k \rangle \leftarrow \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, 0, 0 \rangle$
2  $S[i] \leftarrow \mathcal{I}[i]$                                  ▷ Update the current location to $\mathcal{I}[i]$
3  $\mathcal{G}[: i-1] \cup \mathcal{G}[i+1 :] \leftarrow \emptyset$          ▷ Initialize goals for others
4  $\mathcal{P}[: i-1] \cup \mathcal{P}[i+1 :] \leftarrow \emptyset$          ▷ Initialize plans for others
5  $G' \leftarrow (V, E, \emptyset, \mathcal{S}, \mathcal{G})$
6  $\langle \mathcal{S}, \mathcal{W} \rangle \leftarrow$ ProceedOneStep($G', \mathcal{P}, i, k$)
7  while True do
8      $\langle \psi, \phi \rangle \leftarrow$ SenseConflict($\mathcal{P}, i, \mathcal{S}, k, \mathcal{W}$)
9      if $\psi = \emptyset$ then
10         $k \leftarrow k+1$; $G' \leftarrow (V, E, \omega, \mathcal{S}, \mathcal{G})$     ▷ Increase the time step $k$ by 1
11         $\langle \mathcal{S}, \mathcal{W} \rangle \leftarrow$ ProceedOneStep($G', \mathcal{P}, i, k$)
12         $G' \leftarrow (V, E, \omega, \mathcal{S}, \mathcal{G})$                  ▷ Update $G'$ with new $\mathcal{S}$
13         $\langle \gamma, \omega, \mathcal{P} \rangle \leftarrow$ RecomputeCont($G', \mathcal{P}, i, k, \gamma$)
14     else
15         if $\omega \neq \emptyset$ then                          ▷ It meets another group
16             $\omega \leftarrow \omega \cup \phi$; $G' \leftarrow (V, E, \omega, \mathcal{S}, \mathcal{G})$      ▷ Merge $\omega$ with OC $\phi$
17             $\mathcal{P}' \leftarrow$ PushAndPull($G', i, \gamma$)
18         else
19             $\omega \leftarrow \psi$; $G' \leftarrow (V, E, \omega, \mathcal{S}, \mathcal{G})$            ▷ Set $\omega$ to IC $\psi$
20             $\mathcal{P}' \leftarrow$ Convergence($G', i, k, \phi, \mathcal{P}, \mathcal{W}, \gamma$)
21             if $|\mathcal{P}'| = 0$ then
22                 $\omega \leftarrow \phi$; $G' \leftarrow (V, E, \omega, \mathcal{S}, \mathcal{G})$        ▷ Set $\omega$ to OC $\phi$
23                 $\mathcal{P}' \leftarrow$ PushAndPull($G', i, \gamma$)
24         if $\mathcal{P}' = \emptyset$ then return False
25         $\mathcal{P}[\omega] \leftarrow \mathcal{P}_{1,k}[\omega] + \mathcal{P}'[\omega]$
26     return True
```

involved any conflict yet. That is, robot $i \in \mathcal{R}$ was executing its plan independently. Then, it forms a local coupling ω. It first tries to decouple optimistically through Convergence. If it cannot find a plan $\mathcal{P}'$, then it decouples pessimistically through PushAndPull. After finding a plan $\mathcal{P}'$, it continues to the next iteration to sense if there are new conflicts. In this way, the above process continues until it reaches its goal.[10]

We need to explain some codes and procedures in details. First, in order to simplify each procedure, at line 5, 10, 12, 16, 19 and 22, we use G' as a tuple of V, E, ω, $\mathcal{S}$ and $\mathcal{G}$. Here, V and E are from the workspace $G = (V, E)$, ω is a set of robots which represents a coupling group, $\mathcal{S}$ is a set of current locations, and $\mathcal{G}$ is a set of goal locations. Second, given a tuple G', a set of plans $\mathcal{P}$, a robot $i \in \mathcal{R}$, and i's local time step k, PROCEEDONESTEP returns a set of current locations $\mathcal{S}$ and a current location window $\mathcal{W}$. We remark that PROCEEDONESTEP does not increase the time step variable k. If k is not increased before calling PROCEEDONESTEP, like line 6, then it does not update the current locations $\mathcal{S}$ with the set of plan $\mathcal{P}$. However, it is required to be called because the current local window $\mathcal{W}$ should be updated before sensing a predictable conflict at line 8. Third, given a set of plans $\mathcal{P}$, a robot $i \in \mathcal{R}$, a current set of locations $\mathcal{S}$, i's local time step k and the current local window $\mathcal{W}$, SENSECONFLICT returns a tuple of an IC ψ and an OC ϕ. If no conflict is detected, the IC ψ is empty. Regardless of the existence of conflicts, SENSECONFLICT also returns an OC ϕ. This may require to communicate with other agents (we will explain in the next subsection). Fourth, the contribution value γ is used in RECOMPUTECONT, CONVERGENCE and CONVERGENCE. In the next subsection, we will explain the details about how to update the contribution value γ and how the contribution value γ affects the set of plans $\mathcal{P}$.

Correctness: For Alg. 16, we need to show two conditions. First, if a given problem instance is valid (solvable), robot $i \in \mathcal{R}$ eventually reaches its goal location.

[10] Due to lack of global communication and coordination, our algorithm (running on each robot) would not be able to determine whether all other robots have reached their goals, thus we cannot compute a termination condition. In our simulation, we stop the programs (on all robots) when they have reached their goals.

If there is no conflict from the initial location $\mathcal{I}[i]$ to its goal location $\mathcal{G}[i]$, it can progress through its plan while sensing conflicts at line 8 and proceeding one step at line 11 until it reaches its goal. Whenever there is a conflict, it always computes a valid plan. At line 15, robot i checks if it is already involved in a conflict (with ω). If $\omega \neq \emptyset$ (i.e., it is already involved in a conflict), it merges the OC (i.e., ϕ in Alg. 16) with ω, and then call PushAndPull for i. In line 21, if $\mathcal{P}'$ is not empty, it means that Convergence returns a new plan $\mathcal{P}'$. If $\mathcal{P}'$ is empty, then robot i calls PushAndPull. In both cases, the returned plan $\mathcal{P}'$ is either from Convergence or PushAndPull. We have shown that Convergence or PushAndPull always returns a valid plan in [27] if a valid solution exists.

Second, if a given problem instance is invalid (unsolvable), Alg. 16 returns False. In order to resolve a conflict, Alg. 16 first calls Convergence at line 20 which is for optimistic decoupling in DisCoF. Then, if it cannot compute its new plan, it calls PushAndPull at line 23 which is for pessimistic decoupling. In [27], we showed DisCoF guarantees the completeness, and DisCoF uses these two conflict resolution processes in order to resolve its conflict. Hence, if a solution exists, the combination of these two processes returns a solution. However, if a solution does not exists, it returns False. At line 24, it can check whether it returns a solution or not. If not, it returns false.

We remark that ProceedOneStep in line 11 always results in the robot proceeding one step forward in its plan. If robot i has already reached its final goal (while there are robots that still need to reach their goals), proceeding one step in this case simply adds a step for robot i to stay.

Communication and Leader Selection

There are two major cases in which robots communicate with each other in DisCoF$^+$. The first case is to detect predictable conflicts. For detecting conflicts, given a robot $i \in \mathcal{R}$, SenseConflict requires the following steps:

1. Check nearby environment (i.e., $\mathcal{W}$) through a sensor for other robots (e.g., a laser sensor).
2. Compute the OC ϕ of robot i.
3. Communicate with robots in ϕ to obtain their plans, then check if predictable conflicts exist among them.

In the above process, the first step does not require any communication between robots; it only depends on sensors. Since robots know the environment (i.e., G), they can easily detect when there are moving robots nearby using range sensors.

The second step requires the use of a message relay protocol to compute the OC ϕ. This is because OC ϕ includes robots which cannot directly communicate with the robot i which originally tried to determine its OC ϕ. Even though it computed an OC ϕ in its previous time step, the OC ϕ can be changed whenever SenseConflict is called. This is because each robot in the OC has its own asynchronous time if it is not involved in any conflicts and it can update its OC without considering other members. In this way, if one of the members in the OC moves out of its neighbors' sensing range before communicating with its neighbors, other members cannot update their own OC. In addition, if each member in an OC is involved in a conflict, all the members have a synchronized time step until reaching their local goals. In this case, computing a new OC is still required because each member of the OC can meet another group and each member can update their own OC propagating new information to each other. Hence, whenever each agent calls SENSECONFLICT, it should communicate with others so that it can update its OC.

In the third step, once robot i obtains all the plans of the robots in ϕ, it can check these plans against its own plan for predictable conflicts (from its current time step to the next β steps [27]). In this case, after electing a leader of the IC ψ, the leader computes the new plan for the IC ψ and communicates the new plan back to the others in the IC.[11] In order to compute a new plan, the leader tries CONVERGENCE. If CONVERGENCE returns a valid set of plans $\mathcal{P}'$, then the leader can pass $\mathcal{P}'$ to others. If not, the leader begins PUSHANDPULL. However, in PUSHANDPULL a new leader is selected which is based on the priorities of subproblems. Then, the new leader will send the new plan $\mathcal{P}'$ (which is computed from PUSHANDPULL) back to others in the OC ϕ.

Example 18 (Sensing Conflicts)**.** *Consider the scenario in Fig. 2.13. In this scenario, assume that robot r_4 is robot i in the above procedure, so r_4 tries to sense a predictable conflict. r_4 first senses its nearby environment for other robots. In Fig. 2.13, the local window or the sensing range of r_4 (denoted by $\mathcal{W}$) is shown as the gray region marked with r_4, and r_4 will detect r_3. r_4 then computes the OC ϕ as $\{r_4, r_3, r_2\}$. Since r_2 is not r_4's local window, r_3 will relay the communication between r_2 and r_4. Once r_4 obtains both r_2 and r_3's plans, it will check their plans against its owns plan for predictable conflicts. In this scenario, r_4 will recognize a predictable conflict with r_3 which can be addressed using* CONVERGENCE. $\triangle$

The second case in which robots communicate is to synchronize planning and execution among robots in an OC. Note that robots in different OCs proceed independently and asynchronously. Since planning and plan execution are synchronized

[11] The simplest voting mechanism is to elect the robot with the smallest ID in the group.

within an OC, it is guaranteed that no collision can occur among robots in the OC. In ProceedOneStep, each robot in the OC executes a single plan step, communicates this to the rest of the robots in the OC (through broadcasting to the local network), and then it halts. Only after all robots in the OC have completed a plan step are they free to execute another, thus achieving synchronization.[12] However, when robots move out of the communication range, they do not synchronize their plan steps anymore.

Flexible Decoupling

Flexible decoupling is achieved with the help of contribution values. Contribution values are assigned in DisCoF to each robot in the Convergence process (in optimistic decoupling) in which the robots must compute an update to the current plan to avoid potential conflicts. Contribution values are introduced in DisCoF to ensure that robots are jointly making progress to their goals. In DisCoF, when the Convergence process fails, robots are in a coupling group, running on the plan computed by PushAndPull until they reach their goals. In DisCoF$^+$, however, robots that are executing PushAndPull can also decouple by checking whether certain conditions involving the contribution values hold.

Next, we discuss the new decoupling strategy in DisCoF$^+$ which is illustrated in the following example. Suppose that a conflict is predicted between two robots. Then, an IC ψ (initially including only the two robots) is formed and there is an associated OC ϕ for ψ. When the leader of ψ makes a new plan in the Convergence process, if the leader cannot find a new plan that avoids the conflict with the current set of conflicting robots ψ, then the set of conflicting robots gradually expends (until becoming ϕ). When a new plan is found, DisCoF$^+$ associates each robot with a *contribution value* γ

[12] We assume that in a fixed amount time, each robot can complete its own movement and within the communication range there is no problem to communicate with each other.

which captures the individual contribution of the robot to the summation of shortest distances from all robots' current locations to their goal locations.

For the remaining part of this section, we will use the cost relation $\mathcal{C} : V \times V \rightarrow \mathbb{N}$. For example, $\mathcal{C}(v_1, v_2)$ is the distance of the shortest path from node v_1 to node v_2.

At the very beginning of a problem instance, the contribution value γ is initialized to be 0 for all robots. Given a predictable conflict at time step k, a set of conflicting robots ϕ, the set of current locations S_k for ϕ and the set of goal locations $\mathcal{G}$, the new plan $\mathcal{Q}$ (where $|\mathcal{Q}| < \beta \in \mathbb{N}$)[13] must avoid collisions and satisfy the following:

$$\sum_{i \in \phi} \mathcal{C}(\mathcal{S}_k[i], \mathcal{G}[i]) + \gamma_k^-[i] > \sum_{i \in \phi} \mathcal{C}(\mathcal{S}_k[i](\mathcal{Q}[i]), \mathcal{G}[i]) \tag{2.5}$$

where $\gamma_k^-[i]$ is the contribution value that is associated with robot i at the time step k and $\mathcal{S}_k[i](\mathcal{Q}[i])$ is a local goal for each $i \in \phi$, i.e., the position reached by each robot i after executing plan $\mathcal{Q}[i]$.

We remark that while k in Eq. (2.5) is a constant in DisCoF, in DisCoF$^+$, k represents the synchronized current time step for the group of robots within ϕ which may differ between OCs.

An interesting point of Eq. (2.5) is that the new plan $\mathcal{Q}$ may not satisfy Eq. (2.5) during the execution of $\mathcal{Q}$, as long as Eq. (2.5) is satisfied after $\mathcal{Q}$ has completed. After executing the new local plan $\mathcal{Q}$, each agent reaches its local goal. In this way, they avoid the predicted conflict. Then, each robot $i \in \phi$ can decouple, following its individual plan from the local goal $\mathcal{S}_k[i](Q[i])$ to its goal $\mathcal{G}[i]$. Given a predicted conflict at the current time step and a computed $\mathcal{Q}$, the contribution value γ while

[13] We assume that the length of the plan $\mathcal{Q}$ is bigger than β. If the length of some agent i's plan $\mathcal{Q}[i]$ has shorter than β, then the last state $S_k[i](\mathcal{Q}[i])$ should be appended at the end of $\mathcal{Q}[i]$ until $|\mathcal{Q}[i]| \geq \beta$.

executing the actions in $\mathcal{Q}$ is updated for robot i in ϕ as follows:

$$\gamma_{k+\delta}[i] = \mathcal{C}(\mathcal{S}_k[i](\mathcal{Q}[i]), \mathcal{G}[i]) - \mathcal{C}(\mathcal{S}_{k+\delta}[i], \mathcal{G}[i]) \tag{2.6}$$

where $0 \leq \delta \leq |\mathcal{Q}|$ and $\mathcal{S}_{k+\delta}[i] = \mathcal{S}_k[i](\mathcal{Q}_{1,1+\delta}[i])$. We remark that δ is a relative time step after the robots have formed an OC. For all robots in a group, δ is the same. This update continues until the robot become involved in other conflicts or the value becomes 0.

In DisCoF [27], the contribution value γ is only used for the Convergence process, and robots do not update their contribution values when a coupling group is formed and robots start PushAndPull. This can lead to inefficient behaviors, e.g., when the leader's goal location is located opposite to where the others' goals are located.

This situation is illustrated in the following example.

Example 19 (Narrow Corridor)**.** *Figure 2.17 shows an example of robot r_2 in a narrow corridor meeting with a coupling group $\{r_1, r_3\}$ (executing* PushAndPull*) moving in the opposite direction. The coupling group $\{r_1, r_3\}$ started in the middle corridor, and then r_1 became the leader. While r_1 pushes r_3 to clear away of its path to its goal location g_1, it meets r_2. In this case, they will be merged together. Suppose that r_1 is chosen to be the leader of the new group $\{r_1, r_2, r_3\}$. Until r_1 reaches its goal location g_1, r_2 and r_3 will be pushed to the end of the middle corridor and then they will be pulled after the intersection i_1.* △

In DisCoF, the only way to reduce the size of a coupling group is to have the current leader reach its goal. Then, a new leader will be selected and the remaining robots will follow the new leader to its goal. This is clearly an inefficient solution. In DisCoF$^+$, we use the contribution values γ also in PushAndPull, such that robots can decouple even before the leader reaches its goal.

Next, we discuss how the contribution values can be used in the PushAndPull process. More specifically, we provide a decoupling condition for a coupling group to check which determines when the robots in the group can decouple while executing the PushAndPull process. Suppose that there is a coupling group ω. After ω computes a new plan $\mathcal{P}'$ (in PushAndPull), each robot in ω will progress using the plan. During this execution, robots continue recomputing their contribution values γ as in Eq. (2.6). At any step, if the following condition holds, then the group can be decoupled:

$$\sum_{i \in \omega} \mathcal{C}(S_k[i], \mathcal{G}[i]) + \gamma_k^-[i] > \sum_{i \in \omega} \mathcal{C}(S_{k+\delta}[i], \mathcal{G}[i]) \tag{2.7}$$

where k is the time step when PushAndPull starts planning and $k+\delta$ is the current time step such that $0 < \delta \in \mathbb{N}$. $\gamma_k^-[i]$ is the contribution value that robot $i \in \omega$ had before the PushAndPull returned its plan.

Intuitively, Eq. (2.7) is the condition when the summation of the length of the shortest-path from robots' current locations to their goal locations is less than the summation of the length of the shortest-path from their original coupling locations to their goal locations plus their contribution values just before forming the coupling group.

In Alg. 16, Eq. (2.7) is checked inside of RecomputeCont at line 13. Given a set of current locations $\mathcal{S}$, a set of goal locations $\mathcal{G}$, and contribution values γ, if the condition holds, then RecomputeCont returns an updated plan $\mathcal{P}$ (i.e., the shortest-path plan from $\mathcal{S}[i]$ to $\mathcal{G}[i]$) with an empty coupling group ω. Then, the coupling group ω becomes decoupled and each robot follows their individual plan. Otherwise, RecomputeCont returns the current plan $\mathcal{P}$ without changing the coupling group ω. Then, the coupling group ω follows the current plan $\mathcal{P}$ which was computed from PushAndPull.

Example 20 (Decoupling)**.** *In Fig. 2.17, when the coupling group $\{r_1, r_3\}$ is merged with r_2, then conflict locations for $\{r_1, r_2, r_3\}$ and the contribution values (i.e., γ) are saved. For a simple illustration, assume that $\gamma = 0$. Then, whenever the merged group of robots $\{r_1, r_2, r_3\}$ proceed one time step in their plan (which is returned by* PushAndPull*), they also check the decoupling condition in Eq.(2.7) in* RecomputeCont*. However, until the leader r_1 reaches its goal location g_1, they cannot be decoupled. This is because the summation of the distance between robots' locations to their goal locations keeps increasing. When r_1 reaches its goal location g_1, r_1 is removed from the group. Assume that r_2 is elected as a new leader of the group. Then, r_3 will be pulled until they reach the conflict location where they met previously. (See the place where they are placed in the Fig. 2.17) After passing the conflict location, r_3 and r_2 can be decoupled since Eq.* (2.7) *holds. Consequently, from the intersection I_2, r_2 and r_3 can move independently to their goal locations.* △

When a coupling group is decoupled and it immediately predicts a conflict in the next iteration, it uses the conflict resolution process through Convergence, just as when fully decoupled robots have predicted conflicts. Even though we discussed the correctness of DisCoF$^+$ (Alg. 16), we also need to show that this new decoupling strategy is not subject to live-locks (i.e., robots are always making joint progress to the goals).

Theorem 7. *The decoupling condition in Eq.*(2.7) *ensures that robots in the group gradually progress to their final goals.*

Proof. From Eq. (2.5) and Eq. (2.6), we know that each robot in the group gradually moves towards its final goal. Here, we show that Eq. (2.7) does not prevent any group member from reaching its goal. Given that we use the contribution value γ_{k-} when a coupling group is formed, in order to satisfy Eq. (2.5) when decoupling, either robots can all execute their original plans or Convergence must return a new plan which progresses robots to their local goals. First, their original plans definitely make progress. Second, consider the case when it takes the new plan from Convergence. After progressing through the new plan, all the robots in the group will reach their local goals. Then, the summation of the distance from their current locations (which are their local goals) to their final goals is smaller than the summation of the distance from their locations (where they predicted the conflicts) to their final goals plus their contribution values γ before forming the coupling group. Hence, we can conclude that robots would be making joint progress to their goals. Hence, the decoupling condition Eq. (2.7) does not prevent the group members from progressing to their final goals. □

2.6.3 Results

In this section, we present some experimental results. First, we will show a simulation result on a physics based simulator. Second, we will provide results from numerical experiments on artificial benchmarks.

Simulation in Webots

The simulation shown in Fig. 2.18 was created using Webots 7.3.0 and the included iRobot Create models. A grid environment was modeled which contained 30 iRobots and 40 obstacles placed at random locations. This instance is solvable, i.e., each robot can reach its goal position. Each iRobot was running with a controller which implemented DisCoF+. However, one exception was made: rather than being completely distributed and simulating ad hoc networks and localization, the robots communicated with a central supervisor which provided this information as well as synchronization for robots involved in a conflict, i.e., in the same OC. Robots in different outer closures acted completely asynchronously, but robots in the same outer closure were synchronized if a conflict was detected between any of the member robots.

The target computer for the simulation was a MacBook Pro running Mac OS X 10.10.2 with a 2.3GHz i7 and 16GB of RAM. The simulation was run two times: once with decoupling enabled and once with decoupling disabled. Decoupling enabled yielded a total simulation duration of 3 minutes and 23 seconds. Out of all robots, the maximum number of steps required to reach their destination was 40. When decoupling was disabled, it yielded a total simulation duration of 5 minutes and 1 second. Out of all robots, the maximum number of steps required to reach their destination was 54.

These results are interesting in two aspects: the total running time and the number of maximum steps. First, in terms of the total running time, enabling decoupling performs significantly better than without decoupling. The simulation took only 67% of the time that the other did. Second, in terms of the maximum steps, enabling decoupling took only 74% of the steps than without decoupling. The reason for this discrepancy is that with decoupling enabled there are more stay actions in which a robot's action is to stay where it is. Since robots are asynchronous except for when they are in a conflict, this means robots will take less time to complete a plan with stay actions compared to one that doesn't. It is expected that environments which remains more complex plans will benefit from this fact even more.

We provide the demo video for this simulation. In addition, you may refer to the videos at the following URL: `https://www.assembla.com/spaces/discof/wiki/DisCoF_Plus`.

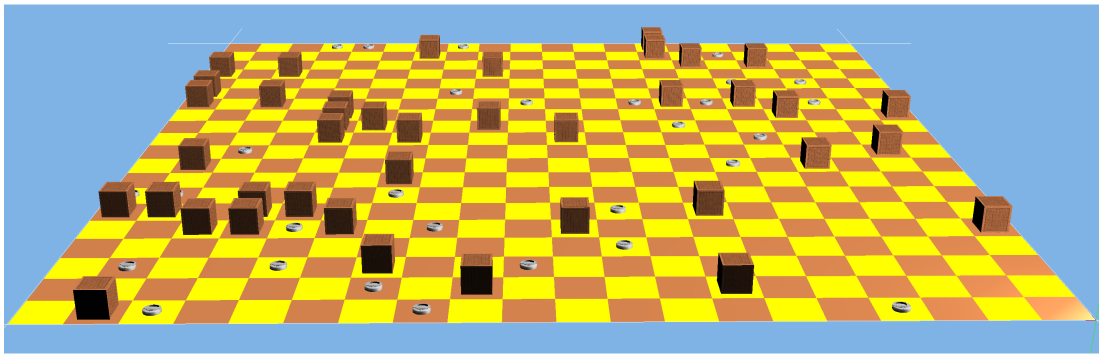


Figure 2.18: A simulation environment in Webots modeling a 20 × 20 grid world with a 10% wooden boxes as obstacles. In this environment, there are 30 iRobot Create finding their path to their goal positions.

	DisCoF						DisCoF$^+$ (DisCoF$^+$/DisCoF)					
	COMP. TIME		STEPS		APPROX. RUN TIME		COMP. TIME		STEPS		APPROX. RUN TIME	
OBSTACLES	AVG	STD	AVG	STD	AVG	STD	AVG	STD	AVG	STD	AVG	STD
5%	10.064	8.405	352.35	356.207	1771.815	1788.861	10.733 (1.0086)	22.068 (0.931)	63.95 (0.4266)	80.632 (0.356)	330.483 (0.43)	423.885 (0.3555)
10%	13.19	10.372	521.1	521.24	2618.69	2615.82	14.37 (1.061)	36.52 (1.538)	73.51 (0.344)	108.93 (0.346)	381.92 (0.348)	579.065 (0.348)
15%	17.6318	13.296	653.67	580.01	3285.982	2911.463	23.92 (1.217)	49.768 (1.3)	99.18 (0.294)	157.356 (0.312)	519.82 (0.3)	831.07 (0.314)
20%	26.39	14.009	954.46	620.08	4798.691	3111.208	52.391 (1.942)	75.8 (2.3989)	175.9192 (0.2427)	218.859 (0.3132)	931.987 (0.2535)	1161.61 (0.3242)

Table 2.7: Simulation Experiments: Comp. Time represents the total computation time in sec, Steps represents concurrent time steps for entire robots' plan, and Approx. Run Time represents approximate running time in sec. AVG stands for average and STD for standard deviation. The ratio inside the parenthesis is DisCoF$^+$/DisCoF.

Simulation Experiments on benchmarks

In order to evaluate the improvement of DisCoF$^+$ over DisCoF, we execute a number of numerical experiments. For these experiments, we used a 3.2GHz i7 and 8GB of RAM in Cygwin environment which runs on Windows 8.1. Our prototype implementation is written in Python 2.7.2.

Since we only want to get the total number of concurrent steps and the computation time for these experiments, instead of using the Webots simulator, we used a simple discrete time simulator which does not simulate the physics of the robots. In addition, we have not computed the overhead of any communication between the robots. Hence, we are comparing the total number of steps and the computation times between DisCoF and DisCoF$^+$.

As a result of this implementation, an approximate running time is calculated for each problem instance by summing the computation time and the movement time, where the movement time is the amount of time required to execute all steps assuming 5 seconds per step.

In order to perform the experimental analysis, instead of scaling up the number of robots, we increase the density of the environment. That is, we increase obstacle rates in the environment. The experiment was performed on a 20×20 grid environment with 30 robots. Obstacles were randomly generated according to their rate which is defined as the percentage of the grid environment that is considered to be an obstacle. Table 2.7 shows the results for 100 instances of DisCoF and DisCoF$^+$ as the obstacle rate was varied from 5% to 20%.

In all cases, DisCoF$^+$ needed 24% to 42% less steps than DisCoF's result and DisCoF$^+$ took 25% to 43% less than DisCoF's approximate run time.

The time ratio in Table 2.7 indicates that if the environment is less populated, then decoupling makes better quality plans in terms of the total number of concurrent steps and the total computation time of plans.

Despite the fact that DisCoF$^+$ consistently outperforms DisCoF in the approximate run time, it is important to comment on the computation time. When the environment is dense, it takes more computation time. This is because in dense environments groups that decouple may have to re-couple with a higher frequency. That is, when it recouples, a group should make a new plan which requires extra computation time.

2.7 Conclusions and Future Directions

In this chapter, we covered mission planning at various levels. We introduced LTL Revision Problems for a simple transition system and a weighted transition system. Then, we moved to a problem for a specification having quantitative preferences. In addition, we introduced a multi-agent LTL planning problem.

For the LTL Revision problem, we have following possible directions.

1. **Online LTL Revision for dynamic environments** The current approach does not consider that a given system (environment) can be changed. This assumption should be relaxed if we consider verifying a specification while running the framework in the real world. This is because the environment can be updated unexpectedly. In this case, a concept of local and global specification or local and global atomic proposition should be introduced.

2. **Resolving uncertainties through learning theory** For this chapter, we assume that the environment is fully known and we do not consider any noise while sensing it. In order to resolve uncertainty issues, instead of transition system $\mathcal{T}$, Markov Decision Process (MDPs) can be utilized [22, 87–93]. Along this direction, [94–100] focus on learning for control synthesis for a given LTL specification. Mostly the above works are for learning the environment or for learning the given mission. However, if we extend this to the users' perspective, we also can help them to reduce their mistakes by suggesting satisfiable, alternative missions.

3. **Behavioral Study for updating the specifications** Consider a problem for a specification which has quantitative preferences. The decision process of choosing a minimal subset of these atomic propositions to be removed may

cause setting up different preference levels later. In this way, updating the preference of each atomic proposition can avoid having the same un-realizable specification. While analyzing which atomic propositions' preferences are revised in a particular system, we can learn about the designer and then can estimate his/her expected behavior.

For the Multi-agent LTL Planning problem, we have the following possible directions.

1. **Partially observable environment** Consider that each agent can sense only nearby environment and the initially given setting can be updated later. In this case, when a conflict is detected between some agents, they have to share each other's map data. This uncertainty may cause to start the resolution process again when another agent is joined in the coupling group with new information about the environment.

2. **Heterogeneous agents** Relaxing the assumption that every agent has the same size may cause changing the order of reaching their goal position. In addition, this heterogeneous characteristic may interrupt some agent moving into particular cells. For this issue, we also discuss in some future directions in the next chapter.

3. **Dynamic obstacles** Unexpected moving obstacles, such as humans, or un-communicable vehicles driven by humans are common in the real world. Particularly, autonomous vehicles in a city cannot avoid this situation. Resolving a conflict with these objects can be a challenging and important future direction.

Chapter 3

MOTION PLANNING

3.1 Motivation

When moving from a grid or graph world abstraction to the real world, we need to consider the motion planning problem for mobile robots (and not only the path planning problem on graphs). As we discussed in Sec. 1.3.3, or motion planning, there are offline planning algorithms like Lattice Planner [61, 62], RTR+TTS [65] and RRT*(G3) [66, 67]. In general, offline algorithms can provide an optimal solution, but they are slow and the planning is limited to relatively shorter distances. On the other hand, online planners like Hybrid-A* [35] and RRT* [88] can avoid this planning horizon issue. However, if we apply these planners for challenging, unstructured environments, it is not easy to get the solution within a short time. In addition, if we add dynamics to the motion model, it is much more difficult for these planners to get a solution. Hence, we want to enable long horizon planning, to get a motion plan considering the vehicle dynamics within a given time in the rural, unstructured road networks. In other words, the contribution of this chapter is the derivation of motion planning algorithms for higher order dynamical systems which need to compute a path in confined environments within wait times imposed by the patience of the human users (typically, vehicle passengers).

In this section, once we cover the background knowledge, the problem is stated and a multi-resolution online lattice planner is proposed as a solution. Then, potential future extensions and conclusions are followed.

3.2 Preliminaries

We assume a bidirectional dynamic vehicle motion model for driving:

$$\frac{d}{dt}\begin{pmatrix} x \\ y \\ \theta \\ v \\ \delta \end{pmatrix} = \begin{pmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \frac{v}{L}\tan(\delta) \\ a \times D \\ \zeta \end{pmatrix} = f(q, u, D). \tag{3.1}$$

Here, $q = [x, y, \theta, v, \zeta]^T \in Q$ is the state vector (configuration) where $(x, y) \in \mathbb{R}^2$, $\theta \in \mathbb{R}$, $v \in \mathbb{R}$ and $\zeta \in \mathbb{R}$ are the vehicle's planar coordinates, orientation, velocity, and steering angle, respectively. The control input is $u = [a, \zeta]^T \in U \subseteq \mathbb{R} \times \mathbb{R}$ where $a \in \mathbb{R}$ and $\zeta \in \mathbb{R}$ are the vehicle's acceleration and steering angle speed, respectively. The desired driving direction is $D \in \{-1, 1\}$ such that $D = 1$ is for forward driving ($v \geq 0$) and $D = -1$ is for reverse driving ($v \leq 0$). The parameter $L \in \mathbb{R}_{>0}$ refers to the wheelbase.

Given two configurations $q_i, q_j \in q$, a ***motion plan*** p between them is a sequence of tuples: (q_i, a_i, q_{i+1}), $(q_{i+1}, a_{i+1}, q_{i+2})$, …, (q_{j-1}, a_{j-1}, q_j) where q_{i+1}, …, q_{j-1} are configurations and each $a_i = (a, \zeta, \tau)$ is a control input for the vehicle's acceleration, steering angle speed and duration.

We assume that the environment is unstructured. Hence, there is no traffic rule such as restricted driving directions and priorities at intersections. However, we assume that the environment is known and decomposed into convex cells (road segments) – see Fig. 3.2(a) for an example. Hence, the road segments and their adjacency relation form a sparse strongly connected graph $\mathfrak{R} = (R, S)$ where R is the set of road segments and $S \subseteq R^2$ is the set of connections of the road segments. Given a sequence of adjacent road segments r, we denote by $\mathbf{b}(r)$ the corresponding line connecting the middle points of the shared boundaries of the road segments (piecewise-linear path).

$h : q \rightarrow \mathbb{R}^3$ is an observation (location) function. Similarly, $h^c(r)$ denotes the sequence of road segment centroids $c \in \mathbb{R}^2$. We also define a labeling function $\mathcal{L} : \mathbb{R}^2 \rightarrow R$ that maps each point of the workspace to a road segment (we assume that the boundary between two road segments belongs only to one road segment). With a slight abuse of notation, we extend the labeling function $\mathcal{L}$ to also map configurations of the vehicle to road segments through the vehicle's planar coordinates in the obvious way. We also let $\mathcal{L}^h = \mathcal{L} \circ h^c$ and $h^r = h^c \circ \mathcal{L}^h$.[1]

For an edge $s = (r_i, r_j) \in S$ between two different road segments r_i and r_j, we define a weight function $W : S \rightarrow \mathbb{R}_{\geq 0}$ as $W(s) = \|h^c(r_i) - h^c(r_j)\|_2$ where $\| \cdot \|_2$ is the Euclidean norm. A path p_h from r_s to r_g is a sequence of connections of road segments $s_0, s_1, ..., s_n$ where $s_i = (r_i, r_{i+1})$ with $r_i, r_{i+1} \in R$ and $i \in \mathbb{N}_{\geq 0}$. Here, $s_0 = (r_s, r_0)$ and $s_n = (r_{n-1}, r_g)$. The total cost of a path p_h is $\Sigma_{i=0}^{|p_h|} W(p_h(i))$. The shortest path p_h^* from r_s to r_g can be computed by $argmin_{p_h \in Path_{r_s,r_g}} \Sigma_{i=0}^{|p_h|} W(p_h(i))$ where $Path_{r_s,r_g}$ is the entire path set from r_s to r_g. For convenience, we also define a function W^* that returns the optimal path cost between any two road segments $W^*(r_s, r_g) = \Sigma_{i=0}^{|p_h^*|} W(p_h^*(i))$.

We denote by $\mathcal{W}^*$ the free workspace of the robot for a specific road network. Moreover, given a sequence r of consecutive road segments, we denote by $\mathcal{W}(r) \subseteq \mathbb{R}_{\geq 0}$ the (static) free workspace that corresponds to the road segments in r. That is, $\mathcal{W}(r) = \{(x, y) \mid \exists r_i \in r, \mathcal{L}(x, y) = r_i\}$. In order to sense the free workspace at a current location (x, y) with a sensing range $\sigma \in \mathbb{R}_{\geq 0}$, we define a function $O : \mathbb{R}^3 \rightarrow 2^{\mathbb{R}^2}$ as $O(x, y, \sigma) = \mathcal{W}^* \cap B_\sigma(x, y)$ where $B_\sigma(x, y)$ denotes a Euclidean ball of radius σ

[1]We remark that the h^c and $\mathcal{L}^h$ are two mapping functions. h^c is to get the centroid c when a road segment r is given and $\mathcal{L}^h$ is to get the road segment r when a centroid c is given.

centered at (x, y). With a slight abuse of notation, we let $O^h(x, y, \sigma) = O \circ \mathcal{L} \circ h^c$ denote the set of centroids of the road segments observable from location (x, y).

Now, we can define the ***vehicle model*** as a tuple $\mathcal{V} = (f, q, u, h, O^h)$ and the ***workspace model*** as a tuple $\mathcal{T} = (R, S, \mathcal{L}, W, \mathcal{W})$.

3.3 Online Motion Planning Problem

This thesis considers the problem of path and motion planning in unstructured rural road networks. Such road networks typically exhibit steep turns and narrow passages as in Fig. 1.1. An autonomous vehicle would have to resolve such motion planning issues in almost real time[2]. Due to the complexity of the problem, it is expected that first a path would be computed over the graph representing the road network and, then, the complex motion planning problem has to be solved. The former problem (path planning) can be solved with different graph search algorithms such as Dijkstra's shortest path algorithm [31]. This thesis focuses on the latter problem.

Problem 8. *Given a vehicle model $\mathcal{V} = (f, q, u, h, O^h)$, a workspace model $\mathcal{T} = (R, S, \mathcal{L}, W, \mathcal{W})$, an initial configuration q_s and a final configuration q_g, compute the motion plan p in a on-the-fly manner.*

Overview of Solution: We will use a hierarchical approach as shown in Fig. 3.1. First, we decompose the workspace into convex cells [30, 31]. Then, we will construct lattice sets and introduce a multi-resolution motion planner. Lastly, we will explain how we utilized an exhaustive search planner ([101], [30]) in the case when the above planner cannot find a feasible motion. Combining these two planners, we provide a resolution-complete algorithm.

[2]In this work, we do not quantify what "almost real time" means since even humans in certain driving scenarios would have to pause before deciding the best driving maneuver.

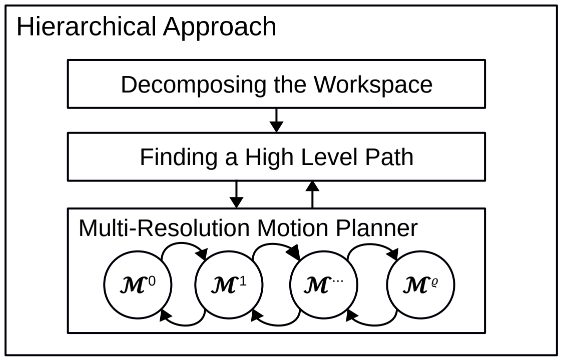


Figure 3.1: Hierarchical Approach

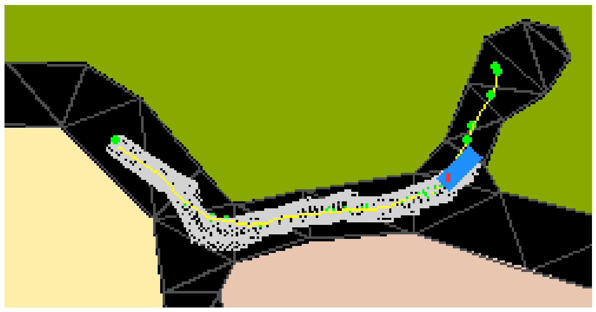
3.3.1 Multi-resolution Motion Planner

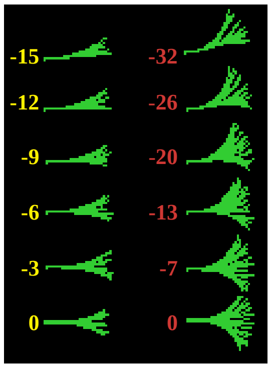
For the high level path finding, we search the road network from the road segment containing the source configuration to the road segment containing the goal configuration. The shortest distance path can be easily computed through a single destination shortest path algorithm. In the following, given a source state q_s and the destination state q_g, the ***high level path*** is denoted by $p^*_h = (r_0, r_1)$, (r_1, r_2), …, (r_{n-1}, r_n) where $r_0 = \mathcal{L}^h(q_s)$ and $r_n = \mathcal{L}^h(q_g)$.

When a high level path p^*_h is given, we can connect the centroids of the corresponding road segments in order to form a continuous path to be tracked. Typically, such a path may need to be smoothened (we use a Gaussian filter). We call this smoothed line *spine*. A *spine* is a sequence of states, $l_0, l_1, \ldots, l_n$, where l_0 is from q_s, l_n is from q_g and $l_1, \ldots, l_{n-1}$ are smoothed states on the path. Each state l_i consists of $(x, y, \theta, \tilde{D})$, where (x, y) is the 2 dimensional coordinates, θ is the orientation[3] of the state and $\tilde{D}$ is the guided direction to drive the path.

Once a path is smoothened, then it can be easier for a vehicle to follow the path. In order to closely follow the *spine*, we have to control the vehicles' motion. Given a

[3]It can be computed through an arctangent with its next (or previous) state's 2 dimensional coordinates.


(a) Vehicle's trajectories



(b) Two lattice sets

Figure 3.2: Vehicle trajectories and lattice sets

configuration q_s, we can sample a configuration l_g along or near the *spine* at a pre-defined farther distance while considering the orientations of q_s and l_g. We call this l_g a *local goal*. Given q_s and l_g, the ***low level plan*** p consists of a sequence of tuples: ρ_0, ρ_1, . . ., ρ_n. Each tuple ρ_i consists of (q_i, u_i, q_{i+1}) where q_i and q_{i+1} are configurations with $q_0 = q_s$ and $\mathcal{L}(q_{n+1}) = \mathcal{L}(l_g)$, and $u_i = (a_i, \zeta_i, \tau_i)$ is an input (a_i, ζ_i) for a duration τ_i. Figure 3.2(a) shows an example of the ***hierarchical approach***. Each triangle is a decomposed cell of the workspace and a vehicle trajectory is generated by the low level plan p by following the *spine* of p_h^*.

State Lattice

Although our lattice planner has some similarities with others [61, 62], our objective is a resolution-complete motion planner. In addition, our motion planner has different aspects from others [61, 62]. First, we do not construct a search space (a tree based graph) while searching for a local goal. Second, we do not regularly distribute the search (with respect to the 2D workspace) in order to expand our search area while finding the local goal. In this way, we can save construction time while generating the motion in the restricted environment. This has an advantage particularly when we compute the motion through a densely covered lattice set.

Next, we will design motion primitives. Since constructing a plan while considering differential constraints requires more computation time, we compute the motion primitives offline. Note that for the configuration states in $\tilde{Q}_s$ the position and orientation are set to zero, i.e., $(x, y, \theta) = (0, 0, 0)$. With a constrained control input set $\tilde{U}_i$ and a pre-defined duration κ_i, the motion primitives can be computed. We denote by $\mathcal{A}(\tilde{Q}_s, \tilde{U}_i, \kappa_i)$ the reachable set from $\tilde{Q}_s$ under possible inputs $\tilde{U}_i$ after duration κ_i. For $\tilde{q}_s \in \tilde{Q}_s, \forall q_l \in \mathcal{A}(\{\tilde{q}_s\}, \tilde{U}_i, \kappa_i), \exists u_l \in \tilde{U}_i$ such that $\tilde{q}_s$ reaches q_l after duration κ_i. We denote this control input u_l by $\mathbf{u}(\tilde{q}_s, q_l)$.

Now, we can construct a lattice set. First, given $\tilde{q}_s$ and a resolution level i, we exhaustively construct trees where the root nodes are $\tilde{q}_s$ and its children nodes are $\mathcal{A}(\tilde{q}_s, \tilde{U}_i, \kappa_i)$ (see Fig. 3.2(b)). Second, we create 2 dimensional KD-Trees based on $\mathcal{A}^{\mathcal{J}}$ with the $(\tilde{x}, \tilde{y})$ cell size. Note that we can also construct this lattice set for the past time with $\mathcal{A}(\tilde{q}_s, \tilde{U}_i, -\kappa_i)$ until the depth reaches $\mathcal{J}$.

Now, consider l_g as a local goal located at a pre-defined distance from q'. Using the KD-Tree in $\mathcal{M}_{q'}$, we can find a fixed number of neighboring states $\mathcal{N}$ to l_g based on their distance from l_g.[4] If for all the states in $\mathcal{N}$ we cannot find any η which is collision free, then we return failure.

In order to provide multi-resolutional lattice sets, we create two types of lattice sets: *consistent* and *shifting*. For example, in Fig. 3.3, $\mathcal{M}^0, \ldots, \mathcal{M}^n$ are the *consistent* lattice sets and $\mathcal{M}^{0,1}, \mathcal{M}^{1,0}, \ldots, \mathcal{M}^{n-1,n}, \mathcal{M}^{n,n-1}$ are the *shifting* lattice sets.

[4][102] uses Hash maps to search in a local grid. This search is for one key, value pair, taking a constant access time. However, we consider that the key may not exist and that we may have to find not only the one key, value pair, but also β number of keys, value pairs which are nearby and ordered by distance. In this case, using KD-Tree is much more efficient.

$$\begin{tikzcd}
\mathcal{M}^0 \arrow[r, bend left, "\mathcal{M}^{0,1}"] & \mathcal{M}^1 \arrow[l, bend left, "\mathcal{M}^{1,0}"] \arrow[r, bend left, "\mathcal{M}^{1,2}"] & \mathcal{M}^2 \arrow[l, bend left, "\mathcal{M}^{2,1}"] \arrow[r, bend left, "\mathcal{M}^{2,3}"] & \cdots \arrow[l, bend left, "\mathcal{M}^{3,2}"] \arrow[r, bend left, "\mathcal{M}^{n-1,n}"] & \mathcal{M}^n \arrow[l, bend left, "\mathcal{M}^{n,n-1}"]
\end{tikzcd}$$

Figure 3.3: *Consistent* and *Shifting* lattice sets for multi-resolution

ϱ	a velocity level index for a lattice set $\mathcal{M}$
$\mathcal{M}$	a lattice set
p_h^*	a high level shortest path
p, p', p''	a low level plan
$\mathcal{A}$	a set of connections of road segments to be avoided
$q_s, q_g, q_c, \ldots$	configurations consisting of $(x, y, \theta, v, \delta)$
κ	a pre-defined duration for the given control inputs
α	$\alpha \in \mathbb{N}_{\geq 2}$ for the number of local goals
$p_h^*.Spine$	a smoothed path of p_h^*
lgs	a sequence of local goals chosen from $p_h^*.Spine$
$\gamma, \gamma_g, \gamma_{lg}$	pre-defined distances $\gamma_i := (\gamma_i^{xy}, \gamma_i^{\theta})$ for Euclidean and heading
$\sigma_{lp}, \sigma_{sensor}, \sigma$	$\sigma_{lp}, \sigma_{sensor}, \sigma \in \mathbb{R}_{>0}$: for pre-defined distances
η, η_i, η_j	local coordinates consisting of $(x, y, \theta, v, \delta)$
$q_i^{\eta_i}$	a transformed global coordinates from q_i with η_i
$q_i^{\eta_i \cdot \eta_j}$	a transformed global coordinates from $q_i^{\eta_i}$ with η_j
$\mathcal{M}^{\varrho}$	$\mathcal{M}^{\varrho} \in \mathcal{M}$ where it has a velocity level ϱ
$\mathcal{M}_{q_i}^{\varrho}[\eta_i]$	a low level plan from q_i to $q_i^{\eta_i}$
β	$\beta \in \mathbb{N}_{>0}$: for sampling the neighbors in local coordinates

Table 3.1: List of Symbols

Multi-resolutional Online Motion Planner

Now, we are ready to present the planning algorithm, which consists of three parts: HIERACHICALPLANNER, LOCALPLANNER and GETLOCALPLAN. A summary list of symbols and functions are presented in Table 3.1 and 3.2.

Algorithm 17 is the high level planner. This gets the initial coordinates q_s and the goal coordinates q_g. With q_s and q_g, it computes the high level plan p_h^* which is a sequence of road segments to reach q_g. Then, until the current configuration q_c reaches near q_g, it continues computing low level plans p' and updating q_c in a receding horizon manner [33]. Once it reaches near q_g, it can return the entire low level plan p.

```
Algorithm 17: HierarchicalPlanner($q_s, q_g, \varrho, \mathcal{V}, \mathcal{M}, \mathcal{T}$)
  Input: an Initial conf. $q_s$, a goal conf. $q_g$, a velocity level index $\varrho$, a vehicle $\mathcal{V}$,
         a lattice set $\mathcal{M}$ and a workspace $\mathcal{T}$
  Output: a low level plan $p$ or Failure
1 $p[:] \leftarrow \emptyset$; $p'[:] \leftarrow \emptyset$; $\mathcal{A} \leftarrow \emptyset$; $q_c \leftarrow q_s$
2 $p_h^* \leftarrow$ GetHighLevelPath($q_c, q_g, \mathcal{A}, \mathcal{T}$)
3 while $\neg(d_3(q_c, q_g) \leq \gamma)$ do
4     $\langle p', \varrho \rangle \leftarrow$ LocalPlanner($p_h^*, p', q_c, \varrho, \mathcal{V}, \mathcal{M}, \mathcal{T}$)
5     if $|p'| = 0$ then
6         $\mathcal{A} \leftarrow \mathcal{A} \cup \{(\mathcal{L}^h(q_c), p_h^*.Next(\mathcal{L}^h(q_c)))\}$
7         $p_h^* \leftarrow$ GetHighLevelPath($q_c, q_g, \mathcal{A}, \mathcal{T}$)
8         if $p_h^* = \emptyset$ then return Failure
9     else
10        $q_c \leftarrow p'.Next(q_c, \kappa)$
11        $p \leftarrow p + p'.Copy^{<}(q_c)$; $p'.Trim^{<}(q_c)$
12 return $p$
```

For Alg. 17 to 19, instead of computing a plan to connect two configurations exactly (TPBV), we frequently check if a configuration is close to another configuration within some pre-defined distance γ. We evaluate whether two configurations' Euclidean distance and their heading difference are within the desired tolerance. Hence, $\gamma := (\gamma^{xy}, \gamma^{\theta})$ consists of two parts: $\gamma^{xy} \in \mathbb{R}_{\geq 0}$ and $\gamma^{\theta} \in [0, \pi]$. Consider two configurations q_i, q_j and a desired tolerance $\gamma = (\gamma^{xy}, \gamma^{\theta})$. We denote the Euclidean distance between the planar coordinates of the configurations q_i and q_j by $d_{xy}(q_i, q_j)$ and the heading distance by $d_{\theta}(q_i, q_j)$. For brevity, we write $d_3(q_i, q_j) \leq \gamma$ as a shorthand for the check $d_{xy}(q_i, q_j) \leq \gamma^{xy} \wedge d_{\theta}(q_i, q_j) \leq \gamma^{\theta}$.

Algorithm 18 is the low level planner. This algorithm takes as input q_c and p_h^* and returns an updated low level plan p' based on the pre-computed lookup table $\mathcal{M}$ with the current velocity level index ϱ.

Algorithm 19 finds the plan for multiple local goals lgs (which is similar with [35]) from q_c through $\mathcal{M}^{\varrho}$.

```
Algorithm 18: LocalPlanner($p_h^*, p', q_c, \varrho, \mathcal{V}, \mathcal{M}, \mathcal{T}$)
   Input: a high path $p_h^*$, a given low plan $p'$, a current coord. $q_c$, a velocity
          level index $\varrho$, a vehicle $\mathcal{V}$, a lattice set $\mathcal{M}$ and a workspace $\mathcal{T}$
   Output: a tuple of a low plan $p'$ and a velocity level index $\varrho$
 1 $\mathcal{V}_{\mathcal{M}} \leftarrow \langle \mathcal{V}, \mathcal{M}, \varrho \rangle$
 2 $q_n \leftarrow p'.Next(p'.Last(), -\kappa)$; $p'.Trim^{\geq}(q_n)$
 3 while $\sum_{(q_i,u_i,q_j)\in p'} d_{xy}(q_i, q_j) < \sigma_{lp}$ do
 4     $lgs \leftarrow$ SetupLocalGoals($p_h^*.Spine, q_n, \varrho, \kappa, \alpha$)
 5     $p'' \leftarrow$ GetLocalPlan($lgs, q_n, \mathcal{V}_{\mathcal{M}}, \mathcal{T}$)
 6     if $|p''| = 0$ then
 7         if $\varrho = 0$ then return $\langle \emptyset, \varrho \rangle$
 8         $\mathcal{V}_{\mathcal{M}} \leftarrow \langle \mathcal{V}, \mathcal{M}, \varrho.Decrement \rangle$
 9     else
10         $p' \leftarrow p' + p''$; $q_l \leftarrow p'.Last()$
11         if $\varrho = 0$ then
12             $\mathcal{V}_{\mathcal{M}} \leftarrow \langle \mathcal{V}, \mathcal{M}, \varrho.Increment \rangle$; $q_n \leftarrow q_l$
13         else
14             if $d_3(q_l, lgs[-1]) \leq \gamma$ then break
15             $q_n \leftarrow p'.Next(q_l, -\kappa)$; $p'.Trim^{\geq}(q_n)$
16 if $\neg$Stiff($p_h^*.Spine, \tilde{p}'.Last(), \varrho, \sigma_{lp}$) then $\varrho.Increment$
17 return $\langle p', \varrho \rangle$
```

Bi-directional Search for narrow and stiff curves

When Alg. 19 returns an empty plan p'', the ϱ is decremented. Then, Alg. 19 is called again. This process makes the velocity slower and the steering range wider while switching from $\mathcal{M}^{\varrho}$ to $\mathcal{M}^{\varrho-1}$. Finally, when it becomes $\mathcal{M}^0$, plans from $\tilde{q}_s \in \tilde{Q}_s$ to some $q' \in \tilde{Q}$ may include back-and-forth movements. This is for the cases when we cannot find any feasible motion plan until the velocity is decreased as slow as possible and the steering range is increased to the maximum.

Car Grid Search (e.g., [101], [30]) can resolve this issue. This planner utilizes 6 different actions: Left+, Right+, GoStraight+, Left-, Right-, and GoStraight-. The first three actions are for the forward driving and the next three actions are for the reverse driving direction. We take the similar approach. First, we pre-compute 6 motion primitives. Then, we construct the lattice set $\mathcal{M}^0$ with the computed primitive set.

Constructing Bi-directional lattice set

That is, we compute 6 actions:

- TurnLeft(+1), TurnRight(+1), GoStraight(+1)
- TurnLeft(−1), TurnRight(−1), GoStraight(−1)

We remark that above (+1) and (−1) are the driving direction D for the primitive function f. Each action continues for a pre-defined duration. In addition, before starting each action and after the action, velocity and steering are 0, 0, respectively.

We take $\tilde{q}_s = (0, 0, 0, 0, 0)$ as an initial state and we create a tree rooted at $\tilde{q}_s$. We run a Best First Search Algorithm exhaustively from $\tilde{q}_s$, connecting with the 6 configurations which are computed with the 6 actions. While expanding each configuration, the algorithm checks:

- if its Euclidean distance to $\tilde{q}_s$ exceeds a pre-defined radius.
- if it is in an already occupied cell in a pre-defined grid.

If one of the above conditions holds, the chosen configuration is discarded. Otherwise, we add this configuration to the cell and expand to its following 6 configurations. While expanding a configuration to its children, we also connect it to its parent node in the tree. We add each configuration in the occupied cell to $\tilde{Q}$. We note that if the cell of the grid is too fine, expanding some coordinates will take exceptionally long. However, within the pre-defined radius, the set of reachable cells from $\tilde{q}_s$ in the state space is finite.

```
Algorithm 19: GetLocalPlan($lgs, q_n, \mathcal{V}_\mathcal{M}, \mathcal{T}$)
  Input: local goals $lgs$, a current coord. $q_n$, a vehicle with a lattice set $\mathcal{V}_\mathcal{M}$
         and a workspace $\mathcal{T}$
  Output: a low level plan $p''$
1 $\left\langle (\overleftrightarrow{f}, q, u, h, O^h), \mathcal{M}, \varrho \right\rangle \leftarrow \mathcal{V}_\mathcal{M}$
2 $Q \leftarrow$ PriorityQueue()
3 $Found \leftarrow \bot$; $p''[:] \leftarrow \emptyset$
4 for $\eta \in$ SearchNeighbors($q_n, lgs[1], \mathcal{M}^\varrho, \beta$) do
5     $Q.Push(\langle$GetCost($q_n^\eta, lgs[1]$)$, [\langle q_n, \eta, 1\rangle]\rangle)$
6 while $\neg Found \wedge |Q| > 0$ do
7     $\left\langle cost_i, p_i^l \right\rangle \leftarrow Q.Pop()$
8     $\langle q_i, \eta_i, i\rangle \leftarrow p_i^l[-1]$
9     $p_i \leftarrow \mathcal{M}_{q_i}^\varrho[\eta_i]$
10    if IsCollided($p_i, O^h(q_i, \sigma_{sensor})$) then continue
11    if $i = |lgs|$ then
12        if $d_3(q_i^{\eta_i}, lgs[i]) \leq \gamma_{lg}$ then $Found \leftarrow \top$
13        continue
14    for $\eta_j \in$ SearchNeighbors($q_i^{\eta_i}, lgs[i+1], \mathcal{M}^\varrho, \beta$) do
15        $cost_j \leftarrow$ GetCost($q_i^{\eta_i \cdot \eta_j}, lgs[i+1]$)
16        $Q.Push(\left\langle cost_i + cost_j, p_i^l + [\langle q_i^{\eta_i}, \eta_j, i+1\rangle]\right\rangle)$
17 if $Found$ then
18    for $j = 1$ to $|p_i^l|$ do
19        $\langle q_i, \eta_i, i\rangle \leftarrow p_i^l[j]$
20        $p'' \leftarrow p'' + \mathcal{M}_{q_i}^\varrho[\eta_i]$
21 return $p''$
```

Querying a state

Like for other $\mathcal{M}^\varrho$, we build a KD-Tree with $\tilde{Q}$ in $\mathcal{M}^0$. Then, given a current coordinate q_c and its local goal l_g, l_g's neighbor states can be searched from $\mathcal{M}_{q_c}^0$ through the KD-Tree. We note that the resolution of the KD-Tree is equal to the pre-defined grid. Finally, we set the bound of l_g's coverage of $\mathcal{M}^0$ as follows: $\kappa \leq l_g$'s coverage of $\mathcal{M}^0$ $<$ a pre-defined radius.

- $p_h^*.Next(r_i)$: returns r_i's next road segment from p_h^*
- $p.Next(q_i, \kappa)$: returns q_i's next coordinates on p, coming after κ duration. If $\kappa < 0$, it returns q_i's previous coordinates on p, coming before κ duration.
- $p.Trim^{<}(q_n)$: trims all the elements coming before q_n on p
- $p.Trim^{\geq}(q_n)$: trims all the elements from q_n to the end on p
- $p.Copy^{<}(q_n)$: copies all the element coming before q_n on p
- $\varrho.Decrement$: decreases the index by 1
- $\varrho.Increment$: increases the index by 1
- $p.Last()$: returns the last coordinates on p
- $\textsc{PriorityQueue}()$: returns an emptied priority queue
- $Q.Pop()$: takes an element from Q having the minimum cost
- $Q.Push()$: adds an element to Q
- $\textsc{GetHighLevelPath}(q_c, q_g, \mathcal{A}, \mathcal{T})$: returns a high level shortest path from q_c to q_s, avoiding $\mathcal{A}$
- $\textsc{SetupLocalGoals}(p_h^*.Spine, q_n, \varrho, \kappa, \alpha)$: gets α local goals from q_n. Each one takes at most κ duration for traveling, considering the velocity level index ϱ
- $\textsc{Stiff}(p_h^*.Spine, p.Last(), \varrho, \sigma_{lp})$: returns $\top$ only if $p_h^*.Spine$'s curve from p's last coordinates is stiff for the next σ_{lp} distance, given the velocity level index ϱ
- $\textsc{SearchNeighbors}(q_n, lgs[i], \mathcal{M}^{\varrho}, \beta)$: returns β local coordinates which are near in $lgs[i]$ when they are transformed with the global coordinates q_n
- $\textsc{GetCost}(q_i, lgs[i])$: computes the cost between q_i and $lgs[i]$: $d_{xy}(q_i, lgs[i]) \times gain_1 + d_\theta(q_i, lgs[i]) \times gain_2$ where $gain_1, gain_2 \in \mathbb{R}_{\geq 0}$
- $\textsc{IsCollided}(p, O^h(q_i, \sigma_{sensor}))$: checks if the rigid bodies in q_i for the low level plan p exceed free workspace, considering the sensing range σ_{sensor}

Table 3.2: List of Functions

3.3.2 Theoretical Analysis

The running for constructing the *Consistent* and *Shifting* lattice sets is $O(|\tilde{Q}| \cdot |\tilde{Q}_s| \cdot |\mathcal{A}(\tilde{Q}_s,\, \tilde{U},\, \kappa)|)$ and $O(|v'|! \cdot \sum_{\tilde{q}_s \in \tilde{Q}_s} \prod_{j=1}^{\mathcal{J}} |\mathcal{A}(\{\tilde{q}_s\}, \tilde{U}, \kappa)|^j)$.

Algorithm 19 consists of SEARCHNEIGHBORS(), ISCOLLIDED() and the while loop. SEARCHNEIGHBORS() takes $O(2 \cdot \sqrt{N})$ where N is the number of nodes (coordinates) in the KD-Tree of each $\mathcal{M}^{\varrho}$. Checking 2D rectangular polygon takes constant time K. Finding road segments within the radius ς from q_c takes $O(2\sqrt{|R|})$. Then, ISCOLLIDED() takes $O(K \cdot |O^h(q_c, \varsigma)| + 2\sqrt{|R|})$. Removing all constants, Alg. 19 takes $O(\sqrt{N} + \sqrt{|R|})$.

Algorithm 18 consists of SETUPLOCALGOALS(), GETLOCALPLAN(), the while loop and STIFF(). SETUPLOCALGOALS() takes $O(\alpha(\log|p_h^*.\mathit{Spine}| + G))$ where G is a constant time for checking the distance between q_n and a way point. GETLOCALPLAN() takes $O(\sqrt{N}+\sqrt{|R|})$ and the while loop takes $O(\alpha(\alpha(\log|p_h^*.\mathit{Spine}|+G)+\sqrt{N}+\sqrt{|R|}))$. STIFF() takes $O(\log|p_h^*.\mathit{Spine}|+F)$ where F is a constant time for getting a cumulative heading difference in order to check the stiffness of the spine. Removing all constants, Alg. 18 takes $O(\log|p_h^*.\mathit{Spine}| + \sqrt{N} + \sqrt{|R|})$.

Algorithm 17 consists of GETHIGHPATH(), LOCALPLANNER() and the while loop. GETHIGHPATH() takes $O(|S| + |R| \cdot \log|R|)$ where $|S|$ is the number of connections of road segments and $|R|$ is the number of road segments. LOCALPLANNER() takes $O(\log|p_h^*.\mathit{Spine}| + \sqrt{N} + \sqrt{|R|})$. The while loop takes $O(|S| \cdot (\log|p_h^*.\mathit{Spine}| + \sqrt{N} + \sqrt{|R|}+|S|+|R|\cdot\log|R|))$. Then, Alg. 17 takes $O(|S|+|R|\cdot\log|R|+|S|\cdot(\log|p_h^*.\mathit{Spine}|+\sqrt{N} + \sqrt{|R|} + |S| + |R| \cdot \log|R|))$.

We will show the proofs on our approach in Appendix C (see Sec. C.1 and C.2).

Theorem 8 (Resolution Complete)**.** *If Prob. 8 has a motion plan p which can be represented in some resolution of the workspace, then Alg. 17 to 19 can find one* *with the decomposition of the workspace in multi-resolution and pre-computed & pre-constructed lattice sets. It indicates failure otherwise. In both cases, it finishes within a finite time.*

Theorem 9 (Correctness)**.** *Algorithm 17 to 19 solve Prob. 8.*

You may find the proposed framework including videos at the following URL: https://cpslab.assembla.com/spaces/multi-robot-planning/.

3.3.3 Numerical Experiments

We compared our deterministic planning approach (based on lattices) against a well-established probabilistic planning approach (RRT [30, 31]). We refer to the RRT planner as the baseline method. We implemented an online version of RRT and tried to make it close to ours in order to test it in the given settings.

We sampled the acceleration and steering value from their maximum range. For a given local goal l_g, we tried to build the tree for 5 times with cutoff at 500 iterations. The RRT used is adapted from [103], and changed for our dynamic model. Then, we used an informed RRT with 0.005 chance to choose l_g as some random coordinates.

Since l_g is given, we narrowed the sampling area from the current coordinates q_c to l_g with some additional boundary areas. Once the RRT failed to return a motion plan, then we switch to bi-directional[5] RRT. We here choose two different driving velocities: one for a forward and another for a reverse direction. We implemented a simple shooting method for steering.

In order to evaluate our approach, we created an unstructured road environment and we considered the three scenarios of Fig. 3.4. The first scenario (S.I) has moderate

[5]This lets RRT work as our $\mathcal{M}^0, \mathcal{M}^{0,1}$ and $\mathcal{M}^{1,0}$ do for multi-resolution. Hence, Bi-directional means here that a tree started from q_c can be grown with nodes having either forward driving direction or reverse driving direction, rather than growing two trees from q_c and l_g.

		Time [s] (mean $\pm$ std) For			Number (mean $\pm$ std) Of			Length	Succ.
		Total	$\neg$Stiff	Stiff	$\neg$Stiff	Stiff	Reversals	[m]	[%]
S.I	M	9.0 $\pm$ 1.6	0.3 $\pm$ 0.8	0.0 $\pm$ 0.0	15.0 $\pm$ 0.0	0.0 $\pm$ 0.0	0.0 $\pm$ 0.0	73.6 $\pm$ 0.0	100
	B	243.2 $\pm$ 84.9	12.2 $\pm$ 4.1	138.3 $\pm$ 75.7	15.9 $\pm$ 0.9	0.2 $\pm$ 0.5	0.0 $\pm$ 0.0	72.5 $\pm$ 0.7	30
S.II	M	18.4 $\pm$ 2.8	0.9 $\pm$ 2.1	4.4 $\pm$ 3.4	8.0 $\pm$ 0.0	2.0 $\pm$ 0.0	9.0 $\pm$ 0.0	63.6 $\pm$ 0.0	100
	B	259.7 $\pm$ 127.3	11.2 $\pm$ 4.3	131.0 $\pm$ 131.0	9.0 $\pm$ 0.5	1.0 $\pm$ 0.0	0.0 $\pm$ 0.0	41.2 $\pm$ 0.9	8
S.III	M	13.1 $\pm$ 0.5	0.6 $\pm$ 1.6	2.2 $\pm$ 0.2	13.0 $\pm$ 0.0	1.0 $\pm$ 0.0	1.0 $\pm$ 0.0	67.7 $\pm$ 0.0	100
	B	N/A	N/A	N/A	N/A	N/A	N/A	N/A	0

Table 3.3: Numerical Experiment result for three scenarios (S.I, S.II and S.III). M stands for Multi-resolutional Online Lattice Planner and B stands for Baseline Approach (RRT).

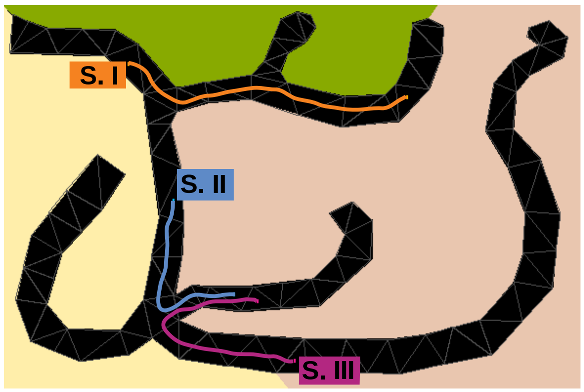


Figure 3.4: Three Scenarios: S.I, S.II and S.III

curves while the others (S.II and S.III) have stiff curves. We ran these scenarios 100 times for each approach. Table 3.3 presents the results. We remark that our approach for each run produced identical path lengths and number of velocity reversals (since it is deterministic). On the other hand, the baseline approach produced the resulting path on S.II very differently, returning many failures. We also remark that the vehicle size in this experiment is 2 M width and 4.8 M length. Even though RRT is known to be probabilistically complete, in restricted environments like in Fig. 3.4, it seems to have some difficulty to compute plans within time bounds tolerable to the human passengers of autonomous vehicles.

It is worth noting that we also tried RRT* [88]. However, the results were not promising due to the narrow and stiff curves and the approximate steering method that we utilized. A potential solution in improving the RRT* performance would be to utilize exact steering functions as in [66]. This will be a topic of future work.

We also tested our work against Hybrid-A*. We implemented multi-resolutional Hybrid-A*. Figure 3.5 shows their trajectories. Since Hybrid-A* [35] requires to manage already visited states, we created grid cells in 3D with the size of $(\tilde{x}, \tilde{y}, \tilde{\theta})$. Here we changed the $\tilde{\theta}$ from 3.5° to 30°, keeping the size of $\tilde{x}$ and $\tilde{y}$ same. It took 597.3 sec, 267.4 sec, 122.7 sec, respectively. On the other hand, our approach took 6.6 sec. As you can see from Fig. 3.5(a) - (c), while increasing $\tilde{\theta}$, the trajectory generated has bigger turns. However, due to the coarser cell, it could return the trajectory relatively faster. We ran 30 times to confirm repeatability in terms of execution times. In addition, Fig. 3.6 shows the trajectories for S.I and S.III. For S.III, $\tilde{\theta}$: 3.5° and $\tilde{\theta}$: 10° exceeded the time limit. We remark that we tested this offline. We also tried an online version of Hybrid-A* (which is adapted from the extended version of [35]) in order to make the setup exactly the same as our proposed work. However, the result was not promising. While passing through the restricted environment, reaching each local goal took more than the expected time (10 min).

It is worth analyzing what makes the online Hybrid-A* fail. Consider that we finished an iteration of the online version of Hybrid-A* after reaching the chosen local goals. After following the motion from the recent iteration, proceeding to the next iteration sometimes leads to difficult configurations. Since this plan is computed for a local optimal configuration, it can be different from the global optimal plan. In this case, the planner can get stuck or can take much more time to resolve this issue while computing a much longer distance path. In the worst case, the planning can be expanded over the whole local space. Post processing [59] can be helpful. In addition, choosing a proper resolution and a control duration for a specific road segment in

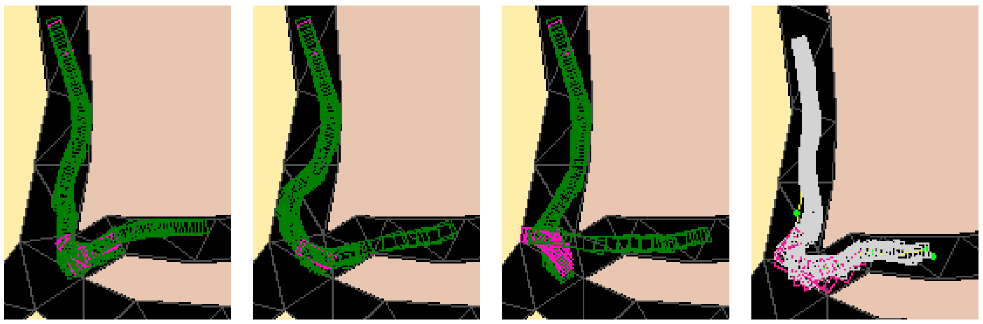

(a) $\tilde{\theta}$: 3.5° (b) $\tilde{\theta}$: 10° (c) $\tilde{\theta}$: 30° (d) $\tilde{\theta}$: N/A

Figure 3.5: Trajectories comparison of S.II between Hybrid-A* (a) - (c) and our proposed approach (d). Green vehicle poses in (a) - (c) and white vehicle poses in (d) represnt driving in a forward direction. Pink vehicle poses in (a) - (d) represent driving in a reverse direction.

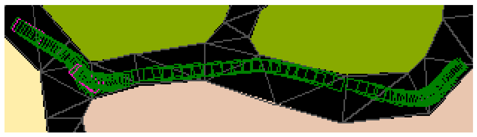
advance is difficult in general, and choosing proper local goals would be another future research topic [104].

We note that we did not compare our approach with RTR+TTS [65] because it does not generate a motion plan p as we do. However, it would be yet another future direction to combine their approach.

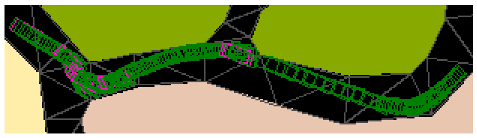
3.4 Experiments with a Physics-based Simulator

In this section, we introduce two possible extensions. First extension is for simulating our proposed approach through a 3D (Webots) simulator [105, 106]. Figure 3.7 shows an autonomous vehicle on a stiff curve in a rural road. This curve requires for the vehicle to move back and forth in order to pass through. The proposed approach was fast enough to generate the motion in on-the-fly manner.

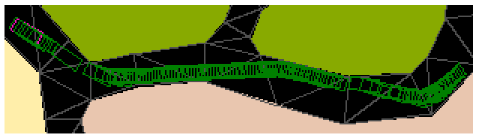
We used the model f in Eq. (3.1) in order to compute the lattice sets with the wheelbase and the steering limit of the vehicle we chose. We captured the road



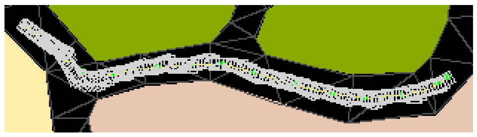
(a) $\tilde{\theta}$: 3.5°



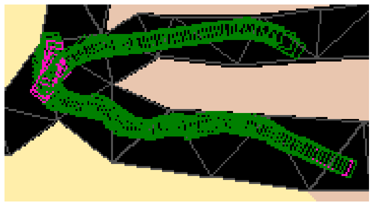
(b) $\tilde{\theta}$: 10°



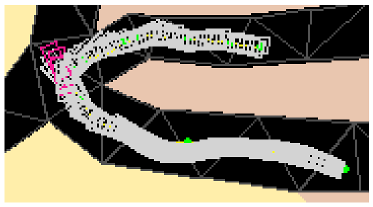
(c) $\tilde{\theta}$: 30°



(d) $\tilde{\theta}$: N/A



(e) $\tilde{\theta}$: 30°

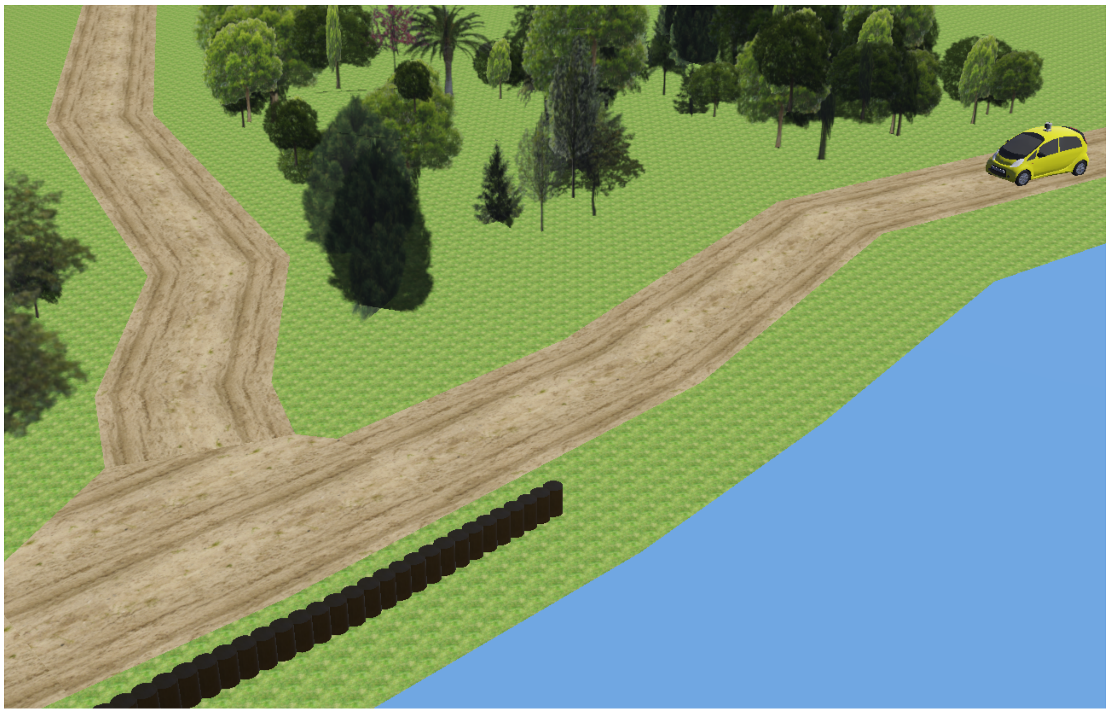


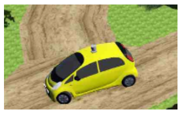
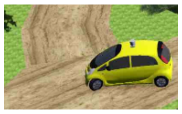
(f) $\tilde{\theta}$: N/A

Figure 3.6: Trajectories of S.I and S.III between Hybrid-A* (a) - (c), (e) and our proposed approach (d), (f). Green vehicle poses in (a) - (c) and (e) and white vehicle poses in (d) and (f) represnt driving in a forward direction. Pink vehicle poses in (a) - (c), (e) and (f) represent driving in a reverse direction.

from a map in OpenStreetMap and then, we imported it through Webots. Next, we decomposed the road into cells, constructing a road network. After choosing a source and target position, we could generate a motion plan.

The second extension is for multiple vehicles. When a vehicle meets another vehicle in the single lane road, one of the vehicles should go back in order for the other vehicle to pass through. For this extension, we considered two different types of vehicles in order to relax the assumption for the homogeneous robots (Sec. 2.6). Hence, these two vehicles have different wheelbase. However, for simplicity, we regarded that their

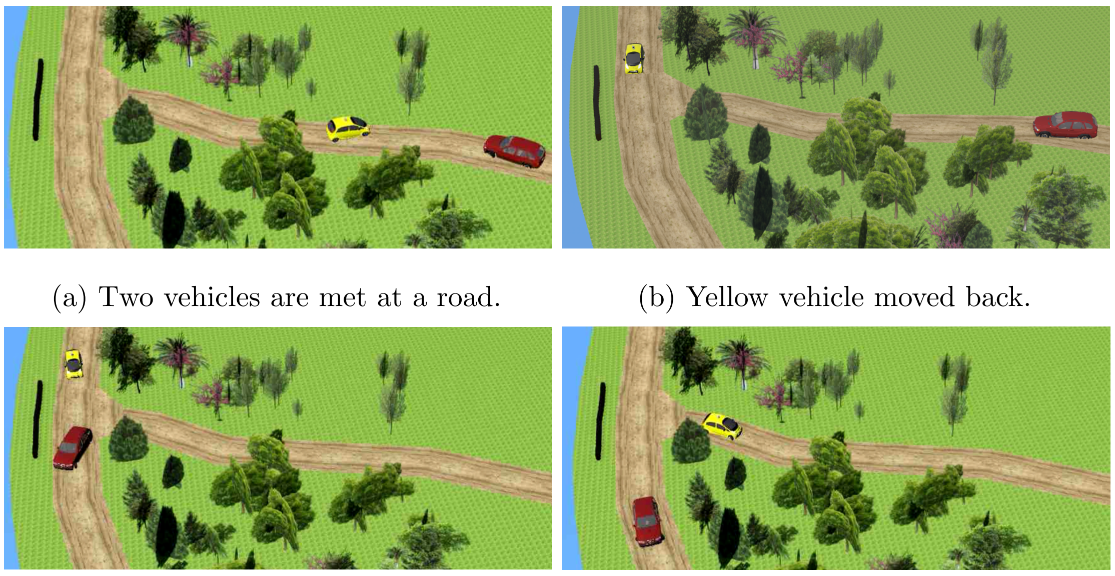

(a) Overview of the stiff curve.

(b) (c) (d) (e) (f) (g)

Figure 3.7: Single vehicle in a rural road. This is a 3D (Webots) simulation. (b) to (g) represent the vehicle's sequence of movements. (c) and (e) show driving in a reverse direction.

sensing and communication range is same. Due to the difference of the wheelbase and the size of each vehicle, we computed two different lattice sets. For the cooperative pathfinding, we utilized the technique mentioned in Sec. 2.6. Instead of having 2 D grid cells, we regarded the decomposed road segments and their connections as a graph. We computed the high level path with this graph, and we followed the path, generating the motion plan. For more about heterogeneous agents, refer the future direction in Sec. 3.5.



(a) Two vehicles are met at a road. (b) Yellow vehicle moved back.

(c) Red vehicle moved forward. (d) Both vehicles finally drive their way.

Figure 3.8: Multiful vehicles in a rural road. This is a 3D (Webots) simulation. (a) to (d) represent two vehicles' sequence of movements.

3.5 Conclusions and Future Directions

In this chapter, we introduced an online motion planner for autonomous vehicles in unstrucutured road networks. Then, we demonstrated this approach as a 3D simulator for a multi vehicle scenario.

Potential future directions are as follows:

1. **Generalized framework for challenging motions** When we choose a motion primitive from $\mathcal{M}^0$, the pre-defined radius and the pre-defined travel distance from the initial state strictly limit possible motions. For example, in some cases more than three point turns can be required. If so, this turning motions will increase the travel distance even if the last state of the motion is still within the

pre-defined radius. Hence, while manipulating, some guidance will be helpful. In this direction, techniques from [66] and [65] can be utilized.

2. **Utilizing an occupancy grid instead of rigid body detection** When our motion planning framework is adapted in an existing car platform, we have to use the data from sensors. Particularly, point clouds from LiDAR and also from other proximity sensors can be utilized. In this case, motion trajectories should be represented on an the occupancy grid. Then, while selecting a motion primitive from the lattice sets, an efficient method to prune unsafe motion primitives is necessary to be devised.

3. **Detecting a stiff curve before exploring local goals** Given a current state q' and local goals $l_i, l_{i+1, i+2}$, the current approach is to find a sequence of motion primitives from the lattice sets while changing the velocity until it reaches the last local goal l_{i+2}. This process eventually switches the lattice set to $\mathcal{M}^0$. However, it takes time to reach to $\mathcal{M}^0$. If we can detect the stiffness of the particular road, considering the current velocity and the steering, we can avoid searching and then switching to $\mathcal{M}^0$.

4. **Re-decomposition of the road network** For multiple vehicles and particularly heterogeneous agents, decompositions may be not sufficient to generate a motion plan. This is because while decomposing the road, we do not consider whether computing a trajectory on the cells is feasible or not. If it fails to pass through due to this issue, we can re-decompose the environment, changing the resolution of the cells. However, this can be repeated many times. Hence, a new classification for decomposing the environment while considering the motion primitives is necessary.

5. **Human-driven-AV interaction** Consider a real world situation when human drivers meet autonomous vehicles. Since we cannot communicate and deliver the new plan to human drivers as we discussed so far, there should be a new framework to resolve this situation. In order to deliver a new plan to human drivers, we have to interact with them in different ways: through head up or LCD displays, head lights, sounds, radio communications and mobile applications. We also have to know what they understood. It can be recognized through their motion (utilizing computer vision or accessing their vehicle control information and sensors), gesture and voice. In a demanding environment, knowing the human's driving skill is also important. Based on their skill level, different information can be provided. Another challenge could be uncooperative human drivers who do not follow the new plan as suggested by an autonomous vehicle. A simple solution could be to give them social penalties or rewards.

REFERENCES

[1] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, Cambridge, Massachusetts, 1999.

[2] Hadas Kress-Gazit, Georgios E. Fainekos, and George J. Pappas. Translating structured english to robot controllers. *Advanced Robotics*, 22(12):1343–1359, 2008.

[3] Juraj Dzifcak, Matthias Scheutz, Chitta Baral, and Paul Schermerhorn. What to do and how to do it: Translating natural language directives into temporal and dynamic logic representation for goal management and action execution. In *Proceedings of the IEEE Conference on Robotics and Automation*, 2009.

[4] Georgios E. Fainekos, Antoine Girard, Hadas Kress-Gazit, and George J. Pappas. Temporal logic motion planning for dynamic robots. *Automatica*, 45(2):343–352, February 2009.

[5] S. Karaman, R. Sanfelice, and E. Frazzoli. Optimal control of mixed logical dynamical systems with linear temporal logic specifications. In *IEEE Conf. on Decision and Control*, 2008.

[6] A. Bhatia, L. E. Kavraki, and M. Y. Vardi. Sampling-based motion planning with temporal goals. In *International Conference on Robotics and Automation*, pages 2689–2696. IEEE, 2010.

[7] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M. Murray. Receding horizon control for temporal logic specifications. In *Proceedings of the 13th ACM international conference on Hybrid systems: computation and control*, pages 101–110, New York, NY, USA, 2010. ACM.

[8] Pritam Roy, Paulo Tabuada, and Rupak @InProceedingsFilippidisDK12cdc, Title = Decentralized Multi-Agent Control from Local LTL Specifications, Author = Ioannis Filippidis and Dimos V. Dimarogonas and Kostas J. Kyriakopoulos, Booktitle = 51st IEEE Conference on Decision and Control, Year = 2012, Pages = pp. 6235-6240 Majumdar. Pessoa 2.0: a controller synthesis tool for cyber-physical systems. In *Proceedings of the 14th international conference on Hybrid systems: computation and control*, HSCC '11, pages 315–316, New York, NY, USA, 2011. ACM.

[9] Hadas Kress-Gazit, Morteza Lahijanian, and Vasumathi Raman. Synthesis for robots: Guarantees and feedback for robot behavior. *Annual Review of Control, Robotics, and Autonomous Systems*, 1:211–236, 2018.

[10] Hadas Kress-Gazit, Gerogios E. Fainekos, and George J. Pappas. Temporal logic based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25(6):1370 – 1381, 2009.

[11] M. Kloetzer and C. Belta. Automatic deployment of distributed teams of robots from temporal logic specifications. *IEEE Transactions on Robotics*, 26(1):48–61, 2010.

[12] Leonardo Bobadilla, Oscar Sanchez, Justin Czarnowski, Katrina Gossman, and Steven LaValle. Controlling wild bodies using linear temporal logic. In *Proceedings of Robotics: Science and Systems*, Los Angeles, CA, USA, June 2011.

[13] Alphan Ulusoy, Stephen L. Smith, Xu Chu Ding, Calin Belta, and Daniela Rus. Optimal multi-robot path planning with temporal logic constraints. In *IEEE/RSJ International Conference on Intelligent Robots and Systems,*, pages 3087 –3092, 2011.

[14] Bruno Lacerda and Pedro Lima. Designing petri net supervisors from ltl specifications. In *Proceedings of Robotics: Science and Systems*, Los Angeles, CA, USA, June 2011.

[15] Amy LaViers, Magnus Egerstedt, Yushan Chen, and Calin Belta. Automatic generation of balletic motions. *IEEE/ACM International Conference on Cyber-Physical Systems*, 0:13–21, 2011.

[16] Ioannis Filippidis, Dimos V. Dimarogonas, and Kostas J. Kyriakopoulos. Decentralized multi-agent control from local LTL specifications. In *51st IEEE Conference on Decision and Control*, pages pp. 6235–6240, 2012.

[17] Georgios E. Fainekos. Revising temporal logic specifications for motion planning. In *Proceedings of the IEEE Conference on Robotics and Automation*, May 2011.

[18] Giuseppe De Giacomo and Moshe Y. Vardi. Automata-theoretic approach to planning for temporally extended goals. In *European Conference on Planning*, volume 1809 of *LNCS*, pages 226–238. Springer, 1999.

[19] Alphan Ulusoy, Stephen L. Smith, and Calin Belta. Optimal multi-robot path planning with ltl constraints: Guaranteeing correctness through synchronization. In *DARS*, 2012.

[20] M. Kloetzer, X. C. Ding, and C. Belta. Multi-robot deployment from ltl specifications with reduced communication. In *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pages 4867–4872, Dec 2011.

[21] Jana Tumova and Dimos V. Dimarogonas. Multi-agent planning under local ltl specifications and event-based synchronization. *Automatica*, 70(C):239–248, August 2016.

[22] Meng Guo and Dimos V. Dimarogonas. Multi-agent plan reconfiguration under local ltl specifications. *Int. J. Rob. Res.*, 34(2):218–235, February 2015.

[23] P. Schillinger, M. Brger, and D. V. Dimarogonas. Multi-objective search for optimal multi-robot planning with finite ltl specifications and resource constraints. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 768–774, May 2017.

[24] Philipp Schillinger, Mathias Bürger, and Dimos V. Dimarogonas. Decomposition of finite ltl specifications for efficient multi-agent planning. In *DARS*, 2016.

[25] J. Tumova and D. V. Dimarogonas. Decomposition of multi-agent planning under distributed motion and task ltl specifications. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 7448–7453, Dec 2015.

[26] Meng Guo, Jana Tumova, and Dimos V. Dimarogonas. Cooperative decentralized multi-agent control under local ltl tasks and connectivity constraints. *53rd IEEE Conference on Decision and Control*, pages 75–80, 2014.

[27] Yu Zhang, Kangjin Kim, and Georgios Fainekos. DisCoF: Cooperative pathfinding in distributed systems with limited sensing and communication range. In *International Symposium on Distributed Autonomous Robotic Systems*, 2014.

[28] Boris de Wilde, Adriaan W. ter Mors, and Cees Witteveen. Push and rotate: Cooperative multi-agent path planning. In *12th International Conference on Autonomous Agents and Multiagent Systems*, 2013.

[29] R. Luna and K. Bekris. Efficient and complete centralized multirobot path planning. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2011.

[30] Howie Choset, Kevin M. Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. *Principles of Robot Motion: Theory, Algorithms and Implementations.* MIT Press, March 2005.

[31] Steven M. LaValle. *Planning Algorithms.* Cambridge University Press, 2006.

[32] David Gonzalez Bautista, Joshué Pérez, Vicente Milanés, and Fawzi Nashashibi. A Review of Motion Planning Techniques for Automated Vehicles. *IEEE Transactions on Intelligent Transportation Systems*, April 2016.

[33] J. Bellingham, A. Richards, and J.P. How. Receding horizon control of autonomous aerial vehicles. In *Proceedings of the 2002 American Control Conference (IEEE Cat. No.CH37301)*, pages 3741–3746 vol.5. IEEE, 2002.

[34] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, Kenny Lau, Celia Oakley, Mark Palatucci, Vaughan Pratt, Pascal Stang, Sven Strohband, Cedric Dupont, Lars-Erik Jendrossek, Christian Koelen, Charles Markey, Carlo Rummel, Joe van Niekerk, Eric Jensen, Philippe Alessandrini, Gary Bradski, Bob Davies, Scott Ettinger, Adrian Kaehler, Ara Nefian, and Pamela Mahoney. Stanley: The robot that won the darpa grand challenge: Research articles. *J. Robot. Syst.*, 23(9):661–692, September 2006.

[35] Janko Petereit, Thomas Emter, and Christian W. Frey. Application of Hybrid A* to an autonomous mobile robot for path planning in unstructured outdoor environments. In *Proceedings of the 7th German Conference on Robotics (ROBOTIK)*, 2012.

[36] Marsha Chechik and Arie Gurfinkel. Tlqsolver: A temporal logic query checker. In *Proceedings of the 15th International Conference on Computer Aided Verification*, volume 2725, pages 210–214. Springer, 2003.

[37] Arie Gurfinkel, Benet Devereux, and Marsha Chechik. Model exploration with temporal logic query checking. *SIGSOFT Softw. Eng. Notes*, 27(6):139–148, 2002.

[38] Yulin Ding and Yan Zhang. A logic approach for ltl system modification. In *15th International Symposium on Foundations of Intelligent Systems*, volume 3488 of *LNCS*, pages 435–444. Springer, 2005.

[39] Marcelo Finger and Renata Wassermann. Revising specifications with CTL properties using bounded model checking. In *Brazilian Symposium on Artificial Intelligence*, volume 5249 of *LNAI*, page 157166, 2008.

[40] J. G. Thistle and W. M. Wonham. Supervision of infinite behavior of discrete-event systems. *SIAM J. Control Optim.*, 32(4):1098–1113, 1994.

[41] Orna Kupferman, Wenchao Li, and Sanjit A. Seshia. A theory of mutations with applications to vacuity, coverage, and fault tolerance. In *Proceedings of the International Conference on Formal Methods in Computer-Aided Design*, pages 25:1–25:9, Piscataway, NJ, USA, 2008. IEEE Press.

[42] A. Cimatti, M. Roveri, V. Schuppan, and A. Tchaltsev. Diagnostic information for realizability. In Francesco Logozzo, Doron Peled, and Lenore Zuck, editors, *Verification, Model Checking, and Abstract Interpretation*, volume 4905 of *LNCS*, pages 52–67. Springer, 2008.

[43] R. Konighofer, G. Hofferek, and R. Bloem. Debugging formal specifications using simple counterstrategies. In *Formal Methods in Computer-Aided Design*, pages 152 –159. IEEE, November 2009.

[44] Vasumathi Raman and Hadas Kress-Gazit. Analyzing unsynthesizable specifications for high-level robot behavior using LTLMoP. In *23rd International Conference on Computer Aided Verification*, volume 6806 of *LNCS*, pages 663–668. Springer, 2011.

[45] Vasumathi Raman, Constantine Lignos, Cameron Finucane, Kenton C. T. Lee, Mitchell P. Marcus, and Hadas Kress-Gazit. Sorry dave, i'm afraid I can't do that: Explaining unachievable robot tasks using natural language. In *Robotics: Science and Systems IX, Technische Universität Berlin, Berlin, Germany, June 24 - June 28, 2013*, 2013.

[46] Vasumathi Raman and Hadas Kress-Gazit. Towards minimal explanations of unsynthesizability for high-level robot behaviors. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, Tokyo, Japan, November 3-7, 2013*, pages 757–762, 2013.

[47] Moritz Göbelbecker, Thomas Keller, Patrick Eyerich, Michael Brenner, and Bernhard Nebel. Coming up with good excuses: What to do when no plan can be found. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI Press, may 2010.

[48] David E. Smith. Choosing objectives in over-subscription planning. In *Proceedings of the 14th International Conference on Automated Planning and Scheduling (ICAPS-04)*, pages 393–401, 2004.

[49] Menkes van den Briel, Romeo Sanchez, Minh B. Do, and Subbarao Kambhampati. Effective approaches for partial satisfaction (over-subscription) planning. In *Proceedings of the 19th national conference on Artifical intelligence*, AAAI'04, pages 562–569. AAAI Press, 2004.

[50] Kris Hauser. The minimum constraint removal problem with three robotics applications. In *In Proceedings of the International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, 2012.

[51] Chitta Baral and Jicheng Zhao. Non-monotonic temporal logics for goal specification. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, pages 236–242, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[52] J. Tumova, L. I. Reyes Castro, S. Karaman, E. Frazzoli, and D. Rus. Minimum-violating planning with conflicting specifications. In *American Control Conference*, 2013.

[53] Tran Cao Son, Enrico Pontelli, and Chitta Baral. A non-monotonic goal specification language for planning with preferences. In *6th Multidisciplinary Workshop on Advances in Preference Handling*, 2012.

[54] M. Bienvenu, C. Fritz, and S. McIlraith. Planning with qualitative temporal preferences. In *International Conference on Principles of Knowledge Representation and Reasoning*, 2006.

[55] Trevor Standley. Finding optimal solutions to cooperative pathfinding problems. In *AAAI Conference on Artificial Intelligence*, 2010.

[56] J. Yu and S. M. LaValle. Multi-agent path planning and network flow. In *Algorithmic Foundations of Robotics X*, volume 86, pages 157–173. Springer, 2013.

[57] D. Silver. Cooperative pathfinding. In *Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2005.

[58] Zahy Bnaya and Ariel Felner. Conflict-oriented windowed hierarchical cooperative A*. In *Proceedings of the 2014 IEEE International Conference on Robotics and Automation*, 2014.

[59] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Path planning for autonomous vehicles in unknown semi-structured environments. In *International Journal of Robotics Research*, volume 29, pages 485–501, 2010.

[60] Sören Kammel, Julius Ziegler, Benjamin Pitzer, Moritz Werling, Tobias Gindele, Daniel Jagzent, Joachim Schröder, Michael Thuy, Matthias Goebl, Felix von Hundelshausen, Oliver Pink, Christian Frese, and Christoph Stiller. Team annieway's autonomous system for the 2007 darpa urban challenge. *J. Field Robot.*, 25(9):615–639, September 2008.

[61] Mihail Pivtoraiko and Alonzo Kelly. Differentially constrained motion replanning using state lattices with graduated fidelity. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2611–2616, 2008.

[62] Mihail Pivtoraiko, Ross A. Knepper, and Alonzo Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, mar 2009.

[63] A Lacaze, Y Moscovitz, N DeClaris, and K Murphy. Path planning for autonomous vehicles driving over rough terrain. In *Intelligent Control (ISIC)*, pages 50–55, 1998.

[64] Anirudh Vemula, Katharina Muelling, and Jean Oh. Path Planning in Dynamic Environments with Adaptive Dimensionality. *9th Annual Symposium on Combinatorial Search*, jun 2016.

[65] Domokos Kiss and Gbor Tevesz. Autonomous path planning for road vehicles in narrow environments: An efficient continuous curvature approach. *Journal of advanced transportation*, 2017:1–27, 10 2017.

[66] H. Banzhaf, N. Berinpanathan, D. Nienhuser, and J. M. Zollner. From g2 to g3 continuity: Continuous curvature rate steering functions for sampling-based nonholonomic motion planning. In *IEEE Intelligent Vehicles Symposition (IV)*, 2018.

[67] H. Banzhaf, L. Palmieri, D. Nienhuser, T. Schamm, S. Knoop, and J. M. Zollner. Hybrid curvature steer: A novel extend function for sampling-based nonholonomic motion planning in tight environments. In *IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–8, Oct 2017.

[68] Kangjin Kim, Georgios Fainekos, and Sriram Sankaranarayanan. On the revision problem of specification automata. In *Proceedings of the IEEE Conference on Robotics and Automation*, May 2012.

[69] Kangjin Kim and Georgios Fainekos. Approximate solutions for the minimal revision problem of specification automata. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[70] Kangjin Kim and Georgios Fainekos. Minimal specification revision for weighted transition systems. In *Proceedings of the IEEE Conference on Robotics and Automation*, 2013.

[71] Kangjin Kim and Georgios Fainekos. Revision of specification automata under quantitative preferences. In *Proceedings of the IEEE Conference on Robotics and Automation*, 2014.

[72] Kangjin Kim, Georgios Fainekos, and Sriram Sankaranarayanan. On the minimal revision problem of specification automata. *International Journal of Robotics Research (IJRR)*, 2015.

[73] Kangjin Kim, Joe Campbell, William Duong, Yu Zhang, and Georgios Fainekos. DisCoF$^+$: Asynchronous discof with flexible decoupling for cooperative pathfinding in distributed systems. In *IEEE International Conference on Automation Science and Engineering (IEEE CASE)*, August 2015.

[74] Kangjin Kim, Yu Zhang, and Georgios Fainekos. Online motion planning for autonomous vehicles in unstructured road networks. In *Proceedings of the IEEE Conference on Robotics and Automation*, 2020. (Submitted).

[75] S. Srinivas, R. Kermani, K. Kim, Y. Kobayashi, and G. Fainekos. A graphical language for ltl motion and mission planning. In *2013 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 704–709, Dec 2013.

[76] Wei Wei, Kangjin Kim, and Georgios Fainekos. Extended ltlvis motion planning interface. In *2016 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2016 - Conference Proceedings*, pages 4194–4199, United States, 2 2017. Institute of Electrical and Electronics Engineers Inc.

[77] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Cliff Stein. *Introduction to Algorithms.* MIT Press/McGraw-Hill, second edition, September 2001.

[78] Stephen L Smith, Jana Tůmová, Calin Belta, and Daniela Rus. Optimal path planning for surveillance with temporal-logic constraints*. *Int. J. Rob. Res.*, 30(14):1695–1708, December 2011.

[79] Kangjin Kim. Temporal logic specification revision and planning toolbox, 2014.

[80] Alphan Ulusoy, Stephen L. Smith, Xu Chu Ding, and Calin Belta. Robust multi-robot optimal path planning with temporal logic constraints. In *2012 IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[81] P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In G. Berry, H. Comon, and A. Finkel, editors, *Proceedings of the 13th CAV*, volume 2102 of *LNCS*, pages 53–65. Springer, 2001.

[82] Kangjin Kim. LTL2BA modification for indexing, 2014. https://git.assembla.com/ltl2ba_cpslab.git.

[83] E.Q.V. Martins, M.M.B. Pascoal, D.M.L.D. Rasteiro, and J.L.E. Dos Santos. The optimal path problem. *Investigacão Operacional*, 19:43–60, 1999.

[84] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[85] J.E. Hopcroft, J.T. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects; pspace- hardness of the "warehouseman's problem". *The International Journal of Robotics Research*, 3(4):76–88, 1984.

[86] Boris de Wilde, Adriaan ter Mors, and Cees Witteveen. Push and rotate: a complete multi-agent pathfinding algorithm. *J. Artif. Intell. Res. (JAIR)*, 51:443–492, 2014.

[87] M. Guo and M. M. Zavlanos. Probabilistic motion planning under temporal tasks and soft constraints. *IEEE Transactions on Automatic Control*, 63(12):4051–4066, Dec 2018.

[88] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. Journal of Robotics Research*, 30(7):846–894, June 2011.

[89] Yiannis Kantaros and Michael M. Zavlanos. Sampling-based optimal control synthesis for multirobot systems under global temporal tasks. *IEEE Transactions on Automatic Control*, 64:1916–1931, 2017.

[90] M. Lahijanian, S. B. Andersson, and C. Belta. Temporal logic motion planning and control with probabilistic satisfaction guarantees. *IEEE Transactions on Robotics*, 28(2):396–409, April 2012.

[91] E. M. Wolff, U. Topcu, and R. M. Murray. Robust control of uncertain markov decision processes with temporal logic specifications. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 3372–3379, Dec 2012.

[92] Marta Kwiatkowska and David Parker. Automated verification and strategy synthesis for probabilistic systems. In Dang Van Hung and Mizuhito Ogawa, editors, *Automated Technology for Verification and Analysis*, pages 5–22, Cham, 2013. Springer International Publishing.

[93] X. Ding, S. L. Smith, C. Belta, and D. Rus. Optimal control of markov decision processes with linear temporal logic constraints. *IEEE Transactions on Automatic Control*, 59(5):1244–1257, May 2014.

[94] Jie Fu and Ufuk Topcu. Probably approximately correct mdp learning and control with temporal logic constraints. *ArXiv*, abs/1404.7073, 2014.

[95] D. Sadigh, E. S. Kim, S. Coogan, S. S. Sastry, and S. A. Seshia. A learning based approach to control synthesis of markov decision processes for linear temporal logic specifications. In *53rd IEEE Conference on Decision and Control*, pages 1091–1096, Dec 2014.

[96] X. Li, C. Vasile, and C. Belta. Reinforcement learning with temporal logic rewards. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3834–3839, Sep. 2017.

[97] Xiao Li, Yao Ma, and Calin Belta. A policy search method for temporal logic specified reinforcement learning tasks. *2018 Annual American Control Conference (ACC)*, pages 240–245, 2017.

[98] Tomáš Brázdil, Krishnendu Chatterjee, Martin Chmelík, Vojtěch Forejt, Jan Křetínský, Marta Kwiatkowska, David Parker, and Mateusz Ujma. Verification of markov decision processes using learning algorithms. In Franck Cassez and Jean-François Raskin, editors, *Automated Technology for Verification and Analysis*, pages 98–114, Cham, 2014. Springer International Publishing.

[99] Alper Kamil Bozkurt, Yu Wang, Michael M. Zavlanos, and Miroslav Pajic. Control synthesis from linear temporal logic specifications using model-free reinforcement learning. *ArXiv*, abs/1909.07299, 2019.

[100] Brandon Araki, Kiran Vodrahalli, Thomas Leech, Cristian Ioan Vasile, Mark Donahue, and Daniela Rus. Learning to Plan with Logical Automata. In *Robotics: Science and Systems Conference (RSS)*, pages 1–9, Messe Freiburg, Germany, June 2019. link.

[101] J. Barraquand and J.-C. Latombe. Nonholonomic multibody mobile robots: controllability and motion planning in the presence of obstacles. In *IEEE International Conference on Robotics and Automation*, pages 2328–2335, 1991.

[102] Mentar Mahmudi and Marcelo Kallmann. Precomputed motion maps for unstructured motion capture. In *Eurographics/SIGGRAPH Symposium on Computer Animation (SCA)*, 2012.

[103] Chang-bae Moon and Woojin Chung. Kinodynamic planner dual-tree rrt (dt-rrt) for two-wheeled mobile robots using the rapidly exploring random tree. *IEEE Transactions on Industrial Electronics*, 62:1–1, 01 2014.

[104] J. D. Hernndez, M. Moll, and L. E. Kavraki. Lazy evaluation of goal specifications guided by motion planning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 944–950, May 2019.

[105] O. Michel. Webots: Professional mobile robot simulation. *Journal of Advanced Robotics Systems*, 1(1):39–42, 2004.

[106] Webots. http://www.cyberbotics.com. Open Source Mobile Robot Simulation Software. It provides commercial supports and consulting.

[107] Dmitry S. Yershov and Steven M. LaValle. Sufficient conditions for the existence of resolution complete planning algorithms. In *Algorithmic Foundations of Robotics IX*, volume 68 of *STAR*, pages 303–320, 2010.

APPENDIX A

NP-COMPLETENESS OF THE MINIMAL CONNECTING EDGE PROBLEM

We will prove the Minimal Connecting Edge (MCE) problem is NP-Complete. MCE is a slightly simpler version of the Minimal Accepting Path (MAP) problem and, thus, MAP is NP-Complete as well.

In MCE, we consider a directed graph $G = (E, V)$ with a source s and a sink t where there is no path from s to t. We also have a set of candidate edges $\hat{E}$ to be added to E such that the graph becomes connected and there is a path from s to t. Note that if the edges in $\hat{E}$ have no dependencies between them, then there exists an algorithm that can solve the problem in polynomial time. For instance, Dijkstra's algorithm [77] applied on the weighted directed graph $G = (V, E \cup \hat{E}, w)$ where the edges in $\hat{E}$ are assigned weight 1 and the edges in E are assigned weight 0 solves the problem efficiently.

However, in MCE, the set $\hat{E}$ is partitioned in a number of classes $\hat{E}_1, ..., \hat{E}_n$ such that if an edge e_i is added from $\hat{E}_i$, then all the other edges in $\hat{E}_i$ are added as well to G. This corresponds to the fact that if we remove a predicate from a transition in $\mathcal{B}_{\mathbf{s}}$, then a number of transitions on $G_{\mathcal{A}}$ are affected. Let us consider the $G_{\mathcal{A}}$ in Fig. 2.3 as an example. Here, e_0, e_2 and e_4 correspond to $y((s_1, s_1), \pi_0)$, e_1 and e_5 to $y((s_1, s_1), \pi_2)$ and e_3 to $y((s_1, s_1), \pi_3)$. Thus, $\{e_0, e_1, e_2, e_3, e_4, e_5\} \in \hat{E}$ and there exist three classes $\hat{E}_i$, $\hat{E}_j$ and $\hat{E}_j$ in the partition such that $\{e_0, e_2, e_4\} \subseteq \hat{E}_i$, $\{e_1, e_5\} \subseteq \hat{E}_j$ and $e_3 \in \hat{E}_k$.

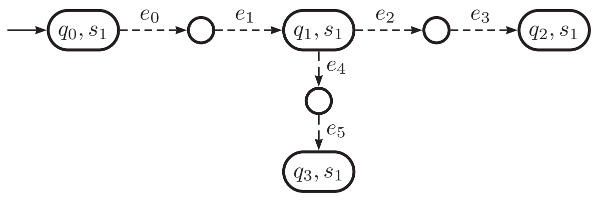


Figure A.1: The MCE instance that corresponds to $G_{\mathcal{A}}$ from Fig. 2.3. The dashed edges denote candidate edges in $\hat{E}$.

Problem 9 (Minimal Connecting Edge (MCE))**.** Input: *Let $G = (V, E)$ be a directed graph with a source s and a distinguished sink node t. We assume that there is no path in G from s to t. Let $\hat{E} \subseteq V \times V$ be a set such that $\hat{E} \cap E = \emptyset$. We partition $\hat{E}$ into $\mathcal{E} = \{\hat{E}_1, \dots, \hat{E}_m\}$. Each edge $e \in \hat{E}$ has a weight $W(e) \geq 0$.*

Output: *Given a weight limit W, determine if there is a selection of edges $R \subseteq \hat{E}$ such that*

1. *there is a path from s to t in the graph with all edges $E \cup R$,*
2. *$\sum_{e \in \cup \mathcal{R}} W(e) \leq W$ and*
3. *For each $\hat{E}_i \in \mathcal{E}$, if $\hat{E}_i \cap R \neq \emptyset$ then $\hat{E}_i \subseteq R$.*

Theorem 10. *MCE is NP-complete.*

Proof. The problem is trivially in NP. Given a selection of edges from $\hat{E}$, we can indeed verify that the source and sinks are connected, the weight limit is respected and that the selection is made up of a union of sets from the partition.

We now claim that the problem is NP-Complete. We will reduce from 3-CNF-SAT. Consider an instance of 3-CNF-SAT with variables $X = \{x_1, \dots, x_n\}$ and clauses $C_1, \dots, C_m$. Each clause is a disjunction of three literals. We will construct graph G and family of edges $\mathcal{E}$. The graph G has edges E made up of variable and clause "gadgets".

Variable Gadgets For each variable x_i, we create 6 nodes u_i, u_i^t, v_i^t, u_i^f, v_i^f, and v_i. The gadget is shown in Fig. A.2. The node u_i is called the entrance to the gadget and v_i is called the exit. The idea is that if the variable is assigned true, we will take the path

$$u_i \to u_i^t \to v_i^t \to v_i$$

to traverse through the gadget from its entrance to exit. The missing edge $u_i^t \to v_i^t$ will be supplied by one of the edge sets. If we assign the variable to false, we will instead traverse

$$u_i \to u_i^f \to v_i^f \to v_i$$

Variable gadgets are connected to each other in G by adding edges from v_1 to u_2, v_2 to u_3 and so on until $v_{n-1} \to u_n$. The node u_1 is the source node.

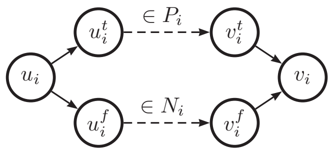


Figure A.2: A single variable gadget. Solid edges are present in the original graph G that will be constructed. Dashed edges (u_i^t, v_i^t) or between (u_i^f, v_i^f) will be supplied by one of the edge sets in $\hat{E}$.

Clause Gadgets For each clause C_j of the form $(\ell_{j1} \vee \ell_{j2} \vee \ell_{j3})$, we add a clause gadget consisting of eight nodes: entry node a_j, exit node b_j and nodes a_{j1}, b_{j1}, a_{j2}, b_{j2} and a_{j3}, b_{j3} corresponding to each of the three literals in the clause. The idea is that a path from the entry node a_j to exit node b_j will exist if the clause C_j will be satisfied. Figure A.3 shows how the nodes in a clause gadget are connected.

Structure We connect v_n the exit of the last variable gadget for variable x_n to a_1, the entrance for first clause gadget. The sink node is b_m, the exit for the last clause gadget. Figure A.4 shows the overall high level structure of the graph G with variable and clause gadgets.

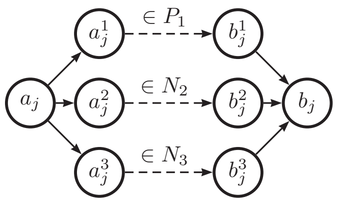


Figure A.3: The clause gadget for a clause with three literals. The clause shown here is $(x_1 \vee \overline{x_2} \vee \overline{x_3})$. The corresponding missing edges will be added to the set P_1, N_2, N_3, respectively, as shown in figure.

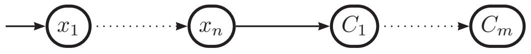


Figure A.4: Connection between gadgets for variables and clauses.

Edge Sets We design a family $\mathcal{E} = \{P_1, \ldots, P_n, N_1, \ldots, N_n\}$. The set P_i will correspond to a truth assignment of true to variable x_i and N_i correspond to a truth assignment of false to x_i.

P_i has the edge (u_i^t, v_i^t) of weight 1 and for each clause C_j containing the literal x_i, we add the missing edge (a_j^i, b_j^i) corresponding to this literal in the clause gadget for C_j to the set P_i with weight 0.

Similarly, N_i has the edge from (u_i^f, v_i^f) of weight 1 and for each clause C_j containing the literal $\overline{x_i}$ it has the missing edge in the clause gadget for C_j with weight 0. We ask if there is a way to connect the source u_1 with the sink b_m with weight limit $\leq n$, where n is the number of variables.

We verify that the sets $P_1, \ldots, P_n, N_1, \ldots, N_n$ partition the set of missing edges.

Claim 1. *If there is a satisfying solution to the problem, then u_1 can be connected to b_m by a choice of edge sets with total edge weight $\leq n$.*

Proof. Take a satisfying solution. If it assigns true to x_i, then choose all edges in P_i else choose all edges N_i if it assigns false. We claim that this will connect u_1 to b_m. First it is clear that since all variables are assigned, it will connect u_1 to v_n by connecting one of the two missing links in each variable gadget. Corresponding to each clause, C_j there will be a path from a_j to b_j in the clause gadget for C_j. This is because, at least one of the literals in the clause is satisfied and the corresponding set P_i or N_i will supply the missing edge. Furthermore, the weight of the selection will be precisely n, since we add exactly one edge in each variable gadget. □

Claim 2. *If there is a way to connect source to sink with weight $\leq n$ then a satisfying assignment exists.*

Proof. First of all, the total weight for any edge connection from source to sink is $\geq n$ since we need to connect u_1 to v_n there are n edges missing in any shortest path. The edges that will connect have weight 1, each. Therefore, if there is a way to connect source to sink with weight $\leq n$, the total weight must in fact be n. This allows us to conclude that for every variable gadget precisely one of the missing edges is present. As a result, we can now form a truth assignment setting x_i to true if P_i is chosen and false if N_i is. Therefore, the truth assignment will assign either true to x_i or false and not both thanks to the weight limit of n. □

Next, we prove that each a_j will be connected to b_j in each clause gadget corr. to clause C_j. Let us assume that this was using the edge $(a^i_j, b^i_j) \in N_i$. Then, by construction have that $\overline{x_i}$ was in the clause C_j which is now satisfied since N_i is chosen, assigning x_i to false. Similar reasoning can be used if $(a_j, b_j) \in P_i$. Combining, we conclude that all clauses are satisfied by our truth assignment. □

APPENDIX B

UPPER BOUND OF THE APPROXIMATION RATIO OF AAMRP

We shall show the upper bound of the approximation algorithm (AAMRP) for a special case.

Theorem 11. *AAMRP on planar Directed Acyclic Graphs (DAG) where all the paths merge on the same node is a polynomial-time 2-approximation algorithm for the Minimal Revision Problem (MRP).*

Proof. We have already seen that the AAMRP runs in polynomial time.

Let $Y = \{y_1, \dots, y_m\}$ be a set of Boolean variables and $G : (V, E)$ be a graph with a labeling function $L : E \to \mathcal{P}(Y)$, wherein each edge $e \in E$ is labeled with a set of Boolean variables $L(e) \subseteq Y$. The label on an edge indicates that the edge is *enabled* iff all the Boolean variables on the edge are set to true. Let $v_0 \in V$ be a marked initial state and $F \subseteq V$ be a set of marked final vertices.

Consider two functions $w' : E^* \to \mathcal{P}(Y)$, and $w : E^* \to \mathbb{N}$ where E^* represents the set of all finite sequences of edges of the graph G. Hence, $w'(P)$ for a path $P = \langle v_0, \dots, v_k \rangle$ is a set of boolean variables of its constituent edges which makes them enabled on the path P:

$$w'(P) = \bigcup_{i=1}^{k} L(v_{i-1}, v_i)$$

while $w(P)$ is the number of the boolean variables of

$$w(P) = \left| \bigcup_{i=1}^{k} L(v_{i-1}, v_i) \right| = \left| w'(P) \right|$$

Given a initial vertex v_0, two vertices v_i, v_j, and a final vertex v_k, let P_{opt} denote the path that produces an optimal revision for the given graph. Let P_a denote a general revision by AAMRP. Suppose that P_{opt} consists of subpaths P_{0i}, P_{ij}, P_{jk}, and P_a consists of subpaths P_{0i}, P'_{ij}, P_{jk}.

We will discuss the cases when P_{0i} and P_{jk} are empty later. The former case can occur when P_{opt} and P_a do not have any common edges from v_0 to v_i in the sense that each path takes a different neighbor out of v_0. This case can also occur when $0 < i$ if from v_0 to v_i there is no boolean variables to be enabled to make the path activated. Likewise, the latter case can occur when $i < j < k$ or when $i \leq j = k$. We do not take $i = j$ unless $j = k$. Considering both cases together, we can get the possibility that P_{opt} and P_a are entirely different from v_0 to v_k.

In v_j, the AAMRP should relax the weight of the path from v_0 to v_j, comparing between two paths P_{ij} and P'_{ij}. Thus, we can denote:

$$w'(P_m) = w'(P_{0i}) \cup w'(P'_{ij}) \cup w'(P_{jk}),$$

$$w'(P^*) = w'(P_{0i}) \cup w'(P_{ij}) \cup w'(P_{jk}).$$

Let $w'(P_{0i}) \cup w'(P'_{ij}) = \Lambda_a$, $w'(P_{0i}) \cup w'(P_{ij}) = \Lambda_{opt}$, and $w'(P_{jk}) = \Lambda$. Then, we can denote:

$$w'(P_a) = \Lambda_a \cup \Lambda,$$

$$w'(P_{opt}) = \Lambda_{opt} \cup \Lambda.$$

Recall that

$$w(P_a) = |\Lambda_a \cup \Lambda|,$$

$$w(P_{opt}) = |\Lambda_{opt} \cup \Lambda|.$$

We will show that $w(P_a) \leq 2w(P_{opt})$.

Note that $w(P_{opt}) \geq 1$, so that $|\Lambda_{opt} \cup \Lambda| \geq 1$. This is because if $w(P_{opt}) = 0$, then it is reachable from v_0 to v_f without enabling any boolean variables which are atomic propositions of the specification.

Remark 2. $|\Lambda_{opt} \cup \Lambda| \geq 1$.

Note that $|\Lambda_a| \leq |\Lambda_{opt}|$. This is because the AAMRP only relaxes the path when it has less number of boolean variables.

Remark 3. $|\Lambda_a| \leq |\Lambda_{opt}|$.

Note that if $|\Lambda_a| = 0$, then $|\Lambda_{opt} \cup \Lambda| = |\Lambda_a \cup \Lambda| \leq 2|\Lambda_{opt} \cup \Lambda|$. In this case, Λ_a is the optimal path if $\Lambda_a = 0$ since Λ is common for the two paths. I.e., $w(P_a) \leq 2w(P_{opt})$.

Consider the case $|\Lambda_a| \geq 1$. We will prove the claim by contradiction. Assume that $2w(P_{opt}) < w(P_a)$ so that $2|\Lambda_{opt} \cup \Lambda| < |\Lambda_a \cup \Lambda|$. Let $|\Lambda_a| = \mu$, $|\Lambda_{opt}| = \eta$ and $|\Lambda| = \tau$. There are four cases.

Case 1: if $\Lambda_{opt} \cap \Lambda = \emptyset$ and $\Lambda_a \cap \Lambda = \emptyset$, then

$$2|\Lambda_{opt} \cup \Lambda| < |\Lambda_a \cup \Lambda| \Rightarrow 2(\eta + \tau) < \mu + \tau$$

$$2\eta + 2\tau < \mu + \tau \Rightarrow 2\eta + \tau < \mu$$

However, $\mu \leq \eta$ by Remark 3. Thus, $\eta + \tau < 0$ which is not possible and it contradicts our assumption.

Case 2: if $\Lambda_{opt} \cap \Lambda \neq \emptyset$ and $\Lambda_a \cap \Lambda = \emptyset$, then let $|\Lambda_{opt} \cap \Lambda| = \zeta$, where $1 \leq \zeta \leq \min(\eta, \tau)$.

$$2|\Lambda_{opt} \cup \Lambda| < |\Lambda_a \cup \Lambda| \Rightarrow 2(\eta + \tau - \zeta) < \mu + \tau$$

$$2\eta + 2\tau - 2\zeta < \mu + \tau \Rightarrow 2\eta - 2\zeta + \tau < \mu$$

If $\eta \leq \tau$, then $\zeta \leq \eta$ and $\eta = \zeta + \alpha$, for some $\alpha \geq 0$.

$$2(\zeta + \alpha) - 2\zeta + \tau < \mu \Rightarrow 2\alpha + \tau < \mu$$

However, $\eta \leq \tau$ and $\mu \leq \eta$ by Remark 3. Thus, $2\alpha < 0$ which is not possible and it contradicts our assumption.

If $\eta > \tau$ and $\eta = \tau + \beta$, for some $\beta > 0$, then $\zeta \leq \tau$ and $\tau = \zeta + \alpha$, for some $\alpha \geq 0$.

$$2(\tau + \beta) - 2\zeta + \tau < \mu \Rightarrow 2\tau - 2\zeta + 2\beta + \tau < \mu$$

$$2(\zeta + \alpha) - 2\zeta + 2\beta + \tau < \mu \Rightarrow 2\zeta + 2\alpha - 2\zeta + 2\beta + \tau < \mu$$

$$2\alpha + 2\beta + \tau < \mu \Rightarrow 2\alpha + 2\beta + \eta - \beta < \mu$$

$$2\alpha + \beta + \eta < \mu$$

However, $\mu \leq \eta$ by Remark 3. Thus, $2\alpha + \beta < 0$ which is not possible and it contradicts our assumption.

Case 3: if $\Lambda_{opt} \cap \Lambda = \emptyset$, $\Lambda_a \cap \Lambda \neq \emptyset$, then let $|\Lambda_a \cap \Lambda| = \theta$, where $1 \leq \theta \leq \min(\mu, \tau)$.

$$2|\Lambda_{opt} \cup \Lambda| < |\Lambda_a \cup \Lambda| \Rightarrow 2(\eta + \tau) < \mu + \tau - \theta$$

$$2\eta + \tau < \mu - \theta \Rightarrow 2\eta + \tau < \mu$$

However, $\mu \leq \eta$ by Remark 3. Thus, $\eta + \tau < 0$ which is not possible and it contradicts our assumption.

Finally the last case: if $\Lambda_{opt} \cap \Lambda \neq \emptyset$, $\Lambda_a \cap \Lambda \neq \emptyset$, then let $|\Lambda_{opt} \cap \Lambda| = \zeta$, where $1 \leq \zeta \leq \min(\eta, \tau)$, and $|\Lambda_a \cap \Lambda| = \theta$, where $1 \leq \theta \leq \min(\mu, \tau)$.

$$2|\Lambda_{opt} \cup \Lambda| < |\Lambda \cup \Lambda| \Rightarrow 2(\eta + \tau - \zeta) < \mu + \tau - \theta$$

$$2\eta + 2\tau - 2\zeta < \mu + \tau - \theta \Rightarrow 2\eta + \tau - 2\zeta < \mu - \theta < \mu$$

If $\eta \leq \tau$, $\zeta \leq \eta$ and $\eta = \zeta + \alpha$, for some $\alpha \geq 0$, then

$$2(\zeta + \alpha) + \tau - 2\zeta < \mu \Rightarrow 2\alpha + \tau < \mu$$

However, $\mu \leq \eta$ and $\eta \leq \tau$ by Remark 3. Thus, $2\alpha < 0$ which is not possible and it contradicts our assumption.

If $\eta > \tau$, $\eta = \tau + \beta$, for some $\beta > 0$, $\zeta \leq \tau$ and $\tau = \zeta + \alpha$, for some $\alpha \geq 0$, then

$$2\eta + \tau - 2\zeta < \mu \Rightarrow \eta + (\tau + \beta) + \tau - 2\zeta < \mu$$

$$\eta + (\zeta + \alpha) + \beta + (\zeta + \beta) - 2\zeta < \mu \Rightarrow \eta + 2\zeta + \alpha + 2\beta - 2\zeta < \mu$$

$$\eta + \alpha + 2\beta < \mu$$

However, $\mu \leq \eta$ by Remark 3. Thus, $\alpha + 2\beta < 0$ which is not possible and it contradicts our assumption.

□

Therefore, $|\Lambda_a \cup \Lambda| \leq 2|\Lambda_{opt} \cup \Lambda|$, and we can conclude that $w(P_a) \leq 2w(P_{opt})$.

APPENDIX C

PROOFS FOR MOTION PLANNING

The next proof is mostly based on [107] and [101]. First, from [107], we borrowed the technique to show how a feasible motion plan can be established through a system and its input. Second, we used the same method in [101] to resolve the boundary condition. Authors in [101] introduced Car Grid Search and addressed that it is resolution complete. Since each turn action in their planner is based on left most turn or right most turn, they used the following proposition: if a feasible free path between two given configurations exists, then there also exists a feasible free path between these configurations that are only controlled with the left most and the right most turn. They use Car Grid Search to find the goal and stop searching it if the expanding tree exceeded the pre-defined configuration space while we use Car Grid Search for pre-computing the tree expanding up to the pre-defined space limit and then search a near node of the goal from the tree.

C.1 Resolution Complete of Multi-resolution Motion Planner

Theorem 12 (Resolution Complete)**.** *If Prob. 8 has a motion plan p which can be represented in some resolution of the workspace, then Alg. 17 to 19 can find one with the decomposition of the workspace in multi-resolution and pre-computed & pre-constructed lattice sets. It indicates failure otherwise. In both cases, it returns an answer in a finite time.*

Proof. First, we will show how a feasible motion plan can be found, given that a solution exists. For this, we will consider two cases: a case for finding a motion plan with $\mathcal{M}^1$ and the case for finding a motion plan with $\mathcal{M}^0$.

Since a motion plan exists, we safely assume that there is a high level path p_h^* and its smoothed path $p_h^*.Spine$. This $p_h^*.Spine$ is a sequence of coordinates from q_s to q_g. Consider these coordinates as strings. Given the current coordinates q_c, a local goal l_g from these coordinates can be transformed to the local coordinates where each coordinates is originated from its q_c. Then, we can use this sequence of local coordinates as input strings and consider each local coordinates as an alphabet of Σ. Let's consider $\mathcal{M}^1$ as a system $\mathcal{T}$; since $\mathcal{M}^1$ is the superset of all $\mathcal{M}^\varrho$ where $\varrho \geq 1$, we will here simply consider $\mathcal{M}^1$ and its set of states η^1. Now, we will show how the system $\mathcal{T}$ and the input Σ can produce a motion plan p. Consider that q_s is the initial state of $\mathcal{T}$. Take this q_s as the current state q_c. Given a string σ_i in Σ, we can pick a η in η^1. This η is within the boundary γ_{lg}. It means $d_3(\eta, \sigma_i) \leq \gamma_{lg}$. Depending on the size of each cell $(\tilde{x}, \tilde{y})$ in $\mathcal{M}^1$, there can be more than one η within γ_{lg}. Hence, when we make the cell finer, more η can be within γ_{lg}. Next, the motion plan p' from $\tilde{q}_s$ to η can be found from $\mathcal{A}^\mathcal{J}$ of $\mathcal{M}^1$ after transforming back to the global coordinates where they are originated from the q_c. Then, we can check the collision with the q_c's near obstacles. If it is collided, then we can pick the next η from η^1 which is within γ_{lg}, repeating the above steps. Otherwise, we can continue for the next string σ_j in Σ, progressing the current coordinates q_c. Concatenating all this plan p', we can construct a feasible motion plan p from q_s to q_g.

Consider the case when $\mathcal{M}^0$ is chosen because no η left in $\mathcal{M}^1$. While we produced the motion plan with $\mathcal{M}^1$ as the system, we only considered one way driving direction. Then, failures can happen if all η within γ_{lg} are collided with the obstacles or there is no η within γ_{lg} due to the environment and the fixed driving direction. Then, we can switch the system from $\mathcal{M}^1$ to $\mathcal{M}^0$. $\mathcal{M}^0$ is computed with motion primitives which enable to progress in a bi-directional way. In addition, since Alg. 18 always switches back to $\mathcal{M}^1$ once it produces a nonempty motion plan through $\mathcal{M}^0$, we here will focus on finding a motion plan from the current coordinates q_c to the local goal l_g. In particular, we will show how the q_c can be connected to a coordinates near by l_g within γ_{lg} through a sequence of feasible motion plans.

We discretized the configuration space into an array of cells. Each cell in this grid is 3 dimensional. Given a coordinates $(x, y, \theta, v, \zeta) = q' \in Q$, the corresponding cell has a key-value pair. The key consists of $(\tilde{x}, \tilde{y}, \tilde{\theta})$ and the value consists of (x, y, θ, v, ζ) such that $|\tilde{x} - x| \leq \Delta x$, $|\tilde{y} - y| \leq \Delta y$, $|\tilde{\theta} - \theta| \leq \Delta\theta$ where $\Delta\cdot$ is a cell size for each dimension. We denote the key $(\tilde{x}, \tilde{y}, \tilde{\theta})$ of q' as $\hat{q}'$. Then, $d_3(q', \hat{q}') \leq \hat{\gamma}$ such that $d_{xy}(q', \hat{q}') \leq \hat{\gamma}^{xy}$ and $d_\theta(q', \hat{q}') \leq \hat{\gamma}^\theta$. We choose sufficiently small values for this $\hat{\gamma}$.

We also discretize the workspace into an array of 2D cells. This is for checking if an obstacle in the workspace is occupied in some particular cell. With this decomposition, we assume that if a cell is occupied, it is $\top$. Otherwise, it is $\bot$. Let denote the cell size $\hat{\gamma}_o = (\hat{\gamma}_o^{xy}, 2 \cdot \hat{\gamma}^\theta)$ where $\hat{\gamma}_o^{xy}$ is the Euclidean distance of each cell in 2D.

Proposition 9. *Each cell of the grid in configuration space and each cell of the grid in workspace are fine enough such that* $\hat{\gamma} < \hat{\gamma}_o$ *and* $2 \cdot \hat{\gamma} \leq \gamma_{lg}$

In $\mathcal{M}^0$, each $q' \in \tilde{Q}$ can have at most 6 successors $q_i = (x_i, y_i, \theta_i, 0, 0)$ where $i \in \{1, \ldots, 6\}$. We can represent the distance from q' to each q_i as $d_3(q', q_i)$. When this distance is bigger than the cell size, we can make the successors being progressed from q'. In addition, in order to safely reach l_g, it should be less than the boundary for the local goal γ_{lg}.

Proposition 10. $\forall q' \in \tilde{Q}$ *in* $\mathcal{M}^0, \forall$*successor* q_i *of* $q', \hat{\gamma} \lesssim d_3(q', q_i)$ *and* $d_3(q', q_i) < min\{2 \cdot \hat{\gamma}, \hat{\gamma}_o, \gamma_{lg}\}$

Here, $d_3(q', q_i) \lesssim \gamma$ denotes that only one of the conditions for d_3 is satisfied.

When we choose a local goal l_g through $p_h^*.Spine$, the l_g should be always within the coverage of $\mathcal{M}^0$.

Proposition 11. $\forall q_c \in q, \forall l_g$ *of* $q_c, d_{xy}(q_c, l_g) <$ *local goal coverage of* $\mathcal{M}^0$

Even we choose the l_g under the coverage of $\mathcal{M}^0$, we have not considered the environment including the obstacles near in. If it is not feasible to take any motion due to this limitation, Alg. 19 and Alg. 18 return an empty plan. Then, Alg. 17 can get an alternative path, repeating the early described procedure again. If no alternative path is found, Alg. 17 returns failure. However, this contradicts our assumption. Hence, we assume that a feasible motion plan exists within $\mathcal{M}^0$'s coverage.

Since in $\mathcal{M}^0$, the resolution of the KD-Tree is equal to the pre-defined grid, each $q' \in \tilde{Q}$ in $\mathcal{M}^0$ can be searched by KD-Tree. In addition, for each q', there is a way to get a sequence of nodes from the root node $\tilde{q}_s$ of the tree $\mathcal{A}^{\mathcal{J}}$ to q'.

Proposition 12. $\forall \eta \in \mathcal{M}^0$*, a motion plan* $p := \mathcal{M}^0[\eta]$ *exists from* $\tilde{q}_s$ *to* η

Given above considerations, when we take q_c as $\tilde{q}_s$ of $\mathcal{M}^0$, we can find cells closer to the l_g within γ_{lg} radius.

Proposition 13. *Prop.* $9, 10, 11 \implies \exists \eta \in \mathcal{M}^0_{q_c}$ *s.t.* $d_3(q_c^\eta, l_g) \leq \gamma_{lg}$

We can retrieve a sequence of nodes from these cells to its root node q_c. Once we simulate this sequence of nodes, checking whether it is collided with the obstacles, we can get a feasible motion plan.

Proposition 14. *Prop.* $12, 13 \implies$ *a motion plan* $p_{q_c} := \mathcal{M}^0_{q_c}[\eta]$ *exists*

Second, we will show how it is guaranteed to terminate within a finite time for the following three cases.

Case 1 In Alg. 17, the while loop is continued until the current coordinates q_c reaches near by the goal coordinates q_g. While progressing to q_g, it calls Alg. 18 to get the motion plan. When it gets the plan, it follows the plan for the duration κ. This process gradually makes the distance from q_c to q_g closer. Hence, assuming that the motion plan exists and the distance to q_g is finite, the while loop is finished in a finite time.

Case 2 Now, we consider the case when it failed to get the motion plan for some part of the path. In this case, this part of the path is added to the block set, and an alternative path is computed. Then, the process to get the motion plan will be repeated. This will lead the while loop is terminated too.

Case 3 Lastly, when it cannot compute an alternative path, considering the block set, it returns failure. This also makes the algorithm being terminated immediately. □

C.2 Correctness of Multi-resolution Motion Planner

Theorem 13 (Correctness)**.** *Algorithm 17 to 19 solve Prob. 8.*

Proof. In Alg. 17, we can progress to the given goal q_g. The while loop is continued until the current coordinates q_c reaches near by q_g. In each iteration, it calls Alg. 18 in order to get the local plan, and then it updates q_c, progressing as long as the duration κ. In Alg. 18, it sets up the local goals from q_c for the duration κ. Then, it calls Alg. 19 in order to get the motion plan. In Alg. 19, it finds a motion plan, checking the collision with the near obstacles.

From the above procedure, if a feasible motion plan exists, it can find the plan from q_s to near q_g, progressing to q_g. □