Specifying User Preferences for Autonomous Robots through Interactive Learning

by

Nils Wilde

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2020

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Professor Anca Dragan Dept. of Electrical Engineering and Computer Science, University of California, Berkeley
Supervisor(s):	Professor Stephen L. Smith Dept. of Electrical and Computer Engineering, University of Waterloo
	Professor Dana Kulic Dept. of Electrical and Computer Engineering, University of Waterloo
Internal Member:	Professor Christopher Nielsen Dept. of Electrical and Computer Engineering, University of Waterloo
	Professor David Wang Dept. of Electrical and Computer Engineering, University of Waterloo
Internal-External Member:	Professor Steven Waslander Dept. of Mechanical and Mechatronics Engineering, University of Waterloo

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

This thesis studies a central problem in human-robot interaction (HRI): How can non-expert users specify complex behaviours for autonomous robots? A common technique for robot task specification that does not require expert knowledge is active preference learning. The desired behaviour of a robot is learned by iteratively presenting the user with alternative behaviours of the robot. The user then chooses the alternative they prefer. It is assumed that they make this decision based on an internal, hidden cost function. From the user's choice among the alternatives, the robot learns the hidden user cost function.

We use an interactive framework allowing users to create robot task specifications. The behaviour of an autonomous robot can be specified by defining constraints on allowable robot states and actions. For instance, for a mobile robot a user can define traffic rules such as roads, slow zones or areas of avoidance. These constraints form the user-specified terms of the cost function. However, inexperienced users might be oblivious to the impact such constraints have on the robot task performance. Employing an active preference learning framework we present users with the behaviour of the robot following their specification, i.e., the constraints, together with an alternative behaviour where some constraints might be violated. A user cost function trades-off the importance of constraints and the performance of the robot. From the user feedback, the robot learns about the importance of constraints, i.e., parameters in the cost function.

We first introduce an algorithm for specification revision that is based on a deterministic user model: We assume that the user always follows the proposed cost function. This allows for dividing the set of possible weights for the user constraints into infeasible and feasible weights whenever user feedback is obtained. In each iteration we present the path the user preferred previously again, together with an alternative path that is optimal for a weight that is feasible with respect to all previous iterations. This path is found with a local search, iterating over the feasible weights until a new path is found. As the number of paths is finite for any discrete motion planner, the algorithm is guaranteed to find the optimal solution within a finite number of iterations. Simulation results show that this approach is suitable to effectively revise user specifications within few iterations. The practicality of the framework is investigated in a user study. The algorithm is extended to learn about multiple tasks for the robot simultaneously, which allows for more realistic scenarios and another active learning component: The choice of task for which the user is presented with two alternative solutions. Through the study we show that nearly all users accept alternative solutions and thus obtain a revised specification through the learning process, leading to a substantial improvement in robot performance. Also, the users whose initial

specifications had the largest impact on performance benefit the most from the interactive learning.

Next, we weaken the assumptions about the user: In a probabilistic model we do not require the user to always follow our cost function. Based on the sensitivity of a motion planning problem, we show that different values in the user cost function, i.e., weights for the user constraints, do not necessarily lead to different robot behaviour. From the implied discretization of the space of possible parameters we derive an algorithm for efficiently learning a specification revision and demonstrate the performance and robustness in simulations. We build on the notion of sensitivity to derive an active preference learning technique based on maximum regret, i.e., the maximum error ratio over all possible solutions. We show that active preference learning based on regret substantially outperforms other state of the art approaches. Further, regret based preference learning can be used as an heuristic for both discrete and continuous state and action spaces.

An emerging technique for real-time motion planning are state lattice planners, based on a regular discrete set of robot states and pre-computed motions connecting the states, called motion primitives. We study how learning from demonstrations can be used to learn global preferences for robot movement, such as the trade-off between time and jerkiness of the motions. We show how to compute a user optimal set of motion primitives of given size, based on an estimate of the user preferences. We demonstrate that by learning about the motion primitives of a lattice planner, we can shape the robot's behaviour to follow the global user preferences while ensuring good computation time of the motion planner. Furthermore, we study how a robot can simultaneously learn about user preferences on both motions of a lattice planner and parts of the environment when a user is iteratively correcting the robot behaviour. In simulations we demonstrate that this approach is suitable to adapt to user preferences even when the features on the environment that a user considers are not given.

Acknowledgements

First of all, I would like to thank my supervisors Dana Kulić and Stephen L. Smith. They helped me with their great expertise and excellent guidance through my PhD and set truly inspiring examples for what makes a distinguished researcher and academic advisor.

I want to thank Anca Dragan, Christopher Nielsen, David Wang and Steven Waslander for serving on my PhD committee and engaging me in valuable discussions.

Moreover, I want to thank Ryan Gariepy from Clearpath Robotics who was involved in the research and always provided interesting insights into the real world challenges their robots are facing.

I would like to thank all the other great people in Dana's and Stephen's research groups. Journeying through our graduate studies together and learning about each others research does not only provide an interesting outlook into other fields, but also sparked many ideas for my own work. And eating cake together when someone got a paper accepted is just grand. Most notably, I would like to thank Alexandru Blidaru with whom I worked through all the challenges of conducting user studies, and Armin Sadeghi who is an excellent travel companion for conferences.

A very special thank goes to Alexander Botros, the best office mate, collaborator, band mate and just mate anyone can wish for.

Finally, I would like to thank my family, above all my parents and my brother, for their endless support and warmest welcomes when visiting home on the other side of the pond. My dearest thanks go to my wife Angelika who inspired me to embark on this endeavour and never stopped encouraging me, and my daughter Helena for always cheering me up.

Dedication

This thesis is dedicated to Helena and the Crimson King.

Table of Contents

List of Figures xiii

List of Tables xvi

List of Algorithms xvii

1 Introduction 1

1.1 Overview . . . 1

1.2 Contributions . . . 3

1.3 Structure . . . 7

2 Related Work 8

3 Problem Formulation 17

3.1 Preliminaries . . . 17

3.1.1 Graph Theory . . . 17

3.1.2 Spatio-Temporal Lattice Planner . . . 18

3.2 Problem Statement . . . 20

4 Deterministic Preference Learning Framework for Specification Revision 23

4.1 Problem Formulation . . . 24

4.2 Approach . . . 25

4.2.1 User Constraints and Cost Function . . . 25
4.2.2 Equivalence Regions . . . 27
4.2.3 Learning Framework . . . 27
4.2.4 Deterministic Learning Algorithm . . . 29
4.2.5 Vertex Search . . . 33
4.2.6 Minimal Vertex . . . 34
4.3 Evaluation . . . 35
4.4 Summary and Discussion . . . 39

5 Evaluation of the Deterministic Framework in a User Study 40
5.1 Overview . . . 40
5.2 Problem Description . . . 42
5.2.1 Learning User Preferences . . . 42
5.2.2 Active Learning Algorithm . . . 44
5.2.3 Multiple Tasks . . . 44
5.2.4 Metrics . . . 46
5.3 User Study . . . 48
5.3.1 Scenario . . . 48
5.3.2 Procedure . . . 48
5.3.3 Interface Design . . . 52
5.3.4 Hypotheses . . . 53
5.4 Results . . . 53
5.4.1 Participants . . . 53
5.4.2 Specifications . . . 54
5.4.3 Hypothesis 1 . . . 54
5.4.4 Hypothesis 2 . . . 58
5.4.5 Differences in the Population . . . 59
5.5 Discussion . . . 60

5.5.1 User Feedback . . . 61
5.5.2 Repeat and Novel Users . . . 62
5.5.3 Learning Framework . . . 62

6 Bayesian Preference Learning using Equivalence Regions 64
6.1 Problem Formulation . . . 65
6.2 Probabilistic Learning . . . 65
6.2.1 Bayesian Learning . . . 66
6.2.2 Probabilistic Learning Algorithm . . . 68
6.2.3 Greedy Policy . . . 70
6.3 Evaluation . . . 73
6.3.1 Performance of `MERR` and `MVR` Query Selections . . . 74
6.3.2 Robustness of `MERR` . . . 76
6.3.3 Extension to other Scenarios . . . 78
6.4 Discussion . . . 78

7 Maximum Regret Learning 81
7.1 Problem Statement . . . 82
7.2 Active Preference Learning Framework . . . 83
7.2.1 User Model . . . 84
7.2.2 Learning Framework . . . 84
7.2.3 Active Query Selection . . . 85
7.3 Min-Max Regret Learning . . . 86
7.3.1 Deterministic Regret . . . 87
7.3.2 Probabilistic Regret . . . 88
7.3.3 Preference Learning with Probabilistic Maximum Regret . . . 89
7.4 Evaluation . . . 89
7.4.1 Learning Error . . . 92

7.4.2 Easiness of Queries . . . 93
7.4.3 Generalization of the Error . . . 94
7.5 Discussion . . . 97

8 User Preferences on State lattices 99
8.1 Problem Statement . . . 101
8.2 Approach . . . 104
8.2.1 User model . . . 104
8.2.2 Estimation of the Loss Function . . . 105
8.2.3 Main Results . . . 105
8.2.4 Computational Complexity . . . 108
8.2.5 Computing an Optimal Control Set . . . 109
8.3 Evaluation . . . 109
8.3.1 Training Error . . . 109
8.3.2 Test Error . . . 112
8.4 Discussion . . . 113

9 Learning User Preferences from Corrections on State Lattices 114
9.1 Problem Formulation . . . 117
9.1.1 User Cost Function . . . 118
9.1.2 Problem Statement . . . 119
9.2 Approach . . . 119
9.2.1 User Model . . . 120
9.2.2 Learning from Corrections . . . 120
9.2.3 Updating Weights . . . 121
9.2.4 Completeness . . . 122
9.3 Evaluation . . . 123
9.4 Discussion . . . 126

10 Discussion and future directions **128**

10.1 Summary 128

10.2 Unified Framework 129

10.3 Sample-free Active Preference Learning 131

10.4 Other Future Work Directions 132

References **134**

APPENDICES **145**

A Sensitivity of Shortest Path Problems **146**

B Hardness of Maximum Regret Computation for Deterministic Motion Planners **153**

List of Figures

1.1 Interactive reward learning from human feedback. 2
1.2 Example: User specification and paths. 3
1.3 State lattice motion planners. 6

4.1 Representation of user constraints on a motion planning graph 26
4.2 Flowchart of the learning process 28
4.3 Example: Deterministic learning 30
4.4 Deterministic learning, initial iteration 32
4.5 Deterministic learning, final iteration iteration 32
4.6 Comparison of deterministic learning policies 37

5.1 Example user specification and revision. 41
5.2 Example for increase in task completion time. 46
5.3 Scenario and environment description from the user study. 49
5.4 Flowchart of the specification revision process. 50
5.5 Preference learning UI displaying duration and traffic rule violations for each path. Additionally, the violated traffic rules are also highlighted on the map. 52
5.6 Example specification from the user study. 55
5.7 Change in the task-dependent and global time ratio metric of the specification due to active learning. 57
5.8 Change in the task time ratio and entropy ratio metrics of the specifications due to active learning. 58

5.9 Change in the task-dependent time ratios of the specification, comparing novice and repeat users. 60

6.1 Experiment 1 of the MERR learning, comparing MERR and MVR for a cost dependent user model. 75

6.2 Experiment 2 of MERR learning, comparing MERR and MVR for different user accuracy. 76

6.3 Experiment 3: Robustness of the MERR user model for two specification (26 and 52 constraints) and different accuracies p in the simulated user and different estimates p' for the learning. 77

6.4 Additional MERR experiment using outdoor environment and PRM graph. . 80

7.1 Example of the sensitivity of a continuous motion planning problem. We show the histogram of the normalized weight error $\texttt{Err}_{\texttt{Weight}}$ and the normalized path error $\texttt{Err}_{\texttt{Path}}$ for 3000 uniformly sampled random weights. . . 86

7.2 'Driver' simulation scenario with different robot trajectories. 90

7.3 Comparison weight and path error in active preference learning with minimizing entropy or minimizing regret. 93

7.4 The likelihood that the simulated user gives the 'correct' answer, i.e., the probability in equation (7.14). 95

7.5 Relationship between training errors measured by the path and weight metric to test errors in the path metric. 96

8.1 Examples of a lattice control set and different trajectories. 100

8.2 Demonstrated and learned behaviour in the training environment. The color of the path indicates speed, where blue corresponds to slow and red corresponds to fast. 110

8.3 The t-error for all connections $\mathcal{B}$ and the MKSCS connections $\mathcal{E}$ compared to a naive approach with $\mathcal{B}_{\texttt{naive}}$ and $\mathcal{E}_{\texttt{naive}}$, respectively. 111

8.4 Training data: The t-error and planning time speedup for different sizes of the MKSCS and the different user types. 112

8.5 Test data: The t-error and planning time speedup for different sizes of the MKSCS and the different user types. 113

9.1 Example of the learning from correction framework, with and without generalization. . . . 116
9.2 Environment with environment and motion weights, optimal and learned from corrections. . . . 124
9.3 Learning from corrections with different values for λ. . . . 125
9.4 Learning from corrections, comparing a naive and the proposed approach. . . 126
9.5 Learning from corrections under different types of user. . . . 127

A.1 Sensitivity analysis: Trivial example graph. . . . 150

B.1 Graph for the reduction of the Subset Sum Problem to the Discrete Maximum Regret under Constraints Problem (DMRUC) in Theorem 6. . . . 155

List of Tables

4.1 Different types of user for Experiment 1. 36

4.2 Comparison of $\pi_{\texttt{vertexSearch}}$ and $\pi_{\texttt{minSearch}}$ 38

4.3 Results of the second experiment using $\pi_{\texttt{minSearch}}$. 38

5.1 Characteristics of the initial user specifications in the user study, showing the individual mean, median, min and max for each type of constraint and the number of traffic rules. 54

List of Algorithms

1 Active learning of constraint weights of user specifications. 31

2 Policy $\pi_{\texttt{vertexSearch}}$ for finding new weights using depth first search. 34

3 Heuristic policy $\pi_{\texttt{minSearch}}$ for finding minimal new weights 35

4 Active Selection of the new task for learning. 45

5 Probabilistic learning using equivalence regions with presampled paths. . . . 68

6 Maximum Regret Learning with presampled paths. 89

7 Learning from Corrections. 120

Chapter 1

Introduction

1.1 Overview

Autonomous robots are being deployed in a growing range of applications, increasing the number of settings where robots share the environment or even directly interact with humans. Despite advancements in robot autonomy and capability, specifying the robots' task often still requires a high level of expertise, which can hinder their acceptance in practice [103, 84, 104]. Consequently, a key challenge in the field of human–robot interaction (HRI) is enabling a broader range of users to supervise robots.

In this work, we study how non-expert users can efficiently deploy autonomous robots. While users often have a preference for how a robot should accomplish a task, it is difficult to directly inform a robot about these preferences. A new branch in HRI research – *reward learning* – aims to reduce the burden for the user when specifying robot behaviour [2, 87, 36, 51, 19, 27].

Broadly speaking, reward learning allows a robot to infer the behaviour a user would find optimal. In extension to classical approaches such as learning from demonstration, research in reward learning recently explored more interactive methods where users are iteratively presented with one or more candidate solutions and then provide feedback. Common methods for user feedback include correction, comparison and critique [51]. Usually, the decision making of a user is modelled to follow a cost function, which reduces the problem to one of learning the function's parameters. In HRI, user cost functions are often assumed to be linear, i.e., a weighted sum of features that describe a robot's behaviour [87, 37, 20, 78]. Figure 1.1 summarizes the framework: A robot is initially only provided

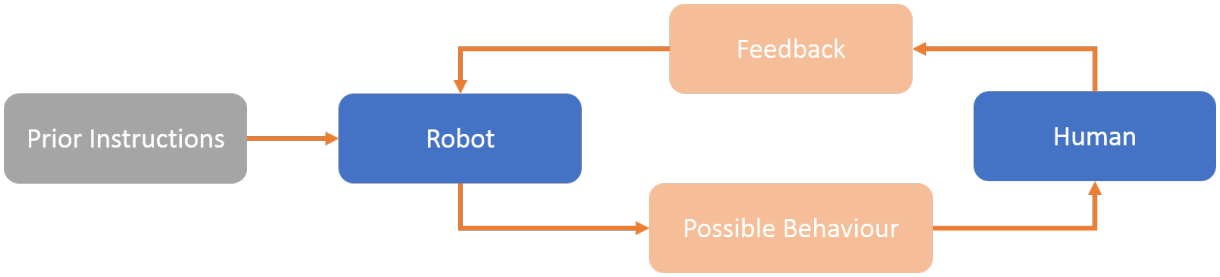


Figure 1.1: Interactive reward learning from human feedback.

with potentially incomplete or inaccurate instructions about its task. Over multiple iterations, the robot presents the user with possible solutions, based on the current belief about the user preferences. The user then provides feedback, allowing the robot to update its belief.

One reward robot learning framework that has found increasing attention is *active preference learning*. The robot iteratively presents the user with two or more possible trajectories and then asks the user to choose among them. This framework is particularly attractive since it minimizes the complexity of the user feedback: Users are not required to describe or demonstrate how they want the robot to behave – which can be highly challenging in some applications. Instead users simply indicate which of the presented solutions they like best.

In this thesis we study user preferences in the context of robot motion planning. We combine planning objectives such as minimizing length or time of a robot trajectory with user objectives in a cost function. Asking users to define such cost functions would potentially be challenging for them – therefore we investigate how a robot can reason about user cost functions using reward learning, with a strong focus on active preference learning. Throughout this thesis we examine the relationship between parameters of user cost functions, usually weights for trajectory features, and the set of possible robot trajectories. Due to the underlying sensitivity of motion planning problems (or more generally, optimization problems), different parameters do not necessarily lead to different trajectories. This induces a discretization of the parameter space which we exploit to design efficient algorithms based on *active preference learning*. We explore different learning frameworks, iteratively relaxing assumptions on user behaviour and allowing for more complex scenarios. As a running example we consider the problem of a revising user specifications, i.e, learning the importance of user-defined soft constraints on robot movements. We validate our work in several extensive simulations and demonstrate the practicality in a user study.

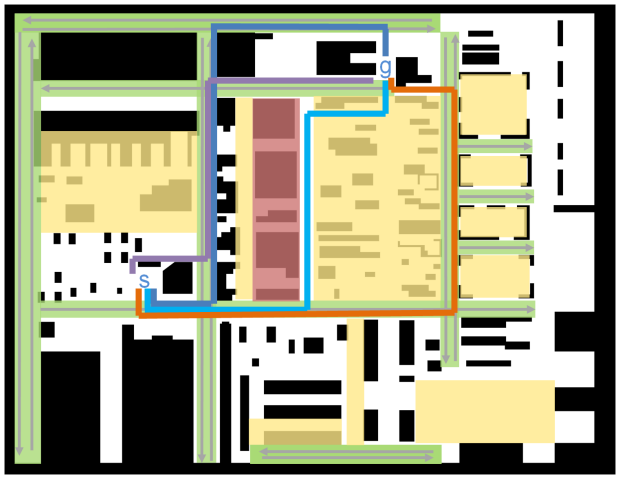


Figure 1.2: Example environment (white) with obstacles (black) and user defined constraints. Soft user constraints include roads (green), speed limit zones (yellow) and areas of avoidance (red). We compare different possible paths a robot could take, depending on under what circumstance the constraints must be followed.

1.2 Contributions

Specification revision We study *active preference learning* as a tool for helping users to create efficient robot task specifications. Here a specification is a user specific set of constraints on robot behaviour. For instance, in a warehouse or manufacturing environment where an autonomous robot performs material transport tasks, a facility operator might define traffic rules. This can include constraints such as areas of avoidance, one way roads or speed limits. Together with a defined set of start and goal locations, this yields a complete specification of a robot task. In practice, designing such rules can be challenging, as users might have little intuition about how their specification will affect the robot's behaviour and therefore the performance of the task. Thus, they might be willing to accept the violation of less important constraints given it is sufficiently beneficial for task performance. For instance, Figure 1.2 shows an industrial environment with several user defined constraints. The robot could take different routes between a fixed start and goal, depending on how a user values each constraint.

A user likely has a preference for which path is a better solution for the given task, based on the completion time and on the importance of the constraints. The latter can be captured via weights, numerically describing the importance of a constraint. However, asking the user to define these weights is an unintuitive and possibly challenging task. Therefore,

it would be easier for the user to only provide the spatial definitions of constraints, while the importance of each constraint is latent.

In Chapter 4 we propose a novel framework for specification revision using active preference learning. By asking the user to rank potential paths to be taken by the robot, the relative importance of each constraint can be learned. The initial user input is interpreted as a specification where all user constraints must be followed, if possible. In each iteration, the previously preferred path is shown again together with a new candidate path. From this interactive learning, a revised specification is obtained where the learned weights on the constraints express their importance, i.e., less important constraints might be violated if at least the learned required performance benefit is achieved. We capture this trade-off in a user cost function that combines the path traversal time with weighted features describing the violation of constraints by the path. Thus, in contrast to other work in reward learning [87, 46, 112], our features, i.e., the violation of constraints, are *user specific*. That is, by providing an initial specification the user identifies regions of the robot's environment that define features in the user cost function. During the interaction the user is assumed to provide feedback that always follows the assumed user cost function. The problem of learning the desired behaviour of the robot then becomes one of efficiently shrinking a hypothesis set – the set of paths the robot could take. As a method for active selection of new paths that are presented to the user we propose a heuristic: In each iteration we greedily show the user the alternative path from the current hypothesis set with minimal time, i.e., the path that maximizes the robot's performance. In simulations, we show that the framework allows for learning the weights for constraints within few iterations.

Furthermore, the practicality of this framework was investigated in a user study. Chapter 5 extends the learning framework to a multi-task scenario where a robot has to navigate between various start and goal locations. We show how the learned information can be combined and introduce another greedy strategy for choosing the task for which the user is presented with paths such that the expected improvement in performance is maximized. We then present the details of the study and illustrates how learning user weights helps users to improve the efficiency of their specifications significantly.

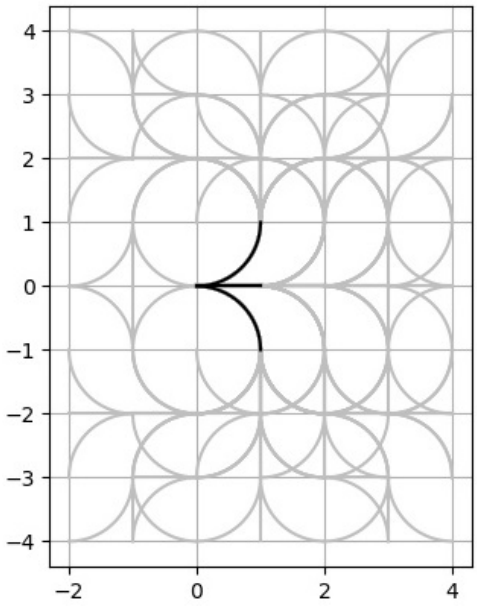
Active preference learning techniques In Chapter 6 we relax the assumption of the above approach about user feedback: Using a probabilistic model we account for users that do not always follow our assumed cost function. The motivation is to improve the robustness of the learning process since users might consider aspects that are not part of the cost function or simply be uncertain in their decision making and thus act noisily. The hypothesis set is replaced with a probability space, introducing the demand for a different

heuristic for the active path selection. We propose two novel approaches, both exploiting the sensitivity of optimization problems, i.e., the circumstance that different parameters do not necessarily yield different optimal solutions. The first approach discretizes the set of all weights appearing in the user cost function. We demonstrate how this technique allows for specification revision under the relaxed, probabilistic user model and compare the performance with another state of the art algorithm [87].

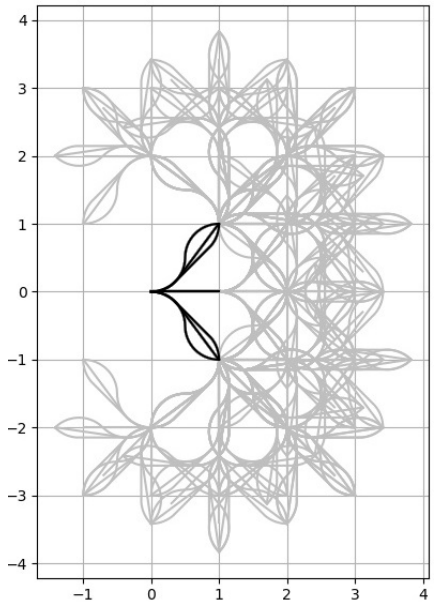
In Chapter 7 we present a second approach: Learning preferences using min-max *regret*. Let A be the solution for parameters $\boldsymbol{w}^A$ of some optimization problem. Regret then describes how sub-optimal A is under some other parameters $\boldsymbol{w}^B$. In preference learning, regret describes the error if the robot uses solution A while the user preferences are $\boldsymbol{w}^B$. We then always present users with solutions (A, B) that maximize the mutual regret, discounted by the learned probability distribution over weights. A similar concept of always learning about the current worst case has been applied to learning policies using inverse reinforcement [18]. We demonstrate how using min-max regret allows for highly efficient learning of user preferences, outperforming other state of the art techniques [20].

State Lattice Motion Planning An emerging new methodology for robot motion planning are state lattices [83]. The basic idea is to precompute a set of dynamically feasible trajectories between few regularly spaced robot states. A pair of states together with a connecting trajectory is called a motion primitive. The set of motion primitives is often referred to as the *control set*. Figure 1.3 shows examples for how a robot can navigate in an environment with two different sets of motion primitives. In Chapter 8 we study how a state lattice planner can represent user preferences on the robot's motions. In contrast to the specification revision problem where the preferences where on the environment, we consider preferences about the motion itself such as the trade-off between time and comfort for autonomous vehicles. While related research has focsed on learning a cost function for a fixed set of motion primitives [1, 37], we introduce a novel algorithm that computes optimal user specific control sets of a given size. That is, after learning an estimate of the user cost function, we compute a user specific discrete subset of the robot's action space which is then used for motion planning. In simulations we illustrated that the proposed approach is suitable to replicate user preferences learned from demonstrations, and that constraining the size of the control set allows for a substantial reduction in planning time.

Learning from Corrections In our framework for active preference learning for specification revision users have to choose among the presented alternatives for robot behaviours. However, if both alternatives are more or less equally undesired for the user, this framework



(a) Lattice with 3 motion primitives.



(b) Lattice with 7 motion primitives.

Figure 1.3: Two examples for state lattices used in robot motion planning. The Figure originally appeared in [15].

can become inefficient and the interaction might be frustrating for the user since they have to choose *the lesser of two evils*. In some cases users may be able to improve the robot behaviour by providing a more direct feedback. In our user study, described in Chapter 5, some participants stated that they would have found it helpful to either directly tell the robot what they did not like about a certain trajectory, or to *correct* the behaviour. In the specification revision for a mobile robot, a correction of the current behaviour could be a partial demonstration where the user draws a sub-path for the robot to take instead of a presented path, or the user defines via-points on a graphical interface.

In Chapter 9 we study how corrections can be used to learn about user preferences. We propose a framework where a user is iteratively presented with the robot's behaviour that is optimal for the current estimate of the user preferences. If that behaviour is not optimal, a user provides a correction which improves the behaviour, but is not necessarily optimal with respect to the user's underlying preferences. While providing corrections is potentially a more challenging task than choosing among preferences, we do not ask users for an initial specification. The corrections allow the robot to learn a specification that describes the

same optimal behaviour. Similar to problems in inverse reinforcement learning, the goal is to infer about the user's intentions and generalize the learned behaviour. In detail, when the user corrects a robot's path such that a certain area of the environment is avoided, the robot should not only avoid the swath from the originally presented path that was bypassed by the correction. Instead, the robot would ideally identify the entire part of the environment that should be avoided. Moreover, we use corrections to not only learn about a users preference of the environment, but also about global preferences about robot motions, like avoiding turns unless necessary.

Unified Framework for Specification Revision Finally, we outline a unified framework for specification revision. We extend the work from our user study (Chapter 5) in three directions: 1) We employ probabilistic user feedback to increase the robustness of the learning. 2) To generate more realistic robot motions we use a spatio-temporal state lattice motion planner with four dimensions: (x, y) position, eight headings and discrete speed values. 3) Similar to the our earlier work, users are still iteratively presented with pairs of paths and asked to choose among them. However, in addition to choosing, user can instead a) provide a correction to one of the presented paths, or b) change their initial specification.

1.3 Structure

This thesis is structured as follows: In the second chapter we review the relevant literature. Chapter 3 presents preliminaries and formally defines the problem of learning user preferences the behaviour of robots as well as our running example of revising user specifications. Chapters 4 and 5 present an approach for specification revision based on a linear and deterministic user model, together with its evaluation in simulations and in a user study. We generalize the learning framework using a Bayesian model and present improved techniques for the active selection of the paths that are presented to the user in Chapters 6 and 7. In Chapter 8 we summarize our work on learning user specific control sets for state lattice motion planners. As an alternative mode of user feedback, we study how a robot can learn user preferences from corrections to the current behaviour in Chapter 9.

We conclude in Chapter 10 with a brief summary of the presented work and an outline of a unified framework for the specification revision problem with more diverse forms of user feedback. Further, we discuss current and future directions of research.

Chapter 2

Related Work

We study how non-expert users can efficiently instruct autonomous robots. Our work relates to various branches of research in human-robot interaction (HRI), including reinforcement learning (RL)[96, 56], learning from demonstration (LfD)[7], inverse reinforcement learning (IRL)[2], learning from corrections [47], and active preference learning [17, 87]. Throughout our work we revisit methodologies for robot motion planning [60] and borrow algorithms and tools from convex optimization problems and their sensitivity analysis [10, 58] as well as statistical inference [105].

In the field of HRI, teaching robots how to perform certain tasks has been studied extensively over the last decades [103, 12, 90]. Robot behaviour specification can be grouped into three types of approaches. The "classic" approach involves an expert operator [12, 7]. This includes providing optimal demonstrations, designing a high level specification for tasks and allowable movements or defining a reward function for autonomous robots. The second approach takes into consideration that demonstrations and specifications – especially when provided prior to the robot executing the task – might be sub-optimal [59, 102]. Therefore, initially learned robot behaviour is revised or refined to find a better solution. More recently, research increasingly focuses on defining robot behaviour using interactive reward learning frameworks [28, 87, 61, 24]. In these interactive approaches users are presented with the tentative robot behaviour and provide feedback such as providing additional demonstrations or choosing among alternative solutions, allowing the autonomous robot to learn about their preferences and iteratively improves its strategy. Often such frameworks employ *active* learning, i.e., the robot can choose how to query the user (i.e., which alternative solutions are shown or for what part of a task more demonstrations are required).

Expert Users Arguably the two primary techniques for defining robot tasks are programming by demonstration and offline programming [25]. In the majority of industrial applications these techniques require users with special training. However, developments in computational abilities increasingly allow for complex demonstration methods such as teleoperation, kinesthetic teaching (haptic manipulation of the robot), the design of trajectories on graphical user interfaces (GUI), natural language, visual cues or demonstrations in augmented and virtual realities [12, 103]. Simultaneously, robots are desired to plan and act with an increasing level of autonomy. This reduces the need for explicitly programming single actions in the configuration or task space and allows for high-level task definitions.

A common way of specifying the high-level behaviour for autonomous systems is to specify a reward function and then learn the appropriate policy using reinforcement learning (RL) [56]. In RL an agent, e.g., a robot, acts autonomously in an environment, often described by a Markov Decision Process (MDP) [65]. A user defined reward function maps the states in the MDP to a numerical value, expressing how desired that state is. This reward function corresponds to a high-level specification for how the robot should behave; through RL the robot then finds a policy that maximizes the reward. RL has been extensively studied as a tool to realize a high level description of a robot's behaviour [56]. For instance, [91] and [95] apply RL in the domain of mobile robots and robots competing in soccer games. In both examples the reward function is designed by a human expert. The practicality of RL approaches has also been investigated in field studies [55]. However, specifying reward functions usually requires a high level of expertise and can be unintuitive . Moreover, the choice of a reward function has a large impact on the resulting policy [42, 80].

Alternatively to specifying reward functions, a robot can be taught by demonstrating the desired behaviour, similar to classic approaches such as teach-in. The field of learning from demonstration (LfD), investigates various kinds of demonstrations for robot programming [12, 7]. Its application range from high-level task specification [30] to the definition of precise actions such as grasping [64] or manipulation trajectories [3]. Usually LfD is distinguished into two categories: Imitation learning and apprenticeship learning. While the former approach tries to directly replicate the demonstrated behaviour, the goal apprenticeship learning is to learn the *intentions of the demonstrator*. The authors of [2] laid the foundation for apprenticeship learning using *inverse reinforcement learning* (IRL). When providing a demonstration the human demonstrator is assumed to optimize an internal reward function, which is unknown to the robot. The robot then tries recover the reward function from the demonstrations allowing it to compute a corresponding optimal policy. Thereby, IRL leverages the capabilities of autonomous robots since a once learned reward

function potentially allows for finding good policies even for different scenarios. In the field of control this problem is also known as inverse optimal control [52]. In IRL reward functions are often modelled as a linear combination of pre-defined features; the problem then consists of learning the weights for all features [19, 2, 116].

A different commonly used approach for defining complex robot tasks is using a specification language such as linear temporal logic (LTL) [11]. LTL allows for abstract specifications including a temporal ordering of robot states, for instance *"First, visit region A and B, then go to C and finally vist D"*, can be expressed easily. As LTL is a mathematical formalism, intuitive concepts for defining complex LTL formulas are desired. In [93] a GUI is designed for LTL based mission planning, while [32] proposes a framework for using natural language to provide LTL specifications. Finally, [88] proposes a framework for learning an LTL formula from demonstrations.

Similar to the IRL problem, we want to learn a user's cost function which is a weighted sum of features of a robot's trajectory. In case of our specification revision problem (Problem 3) some features describe the violation of constraints. Then, learning about the importance of constraints is analogous to recovering a user reward function based on these features. Our work on learning a control set for a lattice planner that represents user preferences, presented in Chapter 8, is based on learning from demonstrations. Given a set of demonstrations we find an estimate of the user preferences, for which we then find an optimal control set. In Chapter 9 we consider users who iteratively correct the current behaviour. Similarly to IRL seek to learn about the user's intention behind their correction, allowing the robot to generalize the information learned from a correction.

High-level specifications with uncertainty Naturally, when considering specifications obtained from experts, the following question arises: What if the expert's specifications do not correctly reflect their intentions? A user might be oblivious to the impact of their high-level specification on the performance of the robot task, misjudge the feasibility of their specification, their priorities might change or they simply make errors when providing the specification. In the field of LTL, specification revision is studied when the initial specification leads to suboptimal outcomes [59] or is infeasible [102, 31, 53]. In [59] a specification is viewed as a set of soft constraints on a robot's movement, while the environment is modelled as an MDP. Strictly following the specification might lead to a sequence of actions with a low probability of success. An optimal trade-off between closeness to the original specification and probability of success is then found in a multi-objective optimization problem. For the application of mobility on demand, [102] and [53] consider

soft and hard constraints of an LTL formula together with deadlines for certain tasks. In order to fulfil a deadline, the violation of soft constraints might be acceptable. A minimum violation trajectory is found such that the deadline is met and all hard constraints are still satisfied. In a general motion planning problem on some configuration space with spatial obstacles, [43] considers the case when no feasible path exists. The minimal constraint removal problem then finds the biggest subset of obstacles such that a feasibility is re-attained.
Revisions of an initial specification provided of an expert are also studied in LfD [88, 38, 74, 19]. Initial demonstrations can be suboptimal or might be insufficient as a robot might encounter unforeseen contingencies during execution. The work of [74] automatically segments the tasks and then efficiently asks for additional demonstrations when needed. The authors of [19] seek to achieve better than demonstrator performance by automatically ranking demonstrations. Moreover, [38] focuses on failed demonstrations. Instead of imitating the human, the learning system tries to avoid repeating the mistakes the operator made.

In a comparable fashion, our specification revision problem (Problem 3) receives a set of constraints from a user. We obtain an initial specification by setting the weights for all constraints to a large value such that the resulting path respects all user constraints. However, we assume that such a path might not necessarily be optimal with respect to the user as some constraint violations might be allowable. The user agreeing to the violation of a constraint corresponds to removing or relaxing the constraint in question, which then leads to a revised specification. Eventually, the proposed procedure finds the optimal trade-off between increasing performance and closeness to the original specification.

Interactive reward learning While achieving excellent results in certain applications, the previously mentioned approaches have various drawbacks. For instance, asking a user for demonstrations is not always desirable, as they might not be applicable [4], difficult to provide [28, 2], the amount of necessary demonstrations may be prohibitively large [24] or the demonstration itself requires a high level of expertise [112, 28]. Providing rich and precise specifications prior to a robot executing a task might also be challenging and more prone to inaccuracies [2]. A key concept to further improve intuitive HRI methods is interactiveness. Instead of asking the user for a complete specification in the beginning or demanding numerous demonstrations, robot tasks can be learned in an iterative, interactive way. This new branch in HRI research is referred to as *reward learning* or *interactive robot learning*.

A popular framework is *active preference learning* where the robot presents the user

with candidate solutions and then learns about the user preferences from their choice among the candidates [46, 87, 9, 20, 24, 112, 4, 45]. This approach is based on preference elicitation [40, 35, 22] where the problem input consists of a set of hypotheses, tests and possible outcomes. By performing tests, some hypotheses become inconsistent with the observed outcome and are rejected. This can be applied to a robot learning: Hypotheses are possible reward functions of the user. Tests correspond to presenting the user with alternative solutions based on these reward functions, while outcomes are the user's selections. The user's internal reward function is then learned by iteratively ruling out reward functions that become inconsistent with the user's choices. Active learning allows the learner to decide what query, i.e., what set of alternative solutions, the user is presented with next. The authors of [87] present a pioneering, application-independent framework for learning reward functions from a user choosing between alternative trajectories. The user is assumed to provide feedback following a probabilistic model, based on a Boltzman model. Subsequent work introduced a tuning parameter for the user uncertainty [9], and compared different approaches for query generation [20]. The authors of [28] apply active preference learning to grasping tasks where experts rank a robot's demonstrations; in [24] active preference learning is combined with RL.

Throughout Chapter 4 to 6 we apply active preference learning to revise user specifications (Problem 3). After a user inputs a set of constraints, the robot queries them about alternative paths to learn about the importance of the user constraints from their feedback. In contrast to preference elicitation problems, the set of hypothesises, i.e., the set of all paths the robot could take, is *not* a problem input. When planning on a graph, finding the set of all paths from start to goal is computationally intractable since it is a #P-complete problem [101]. Other work in the field of active preference learning often considers a predefined set of features that considered in the user cost function [87, 20, 112]. In contrast, the features in the specification revision problem correspond to user defined constraints and thus are user specific. This can result in high dimensional features and weight spaces. However, these user specific features come with implicit prior information about the user's preferences: We assume users follow two objectives: Minimizing time, and only allowing constraint violation when sufficiently beneficial. This helps us designing strategies for presenting alternative paths such as greedily maximizing the potential improvement in performance, i.e., path time.

In Chapter 7 we study active preference learning independent of the application, e.g., the specification revision problem. In contrast to the work of [87, 9, 20] we pose the problem as one of finding a *path* that is similar to the optimal path – as stated in Problem 2 – instead of finding weights for the user cost function that are as similar to the optimal weights. The proposed path error is related to the spanning error of minimal control sets

for state-lattice planners [15, 83]; a comparable error function has also been used in risk-aware IRL [18]. We show that the path error metric is more robust than the weight error and allows for a better prediction of test cases. Moreover, we introduce a method for query generation that greedily minimizes the maximum path error and compare this approach to the query selection proposed in [20].

An interactive variation of LfD that potentially reduces the complexity of the user input is learning from corrections (LfC) [47, 115, 66, 41]. Instead of asking the user to demonstrate the complete task, the robot presents its current guess about optimally solving the task to the user. They can then *correct* the presented behaviour, i.e., change only parts of a robot's trajectory. This can be done by either physical interaction with the robot or by using a graphical interface to draw parts of a trajectory or set up via points. Learning from corrections was proposed as an alternative to IRL in [47], aiming to reduce the burden on the user by not asking for optimal demonstrations, but only to iteratively improve the current behaviour of a manipulator robot. Recent work in LfC investigates how to extrapolate the information obtained from a correction to learn about the user cost function efficiently [115]. The authors of [66] introduce a Kalman Filter for LfC to express uncertainty for the learned preferences. In [41] corrective demonstrations are used to learn about task models described by a finite-state automaton.

In Chapter 9 we propose an LfC framework as an alternative approach for solving the specification revision problem: We do not require the user to define any constraints initially, but iteratively show them one path for a specific task to which they provide a correction if the path is suboptimal. From that correction we can infer constraints on the robot's environment, allowing us to learn a specification. Thus, the increased complexity of the user feedback is trade-off with omitting the need for the user to define constraints. In the unified framework in Section 10.2 we combine active preference learning with learning from corrections to enable users to choose the form of feedback that they provide.

User centered learning A common challenge in many learning problems is balancing between exploration and exploitation. This problem also arises when learning user preferences for robot behaviour. Generally, learning systems are designed to find optimal solutions as fast as possible; an active learner usually tries to minimize the number of queries. This is extensively studied in the field of preference elicitation, namely when constructing an optimal decision tree [22]. Often a learning agent is maximally focused on exploring. In an HRI scenario however, this behaviour might not always lead to a desired outcome. It can be frustrating for a user if the learning system repetitively presents

solutions that diverge drastically from the ones the user previously preferred, seemingly ignoring the feedback given so far. Therefore, exploiting the knowledge about the user's preferences might be important to gain acceptance of the user. Furthermore, the authors of [20] consider the easiness of queries, i.e., the robot presents alternatives where the user potentially has a strong preference. Similarly, [85] proposes a strategy for active learning that consider the flow of the queries to reduce the mental effort of the user and thus decrease the user's error rate. More generally, [21] studies what makes good questions from both - a machine learning and human learning perspective.

We address the user-friendliness of queries when learning specifications (Chapter 4 to 6) by always presenting the user with a pair of paths where one path is the preferred option from the previous query. This creates a workflow where the current solution, i.e., the one that was previously preferred, evolves iteratively. Furthermore, the approach implies a balance between *exploration and exploitation* [73] since each query gains more information about the current solution but also shows a path that was chosen actively and thus explores the solution space. A similar approach where the previously preferred solution constitutes one of the options in the next iteration is proposed in [77], where active preference learning is combined with LfD. In Chapter 7 we propose a query selection method that presents pairs of paths that are maximally different in cost and show in simulations that user can choose between these paths easily and accurately, outperforming the approach from [20].

Applications The specification revision problem is motivated by scenarios where autonomous mobile robots perform transportation tasks in industrial facilities [76]. As in any shared environment, robots are required to adjust to human behaviour. Recent research in this domain investigates predictions of human movement [67, 106, 98] to ensure safe, stable and yet performative workflows. A human aware robot behaviour is also of interest in non-industrial settings to ensure social acceptance of autonomous systems [57]. Our work looks as these issues from the opposite perspective: How can a user specify a robot to adapt to existing work flows and act in a way that is natural to the user?

Adapting to user preferences is also of interest in various other applications. For instance, in [79] human preferences are learned to allow personalized generation of routes in navigation systems. Thereby, paths that are optimal with respect to multiple objectives including time and various route-related attributes are found. Each user might have individual preferences on how to trade-off between these objectives, which can be learned by observing the user's choices between alternative route options. A co-active learning approach has also been applied to manipulator robots [48]. Users are not required to provide

optimal demonstrations; a robot's trajectories are improved iteratively through interaction. A similar approach has been applied to marine robot missions [92], where human experts propose an initial trajectory. A co-active learning system then proposes alternative solutions based on additional environmental information. From these alternatives the user then makes a final selection. A hidden cost function for the human preferences is learned to then find trajectories that match the operator's objective.

In order to enable practical use of intuitive task specification methods, the design of graphical user interfaces is studied in [89, 93]. Closely related to the specification learning problem, [13] presents a GUI for specifying spatial constraints on a known environment. In Chapter 5 we use an evolution of this GUI: In addition to enabling users to draw constraints, the revised GUI can show alternative paths and let the user choose between them. Especially in applications with safety concerns, a presentation on an interface is preferable to having the actual robot performing a trajectory that the user may not approve.

Finally, adapting to user preferences is actively researcher in autonomous driving. The objective is that an automated vehicle cannot only navigate safely and drive the shortest route, but also considers passenger comfort. The most common problem is learning a human driving style from demonstrations [1, 37, 39]. The work of [23] proposes a set of features to identify different driving behaviours of human drivers. In Chapter 8 we present an algorithm for computing an optimal control set for state lattices that represents user preferences learned from demonstrations. While [1, 37] estimate user preference and then plan optimal paths using a *fixed* state lattice planner, our work computes a user specific lattice planner, based on an estimate of the user preferences given a set of demonstrations.

Summary: Specification learning The proposed specification learning problem combines several of the discussed approaches. Specifying constraints on a known environment is similar to a high level task definition provided by an expert user. However, we assume that the user would accept the violation of constraints for sufficient time benefit. This makes the provided specification incomplete: The user does not provide information about under what circumstances constraint violation is accepted. In terms of RL, the reward function is left undefined, i.e., the weights of constraints are unknown. On the other hand, by asking only for this truncated expert specification, the burden on the user is reduced. Moreover, this also allows for the user specification to be uncertain to some extent: If a user initially misjudges the impact of a constraint, they might allow a robot to violate it in some tasks. This is equivalent to revising the specification by removing parts of it.
To fill the gap between incomplete or uncertain user input and a complete task specification, we apply active learning. Applying LfD to our scenario would require the user to

design entire paths. Similar to asking for complete specifications, this can be challenging and might lead to suboptimal outcomes. Active preference learning allows us to present the user with different paths and ask them for a ranking. This is not just less arduous, but comparative feedback potentially yields more information about the user cost weights. We assume that the user evaluates the quality of paths based on a cost function in the form of a weighted sum of features. As features we consider a path's time and the constraints it violates. This is potentially beneficial in comparison to other research on active preference learning: Features are often picked manually and are therefore a design choice for the learning system. In our case, the violation of constraints follows from the user specification and thus the features are user specific.

Chapter 3

Problem Formulation

3.1 Preliminaries

Notation We denote vectors with bold, lower case letters $\boldsymbol{v}$. An element of a vector is indicated with a subscript index v_i, while a superscript index $\boldsymbol{v}^i$ identifies some specific vector. Upper case letters denote a set (G), bold upper case letters represent matrices ($\boldsymbol{A}$).

3.1.1 Graph Theory

Many motion planning algorithms in robotics discretize the robot's state space $\mathcal{X}$ into a finite set and then compute dynamically feasible connections between these state [60, 50]. Then, the problem of finding a trajectory between some start and goal state is approximated by solving a discrete optimization problem on the *graph* that is induced by the set of states and the connecting trajectories.

Following [58], a graph is an ordered pair $G = (V, E)$, where V is a set of vertices and E is a set of edges. Multi-graphs are described as a triple $G = (V, E, \Psi)$, the function $\Psi : E \to \{(v, u) \in V \times V : v \neq u\}$ associates each edge with an ordered pair of vertices. Multi-graphs can feature parallel edges; two edges are called parallel if they connect the same ordered pair of vertices. In a weighted graph $G = (V, E, c)$ a real value function associates weights to each edge of the graph: $c : E \to \mathbb{R}$. Doubly weighted graphs have two independent weight functions c_1, c_2 for all edges. We write a weighted multi-graph as $G = (V, E, \Psi, c)$ and a doubly weighted graph as $G = (V, E, \Psi, c_1, c_2)$.

A walk between two vertices v_1 and v_{k+1} on a graph G is a finite sequence of vertices and edges $v_1, e_1, v_2, e_2, \ldots, e_k, v_{k+1}$ where $e_1, e_2, \ldots e_k$ are distinct. A path P between two vertices v_1 and v_{k+1} is defined as a graph $(\{v_1, v_2, \ldots, v_{k+1}\}, \{e_1, e_2, \ldots, e_k\})$ where $v_1, e_1, v_2, e_2, \ldots, e_k, v_{k+1}$ is a walk. On a weighted graph, the cost of a path is defined as $c(P) = \sum_{e \in P} c(e)$. In doubly weighted graphs we can define two costs c_1 and c_2 where $c_1(P) = \sum c_1(e),\ e \in P,\ c_2(P) = \sum c_2(e),\ e \in P$.

In robotics, planning the shortest or fastest trajectory between two given states can be approximated with a shortest path problem on the respective graph.

Problem 1 (Shortest path on a graph)**.** Given a weighted graph $G = (V, E, c)$ and a start (s) and goal (g) vertex on G, find a path P where the cost $c(P) = \sum_{e \in P} c(e)$ is minimum.

In this work, we consider cost functions c that do not only represent a physical property such as length or time, but rather cost functions that also include user preferences. The computation time of shortest path problems can be enhanced greatly by using heuristic approaches such as the A*-algorithm instead of Dijkstra's-algorithm [60]. This requires an *admissible* heuristic, i.e., a function h that always underestimates the cost of a path between two given vertices. One of the most common heuristics is the euclidean distance. However, when the cost function on the graph c represents user preferences, finding an admissible heuristic becomes difficult: In case the cost of an edge can become 0, the only admissible heuristic will always return 0, effectively rendering A* to Dijkstra's. Therefore, throughout this work we rely on Dijkstra's-algorithm to solve shortest path problems.

3.1.2 Spatio-Temporal Lattice Planner

Some of the most well known and widely used motion planning algorithms are *probabilistic motion planners* such as Probabilistic Roadmap (PRM), Rapidly Exploring Random Trees (RRT, RRT*) and Fast Marching Trees (FMT) [60, 50]. In these techniques, robot states are sampled randomly and a local planner then finds connection between sampled states. An alternative approach is a state lattice planner [83]. The basic idea is to choose the discrete set of robot states such that the states are regularly spaced and then precompute a set of dynamically feasible trajectories between few robot states. Figure 1.3 shows two examples for lattice planners for a three-dimensional robot state, consisting of (x, y) position and heading.

We follow [83] for a formal definition. For a more fundamental description of lattice using group theory, the interested reader might be referred to [15]. Given the state space

of a mobile robot $\mathcal{X}$, let $V \subset \mathcal{X}$ denote a regularly spaced, finite subset of robot states, also called lattice states, and let $s \in V$ denote an arbitrary starting state. Further, $\mathcal{B} = \{(s, j) : j \in V\}$ is the set of tuples of s and all vertices $j \in V$. We call any $\mathcal{E} \subseteq \mathcal{B}$ a *connection set*. Each tuple can be thought of as a start-goal pair of states in the configuration space. Let T denote a set containing a unique trajectory from s to j for each $(s, j) \in \mathcal{B}$. The key idea is that connections in $\mathcal{B}$ – together with their associated trajectories – can also be applied on other vertices $j \in V$ to reach another state i in $\mathcal{X}$. By pre-computing trajectories from s to j for all $j \in V$, we can construct trajectories from s to i via successive valid concatenations [15] of the trajectories in T.

Given a set of connections $\mathcal{E} \subseteq \mathcal{B}$ the tuple $(\mathcal{E}, T)$ is called a *control set* of the lattice. For any $b = (s, j) \in \mathcal{E}$, and trajectory $T_b \in T$ from s to j, we call the tuple $(b, T_b) \in \mathcal{E} \times T$ a *motion primitive*. For a given control set $(\mathcal{E}, T)$ with $\mathcal{E} \subseteq \mathcal{B}$, we may define a weighted, directed graph $G_T^{\mathcal{E}} = (V, E, c)$. The edges E are all those pairs (i, j) such that there exists a connection $b = (s, k) \in \mathcal{E}$ that takes i to j. The cost c of the edge (i, j) is the cost of the trajectory from s to k. This cost is assumed to be an *almost-metric* (that is, it satisfies all properties of a metric except for the symmetry property). The weighted, directed graph $G_T^{\mathcal{E}}$ is called a *state lattice*. Similar to $G_T^{\mathcal{E}}$ we can define a graph $G_T^{\mathcal{B}} = (V, B, c)$ where B are all edges defined by connections $b \in \mathcal{B}$.

Minimal Size Connection Sets A key challenge when designing a lattice planner is choosing the size of the connection set. Using a larger set motion primitives increases the fidelity of planner, since it has to rely on fewer concatenations; using a smaller control set improves the planning time since the robot has fever controls available at each state, i.e., the branching factor of the search problem is smaller.

The problem of finding optimal connection sets with an upper bound on the error has been studied in [83, 15]. Given $G_T^{\mathcal{E}} = (V, E, c)$ let $c_j^{\mathcal{E}}$ be the cost of the minimal-cost path from s to vertex j in $G_T^{\mathcal{E}}$. This definition together with the assumption that c is an almost metric (and therefore satisfies the triangle inequality) implies that $c_j^{\mathcal{B}}$ is the cost of the direct trajectory s to j, i.e., the trajectory computed without concatenation. To evaluate the quality of a control set $\mathcal{E}$, we use the t-error of defined here:

Definition 1 (t-error)**.** Given a lattice graph $G_T^{\mathcal{B}} = (V, B, c)$, the *$t$-error* of a subset $\mathcal{E} \subseteq \mathcal{B}$ is given by

$$\tau(\mathcal{E}) = \max_{j \in V} \; c_j^{\mathcal{E}} / c_j^{\mathcal{B}}. \tag{3.1}$$

For each $j \in V$, the value $c_j^{\mathcal{E}} / c_j^{\mathcal{B}}$ represents the ratio of the cost-minimal path using connections only in $\mathcal{E}$ to the cost of the direct trajectory from s to j.

The *Minimum t-Spanning Control Set Problem* (MTSCSP) is formulated as follows: Given a state lattice $G_T^{\mathcal{B}}$ and some real number $t \geq 1$, compute a set $\mathcal{E} \subseteq \mathcal{B}$ of minimal size such that $\tau(\mathcal{E}) \leq t$. This problem is NP-complete [15].

3.2 Problem Statement

General Problem We consider a robot acting in a known environment with state and action spaces $\mathcal{X}$ and $\mathcal{A}$. Further, let $\mathcal{Z}$ be a set of tasks where each task describes a robot's initial and goal state s and g in $\mathcal{X}$.

Moreover, we consider a user with an internal cost function $\mathcal{F}$ mapping from a trajectory $\mathcal{T}$ to a real number and a motion planner that computes trajectories between a given pair (s, g) that are optimal with respect to $\mathcal{F}$. However, we consider this cost function to be *hidden*, i.e., we do not have direct access to it. For every task $(s_i, g_i) \in \mathcal{Z}$ we are interested in finding the optimal trajectory $\mathcal{T}^*$ with respect to the user's cost function:

$$\mathcal{T}^* = \arg\min_{\mathcal{T}} \{\mathcal{F}(\mathcal{T})\} \tag{3.2}$$

We can learn about the cost function $\mathcal{F}$ by presenting the user with alternative trajectories. For a given set $\{\mathcal{T}^1, \mathcal{T}^2, \dots\}$, we obtain a user feedback in the form of a partial ordering of the trajectories according to the user's preferences. We can write that as a set of pairwise relations $\mathcal{U}^{ij}$ indicating if $\mathcal{F}(\mathcal{T}^i) > \mathcal{F}(\mathcal{T}^j)$, $\mathcal{F}(\mathcal{T}^i) < \mathcal{F}(\mathcal{T}^j)$ or $\mathcal{F}(\mathcal{T}^i) = \mathcal{F}(\mathcal{T}^j)$ for any pair $(\mathcal{T}^i, \mathcal{T}_j)$.

We model $\mathcal{F}(\mathcal{T})$ to be a weighted sum of scalar features $\phi_1(\mathcal{T}), \phi_2(\mathcal{T}), \dots, \phi_n(\mathcal{T})$ of the trajectory. Thus, the cost function is of the form

$$\mathcal{F}(\mathcal{T}) \;=\; \sum \phi_i(\mathcal{T}) w_i \;=\; \boldsymbol{\phi}(\mathcal{T})\boldsymbol{w}, \tag{3.3}$$

where the row vector $\boldsymbol{\phi}(\mathcal{T}) = \begin{bmatrix}\phi_1(\mathcal{T}) & \phi_2(\mathcal{T}) & \dots & \phi_n(\mathcal{T})\end{bmatrix}$ summarizes all features of $\mathcal{T}$ and the column vector $\boldsymbol{w} = \begin{bmatrix}w_1 & w_2 & \dots & w_n\end{bmatrix}^T$ contains weights for the features. The problem of learning $\mathcal{F}$ in order to find the optimal solution to (3.3) then consists of learning the *user optimal weights* $\boldsymbol{w}^{\texttt{user}}$. We notice that it might not be possible to learn $\boldsymbol{w}^{\texttt{user}}$ exactly. Therefore, let $\mathcal{T}_i'$ be the trajectory that minimizes $\boldsymbol{\phi}(\mathcal{T})\boldsymbol{w}'$ for any given $\boldsymbol{w}'$ and some task (s_i, g_i). The objective is to find a $\boldsymbol{w}'$ such that the relative error in cost between $\mathcal{T}_i'$ and $\mathcal{T}_i^{\texttt{user}}$ is minimized. This allows us to formally state the problem:

Problem 2 (Active Preference Learning)**.** Given is a robot's state and action space $(\mathcal{X}, \mathcal{A})$, a set of tasks $\mathcal{Z} = \{(s_1, g_1), (s_2, g_2), \dots\}$, a set of feature functions $\phi_1, \phi_2, \dots, \phi_n$, and a user with hidden user preferences $\boldsymbol{w}^{\texttt{user}}$. For up to k iterations, the user can be queried about an ordering of a set of trajectories $\{\mathcal{T}^1, \mathcal{T}^2, \dots\}$. Find an estimate $\boldsymbol{w}'$ of the user weights $\boldsymbol{w}^{\texttt{user}}$ such that the corresponding optimal trajectories $\mathcal{T}_i'$ for all tasks (s_i, g_i) minimize

$$\texttt{Err}(\boldsymbol{w}', \boldsymbol{w}^{\texttt{user}}) = \sum_{(s_i, g_i) \in \mathcal{Z}} \frac{\mathcal{F}(\mathcal{T}_i')}{\mathcal{F}(\mathcal{T}_i^{\texttt{user}})}. \tag{3.4}$$

Specification Revision Problem The problem of revising a specification is a variation of Problem 2. The problem input additionally includes a specification $\Gamma = \{\gamma_1, \gamma_2, \dots, \gamma_d\}$. Each constraint γ_i is a triplet $(\mathcal{X}_i, \mathcal{A}_i, w_i^{\texttt{user}})$ where $\mathcal{X}_i \subseteq \mathcal{X}$, $\mathcal{A}_i \subseteq \mathcal{A}$ and $w_i^{\texttt{user}} \in \mathbb{R}$ is a weight. That is, a constraint associates a weight $w_i^{\texttt{user}}$ with a set of actions $\mathcal{A}_i$, if these actions are taken at states $\mathcal{X}_i$. Hence, $\gamma_1, \gamma_2, \dots, \gamma_d$ are potentially soft constraints; the weights describe rewards or penalties for actions that are encouraged or discouraged by the user. Furthermore, we assume that the weights $w_i^{\texttt{user}}$ are *hidden*, i.e., the user only specifies states and associated actions $\mathcal{X}_i$ and $\mathcal{A}_i$, but not how desirable it is that the robot follows these constraints. This reduces the complexity of the user input to enable non-experts to create robot specifications: Choosing weights for constraints potentially requires a high level of expertise and thus in current industrial practice such constraints are designed by trained personnel [76]. We pose our problem as one where the weights $w_i^{\texttt{user}}$ for the constraints have to be learned from user interaction. Thus, the robot gains information about which constraints have to be followed at all times, i.e., are hard constraints, and which ones can be violated, i.e., are soft constraints, and when a violation of these soft constraints would be acceptable. We then have a user cost function $\mathcal{F}_\Gamma(\mathcal{T})$ that is specific to the specification Γ, based on features $\boldsymbol{\phi}(\mathcal{T}) = \begin{bmatrix}\phi_1(\mathcal{T}) & \phi_2(\mathcal{T}) & \phi_n(\mathcal{T})\end{bmatrix}$ where $n \geq d$. The first d features describe how the trajectory $\mathcal{T}$ relates to the constraint in Γ, i.e., if $\mathcal{T}$ uses actions that are associated with some γ_i. The features $\phi_{d+1}, \phi_{d+2}, \dots \phi_n$ are auxiliary features independent of the specification such as the time it takes to execute the trajectory. The problem statement is equivalent to Problem 2, but the feature vector contains d features that depend on the user specific specification Γ in addition to pre-defined features.

Problem 3 (Specification revision)**.** Given is a robot's state and action space $(\mathcal{X}, \mathcal{A})$, a set of tasks $\mathcal{Z} = \{(s_1, g_1), (s_2, g_2), \dots\}$, and a set of feature functions $\phi_1, \phi_2, \dots, \phi_n$. Furthermore, we have a user providing a specification Γ with hidden user preferences $\boldsymbol{w}^{\texttt{user}}$. For up to k iterations, the user can be queried about an ordering of a set of trajectories

$\{\mathcal{T}^1, \mathcal{T}^2, \dots\}$. Find an estimate $\boldsymbol{w}'$ of the user weights $\boldsymbol{w}^{\texttt{user}}$ such that the corresponding optimal trajectories $\mathcal{T}_i'$ for all tasks (s_i, g_i) minimize

$$\texttt{Err}(\boldsymbol{w}', \boldsymbol{w}^{\texttt{user}}) = \sum_{(s_i,g_i)\in\mathcal{Z}} \frac{\mathcal{F}_\Gamma(\mathcal{T}_i')}{\mathcal{F}_\Gamma(\mathcal{T}_i^{\texttt{user}})}. \tag{3.5}$$

Relationship of weights and trajectories In equation (3.2) we defined an optimal trajectory $\mathcal{T}^*$ as the minimizer of a cost function $\mathcal{F}$ which we assume to be linear (3.3) and hence parametrized by weights $\boldsymbol{w}$. That is, trajectories are found with a trajectory planner that takes some weight $\boldsymbol{w}'$ as an input and computes a corresponding trajectory $\mathcal{T}'$ that minimizes $\mathcal{F}(\mathcal{T}')$. Consequently, when solving Problem 2 and 3 we do not search over the set of all possible robot trajectories. Instead we only consider those trajectories that are optimal for *some* weight $\boldsymbol{w}$ and hence can actually be computed using the trajectory planner. Furthermore, the optimization problem that the trajectory planner solves is *sensitive* (or *robust*) towards its parameters $\boldsymbol{w}$. That is, small changes in the weights $\boldsymbol{w}$ can still result in the exact same solution, i.e., trajectory. Thus, in order to find an optimal solution to Problem 2 one does not need to learn the user weights $\boldsymbol{w}^{\texttt{user}}$ exactly. Throughout this thesis we explore the implications of this observation and use the sensitivity of the motion planner to design efficient learning approaches.

Chapter 4

Deterministic Preference Learning Framework for Specification Revision

A version of this chapter was published in the proceedings of the IEEE International Conference on Robotic and Automation (ICRA) 2018 [108].

We introduce a framework to solve the specification revision problem (Problem 3) where we consider discrete representation of the robot's state and action space using a graph. A mobile robot has to accomplish a single task which consists of navigating between a given a start and goal, i.e., finding a path between two vertices. A user specification consists of traffic rules, i.e., spatio-temporal constraints, for the robot. We assume that these might be soft constraints – the robot might be allowed to violate a constraint for sufficient time benefit. This trade-off can be captured by assigning a weight to each constraint, expressing for what time benefit a violation would be acceptable for the user. We show how these constraints can be incorporated into the robot's motion planning graph and pose a minimum cost path problem, where the cost combines the robot's travel time and the constraint violation. However, we do not ask the user to specify the weights on constraints since this might be challenging and potentially prone to errors. Instead we learn the weights for constraints using *active preference learning*: The user is iteratively presented with two candidate paths and chooses among them. We assume that this choice is always consistent with the user cost function, i.e., the user is deterministic. We show how the robot can efficiently learn from this user feedback and demonstrate the performance of this framework in a material transport simulation in an industrial facility.

4.1 Problem Formulation

We consider planning the path from a start to a goal in a robotic roadmap encoded as a graph. A user specification is given as a set of constraints, however without an explicitly defined weight. We want to minimize a cost that captures time and constraint violation. To learn the importance of a constraint relative to the time saved by violating it, we can ask for user feedback on alternative paths.

The problem takes the following inputs:

- A single weighted strongly connected multi-graph $G' = (V, E, \Psi, t)$ representing a robot's motion in an environment including obstacles. The weight function $t : E \to \mathbb{R}_{\geq 0}$ describes the traversal time for all edges $e \in E$ of the graph. Parallel edges in G' express different traversal speeds, e.g., between two nodes u and v there could be two edges with $t_1 = 1$ and $t_2 = 3$.
- A user specification consisting of n user constraints: $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_n\}$. A user constraint is defined as a pair (E_i, w_i) where $E_i \subseteq E$ and w_i is some constant in $\mathbb{R}_{\geq 0}$, defining a second weight for all edges $e \in E_i$. We refer to this weight as the user preference, which is also in units of time. The values of w_i are hidden.
- A start and a goal vertex $v_{\text{start}}, v_{\text{goal}}$ in V.

G' and Γ can be combined into a doubly weighted multi-graph $G = (V, E, \Psi, t, w)$ if the hidden weights w_i are known. On G, w is defined as follows:

$$w(e) = \sum_{\gamma_i \in \Gamma | e \in E_i} w_i \tag{4.1}$$

Moreover, for all connected pairs of vertices on G we delete all parallel edges where $w(e) = 0$ with exception of the one with the minimal value for t. On G we want to find an optimal path from v_{start} to v_{goal} minimizing t and w:

$$\min_P \sum_{e \in P} w(e) + t(e) \tag{4.2}$$

We state this objective without a scaling factor as w is in units of time. That is, w captures the maximum increase in completion time that the user is willing to allow in order to satisfy the constraint (or inversely, the minimum time saving necessary such that the

user would allow the violation of a constraint). Even though the values for w_i are unknown, we can pick an estimate for all w_i. Then, the graph collapses to a singly-weighted directed graph, and the shortest path can be efficiently computed with `Dijkstra's` algorithm. The goal is to find estimates for all w_i such that the corresponding path is optimal according to objective (4.2).

Example 1 (Problem setup)**.** In an industrial facility a user specifies a transportation task with a start and a goal location as well as constraints for the robot's movement like areas of avoidance, one-way roads or speed limits. Our problem can describe such a scenario as follows: The physical environment is represented with the single weighted multi-graph G', where the weight describes the traversal time between locations. The user constraints then define a second cost for some edges according to equation (4.1). Figure 4.1 shows an example of the resulting graph G, the traversal time $t(e)$ for all edges is some constant t, with the exception of the outer edges between vertices 3 and 4. The user defines a no-go area between vertices 2 and 5, resulting in a cost for traversing the respective edges. Moreover, a one way road is set up between vertices 5 and 8: A user constraint assigns a cost for moving from 5 to 8, but not for the opposite direction. Finally, there is a speed limit between 3 and 4 for both directions. In this case, we keep two parallel edges, one with the normal traversal time and the other with $t(e) = 2t$. While the edges with half speed have no user cost, a third user cost is assigned for traversing with full speed. Our goal is then to plan a path from start to goal that minimizes objective (4.2). To do so, we need to gain information about the hidden user preferences w_i.

4.2 Approach

We wish to learn user preferences by iteratively presenting the user with alternative paths. Based on the user feedback we learn the weights w_i of each constraint.

4.2.1 User Constraints and Cost Function

Let $(G', \Gamma, v_{\text{start}}, v_{\text{goal}})$ be an instance of our problem. We summarize the weights of all constraints $\gamma_i \in \Gamma$ with a column vector $\boldsymbol{w} = \begin{bmatrix} w_1 & w_2 & \cdots & w_n \end{bmatrix}^T$. Let $\boldsymbol{w}^{\texttt{user}}$ be the vector containing the hidden *user optimal* weights, while $\boldsymbol{w}$ is our estimate of $\boldsymbol{w}^{\texttt{user}}$.

Definition 2 (Constraint violation)**.** Consider a path P on G and a user constraint γ_i. We say that P violates γ_i if there exists an edge in P that is in E_i.

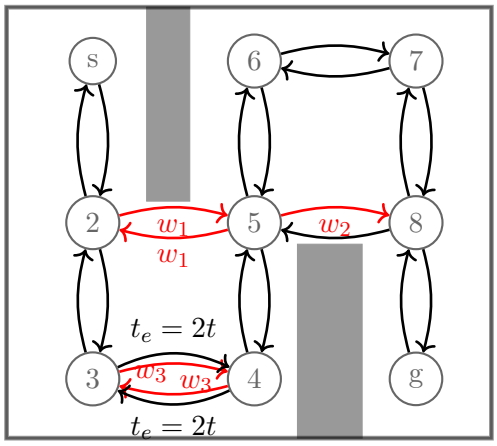


Figure 4.1: Representation of user constraints on a motion planning graph. The traversal times t_e between all edges are t, with exception of the outer pair between vertex 3 and 4. w_1, w_2, w_3 indicate the cost associated with the constrained edges, shown in red.

We introduce a variable to count the violated edges $\phi_i(P) = |E(P) \cap E_i|$. Moreover, we define $\boldsymbol{\phi}(P) = \begin{bmatrix}\phi_1(P) & \phi_2(P) & \dots & \phi_n(P)\end{bmatrix}$ as the row vector summarizing all violations of P, called the *violation vector*. If a path P is the shortest path according to equation (4.2) for some weight $\boldsymbol{w}^j$, we write P^j. Moreover, we denote the time of a path $t(P^j)$ as t^j and the violation vector as $\boldsymbol{\phi}^j$. The cost of a path P^j, *evaluated* by weights $\boldsymbol{w}^i$ is

$$c(P^j, \boldsymbol{w}^i) = \boldsymbol{\phi}(P^j)\boldsymbol{w}^i + t(P^j) = \boldsymbol{\phi}^j \boldsymbol{w}^i + t^j \tag{4.3}$$

This expresses how good the path P^j (that is optimal for a weights $\boldsymbol{w}^j$) is for some other user with weights $\boldsymbol{w}^i$. We introduced the hidden, optimal user weights as $\boldsymbol{w}^{\texttt{user}}$, the cost of some path P^j evaluated by the user then is $c(P^j, \boldsymbol{w}^{\texttt{user}}) = \boldsymbol{\phi}(P^j)\boldsymbol{w}^{\texttt{user}} + t^j$.

We now specify the user interaction: Let P^{best} be the currently best path. For a set of $k \geq 1$ different weights the user is presented with the corresponding paths, the violation vectors and the time improvements compared to P^{best}. They then provide a feedback in form of a vector $\boldsymbol{u} \in \mathbb{R}^k_{\geq 0}$. Thereby, u_i expresses the user preference for path P^i. If $u_i < u_j$, then the user prefers P^i over P^j. Consequently, $\boldsymbol{u}$ is a ranking of all presented alternatives. The user behaviour over multiple interactions is considered consistent if and only if all triplets (u_1, u_2, u_3) have a transitive relation.

4.2.2 Equivalence Regions

Shortest paths are sensitive to the edge weights on the graph [10]. As different values for $\boldsymbol{w}$ do not automatically lead to distinct solutions, we introduce *equivalence regions*.

Definition 3 (Equivalence region)**.** Let $\boldsymbol{w}^i$ and $\boldsymbol{w}^j$ be two different user weights and let P^i and P^j be their corresponding shortest paths. If $t(P^i) = t(P^j)$ and $\boldsymbol{\phi}(P^i) = \boldsymbol{\phi}(P^j)$, we call $\boldsymbol{w}^i$ and $\boldsymbol{w}^j$ *equivalent*. An *equivalence region* of a weight $\boldsymbol{w}^i$ is then the set of all weights that are *equivalent* to $\boldsymbol{w}^i$: $\Omega(\boldsymbol{w}^i) = \{\boldsymbol{w}^j \in \mathbb{R}^n_{\geq 0} | \boldsymbol{w}^j \text{ is } \textit{equivalent} \text{ to } \boldsymbol{w}^i\}$.

We notice that for any two equivalent weights $\boldsymbol{w}^i$ and $\boldsymbol{w}^j$ with corresponding optimal paths P^i and P^j, we have $c(P^i, \boldsymbol{w}) = c(P^j, \boldsymbol{w})$ for any $\boldsymbol{w}$, i.e., the paths have equal cost for *any* user weight. However, the paths themselves are not necessarily the same. Moreover, a weight $\boldsymbol{w}^i$ lies in multiple equivalence regions when the shortest path given $\boldsymbol{w}^i$ is not unique. Equivalence regions are closely related to the sensitivity of the shortest path to changes in the weights of the graph. According to [86], given a shortest path P^* on a weighted graph G, its sensitivity describes for which disturbances of the weights P^* remains optimal. In our case, the weight of an edge is described by $t(e) + w(e)$, hence an equivalence region describes the sensitivity towards $\boldsymbol{w}$.

Lemma 1 (Convexity)**.** Equivalence regions are convex.

Proof. We consider a weighted graph $G = (V, E, w)$ and investigate the sensitivity of the shortest path between some v_{start} and v_{goal} with respect to all edge weights. Let $|E| = m$. As a walk is a distinct sequence of vertices and edges, there is a finite number of paths $P^1, P^2, \dots P^l$ on the graph. The set of weights for which a path P^i is optimal satisfies

$$\sum_{e \in P^i} w(e) \leq \sum_{e \in P^j} w(e), \ \forall i \neq j, \ i, j = 1, 2, \dots, l. \tag{4.4}$$

All constraints in (4.4) define a half-space in $\mathbb{R}^m_{\geq 0}$. By definition, the intersection of halfspaces form a convex set [10]. *Equivalence regions* describe a special case where only some edge costs can vary, thus are also convex. □

4.2.3 Learning Framework

We now present the procedure for learning user preferences, illustrated in Figure 4.2. Assuming that there exists a path from start to goal that does not violate any user constraints,

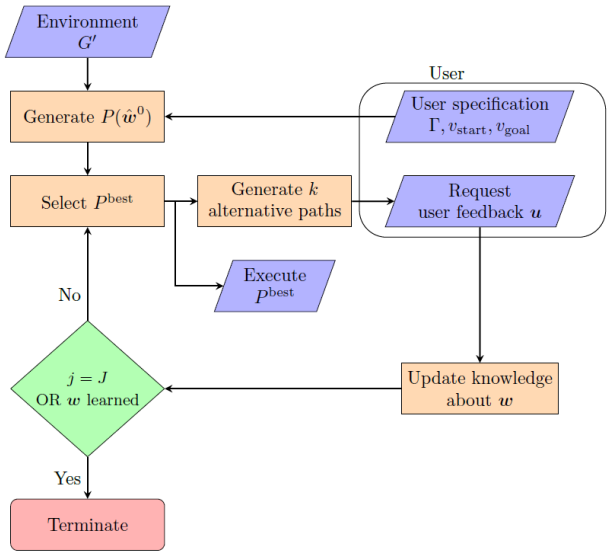


Figure 4.2: Flowchart of the learning process. j denotes the single iteration, J is the maximum number of user interactions.

we find such a path and denote it by P^0. In each iteration we select the path that received the best user feedback in the previous iteration P^{best} and execute it. We then iteratively generate sets of new paths in accordance to what we have learned about the user weights $\boldsymbol{w}$. After requesting user feedback for these alternative paths and P^{best}, we update our knowledge about the user weights. The process is repeated until the maximum number of user interactions J is reached or the algorithm converged to a weight that is equivalent to $\boldsymbol{w}^{\texttt{user}}$. The user feedback is on-the-loop: At any time P^{best} is executed, new user feedback might improve the solution but is provided at the user's convenience.

User model Let P^i and P^j be two different paths for some $\boldsymbol{w}^i$ and $\boldsymbol{w}^j$. If $c(P^i, \boldsymbol{w}^{\texttt{user}}) > c(P^j, \boldsymbol{w}^{\texttt{user}})$ the user always provides a feedback where $u_i > u_j$. Conversely, $u_i > u_j$ implies that the cost of P^j, evaluated by $\boldsymbol{w}^{\texttt{user}}$, is smaller than the cost of P^i by at least some small $\epsilon > 0$. If $c(P^i, \boldsymbol{w}^{\texttt{user}}) = c(P^j, \boldsymbol{w}^{\texttt{user}})$ the user feedback is $u_i = u_j$. Using the definition of c in Equation (4.3) we can derive the following constraints on the true user weights:

$$\begin{aligned} \text{if } u_j < u_i \text{ then } & \left(\boldsymbol{\phi}^j - \boldsymbol{\phi}^i\right) \boldsymbol{w}^{\texttt{user}} \leq t^i - t^j - \epsilon, \\ \text{if } u_j = u_i \text{ then } & \left(\boldsymbol{\phi}^j - \boldsymbol{\phi}^i\right) \boldsymbol{w}^{\texttt{user}} \leq t^i - t^j, \text{ and} \\ & \left(\boldsymbol{\phi}^i - \boldsymbol{\phi}^j\right) \boldsymbol{w}^{\texttt{user}} \leq t^j - t^i. \end{aligned} \tag{4.5}$$

From each pair (u_i, u_j) we learn that the user weight $\boldsymbol{w}^{\texttt{user}}$ lies either on one side of the hyperplane described by equation (4.5) if $u_i \neq u_j$, or on the hyperplane if $u_i = u_j$.

Definition 4 (Feasible Set)**.** Given a user feedback $\boldsymbol{u}$ of length k, we can derive $k-1$ non-redundant constraints on the user weights $\boldsymbol{w}^{\texttt{user}}$ from expression (4.5). The feasible set is the intersection of all $k-1$ half-spaces and hyperplanes described by these inequalities and is a convex set. It can be written as a polyhedron of the form $\boldsymbol{Aw} \leq \boldsymbol{b}$ [10].

The initial feasible set before any user feedback is bounded by some finite values $\boldsymbol{w}^{\max}$ as all paths on G have finite length. If prior information about the user constraints is available, it can be incorporated by adding rows to $\boldsymbol{Aw} \leq \boldsymbol{b}$, which define the feasible set, given the prior is linear.

Example 2 (Learning constraints of the feasible set)**.** We consider a simplified version of the introductory example with undirected edges. We define two areas of avoidance both affecting one edge, shown in Figure 4.3. The traversal time of any edge is 1. The path P^1 (blue) traverses the vertices $\{s, 2, 5, 8, g\}$ while P^2 (green) traverses $\{s, 2, 3, 4, 5, 8, g\}$. Hence, $\boldsymbol{\phi}(P^1) = \begin{bmatrix}1 & 1\end{bmatrix}$, $t(P^1) = 4$ and $\boldsymbol{\phi}(P^2) = \begin{bmatrix}1 & 0\end{bmatrix}$ with $t(P^2) = 6$. Given a user feedback that $u_2 \leq u_1$ we can infer that $\left(\begin{bmatrix}1 & 0\end{bmatrix} - \begin{bmatrix}1 & 1\end{bmatrix}\right) \begin{bmatrix}w_1 & w_2\end{bmatrix}^T \leq 4-6$, and therewith $w_2 \geq 2$. The feasible set then is defined by $\begin{bmatrix}0 & -1\end{bmatrix} \begin{bmatrix}w_1 & w_2\end{bmatrix}^T \leq \begin{bmatrix}-2\end{bmatrix}$.

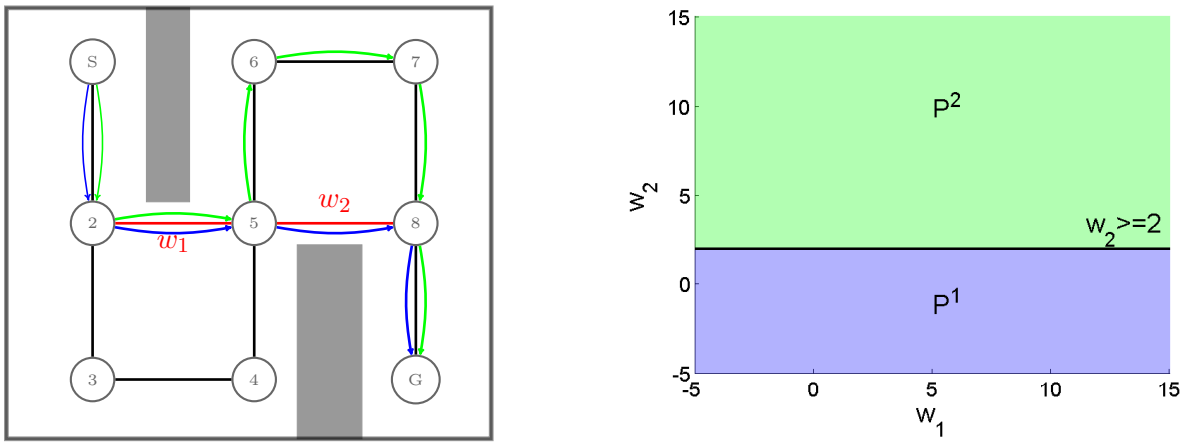
4.2.4 Deterministic Learning Algorithm

Notice that the user ranks paths of equivalent weights equally. Thus, our goal is to find a weight $\boldsymbol{w}^{\text{best}}$ that lies in the equivalence region of the true user preference $\boldsymbol{w}^{\texttt{user}}$. Based on the flowchart in Figure 4.2, we introduce Algorithm 1.

In line 1 of the algorithm we initialize the feasible set with some finite upper bounds $\boldsymbol{w}^{\max}$. We initialize $\boldsymbol{w}^{\text{best}} = \boldsymbol{w}^{\max}$ and find the corresponding path P^{best}, which does not violate any user constraints. Values for $\boldsymbol{w}^{\max}$ can be found according to the following lemma:

Lemma 2 (Upper bound)**.** A weight $\boldsymbol{w}^{\max}$ such that the corresponding path does not violate any constraints can be found as follows: Find the path P^∞ that is optimal for weights $\boldsymbol{w}^\infty$ where $w_i^\infty = \infty$ for all $i = 1, \ldots, n$. Then choose $w^{\max} = t(P^\infty)$ and $\boldsymbol{w}^{\max} = w^{\max}\mathbf{1}^T$.

Proof. Let P' be a path that violates at least one constraint. According to equation (4.3), we have $c(P', \boldsymbol{w}^{\max}) \geq w^{\max} + t(P')$. As $t > 0$ for all edges, P' has a higher cost than $P^{\max}$. □



(a) Graph with different paths

(b) Feasible set.

Figure 4.3: Example: Deterministic learning. (a) illustrates an example environment with two constraints (red) and two distinct paths (blue and green) from start to goal. (b) shows how the hyperplane learned from comparing the paths separates the weight space.

In line 4, we iteratively find k new weights according to some admissible policy π, which is defined as follows:

Definition 5 (Admissible policy.)**.** A policy is a function π that maps from $(\boldsymbol{A}, \boldsymbol{b}, k, \mathcal{W}, \boldsymbol{w}^{\text{best}})$ to a set of k weights $\mathcal{W}^{\text{new}}$. A policy is *admissible* if each $\boldsymbol{w} \in \mathcal{W}^{\text{new}}$ satisfies:

(i) $\boldsymbol{A}\boldsymbol{w} \leq \boldsymbol{b}$, and

(ii) $\boldsymbol{w}$ is not equivalent to any previous weight in $\mathcal{W}$ or any other new weights in $\mathcal{W}_{\text{new}}$.

We discuss two alternative policies in Sections 4.2.5 and 4.2.6. In lines 7-8 we compute the corresponding paths and present them to the user together with the best path so far. Based on the user feedback, the feasible set and the path with the best user response so far are updated (line 9-10) and all new weights are added to the set $\mathcal{W}$ (line 11). Eventually, the policy will not be able to find new weights. Then, an empty set is returned and the algorithm terminates in line 6. Using *equivalence regions* we can formally state a proposition for optimality:

Proposition 1 (Termination)**.** If there exists an *equivalence region* containing all vertices of the polyhedron $\boldsymbol{A}\boldsymbol{w} \leq \boldsymbol{b}$, then $\boldsymbol{w}^{\text{best}}$ is *equivalent* to $\boldsymbol{w}^{\texttt{user}}$ and the optimal solution is obtained.

Algorithm 1: Active learning of constraint weights of user specifications.

Input: G', Γ, J, k
Output: $\boldsymbol{w}^{\text{best}}$

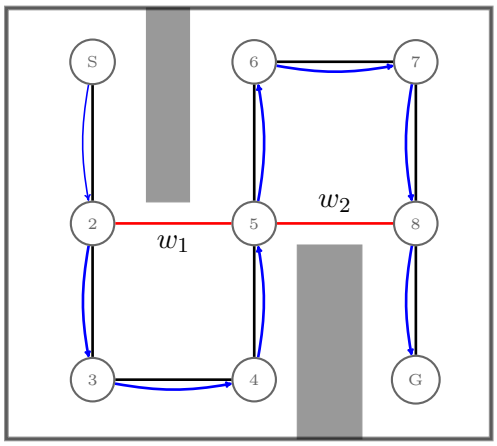
Initialize $\boldsymbol{A}$, $\boldsymbol{b}$, $\boldsymbol{w}^{\text{best}}$, P^{best} and $\mathcal{W} = \emptyset$
for $j = 1$ *to* J **do**
  $\mathcal{W}_{\text{new}} \leftarrow \pi(\boldsymbol{A}, \boldsymbol{b}, k, \mathcal{W}, \boldsymbol{w}^{\text{best}})$
  if $\mathcal{W}_{\text{new}} = \emptyset$ **then**
    return $\boldsymbol{w}^{\text{best}}$
  Compute paths $P^1, \ldots, P^k$ for all $\boldsymbol{w} \in \mathcal{W}_{\text{new}}$
  Get user feedback $\boldsymbol{u}^j$ for paths $P^{\text{best}}, P^1, \ldots, P^k$
  Update $\boldsymbol{A}$ and $\boldsymbol{b}$ based on $\boldsymbol{u}^j$ (Definition 4)
  Choose $\boldsymbol{w}^{\text{best}}$ as the element from $\boldsymbol{w}^{\text{best}} \cup \mathcal{W}_{\text{new}}$ with the best user feedback, $P^{\text{best}} = P(\boldsymbol{w}^{\text{best}})$
  $\mathcal{W} = \mathcal{W} \cup \mathcal{W}_{\text{new}}$
return $\boldsymbol{w}^{\text{best}}$

Proof. Suppose there exists an *equivalence region* containing all vertices of $\boldsymbol{A}\boldsymbol{w} \leq \boldsymbol{b}$. By convexity of the feasible set and the equivalence regions, we conclude that all feasible weights lie in the equivalence region. As $\boldsymbol{w}$ lies in the feasible set, all feasible weights are equivalent to $\boldsymbol{w}$. Trivially, the corresponding path is optimal. □

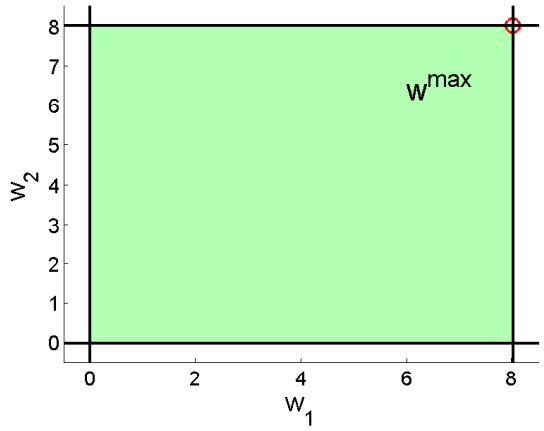
Naively checking if all vertices of the feasible set are equivalent is impractical as the number of vertices can grow exponentially with the number of user feedback. However, in the current work we focus on approximating the optimal solution.

Example 3 (Learning user preferences)**.** Again, we consider the graph from Figure 4.3. We initially find the upper limit $w^{\max} = 8$ and obtain a path P^0 with $t(P^0) = 8$, shown in Figure 4.4. We then pick a new point of the feasible set, say $\boldsymbol{w}^1 = \begin{bmatrix} 0 & 0 \end{bmatrix}^T$. The resulting path P^1 is the same as the previous example with $\boldsymbol{\phi}^1 = \begin{bmatrix} 1 & 1 \end{bmatrix}$. We present the user with P^0 and P^1 and they prefer P^1. According to expression (4.5) we learn $w_1 + w_2 \leq 4$. In the next iteration we present P^2 from the previous example and get the additional constraint $w_2 \geq 2$. The resulting polyhedron is shown in Figure 4.5. After three iterations all vertices of the feasible set are *equivalent* and result in path P^2.

Using *equivalence regions* we establish our main result:



(a) Graph with initial path P^0.



(b) Feasible set

Figure 4.4: Deterministic learning, initial iteration(a) shows the initial Path P^0 on the graph. In (b) we see the initial feasible set with $w^{\max}$.

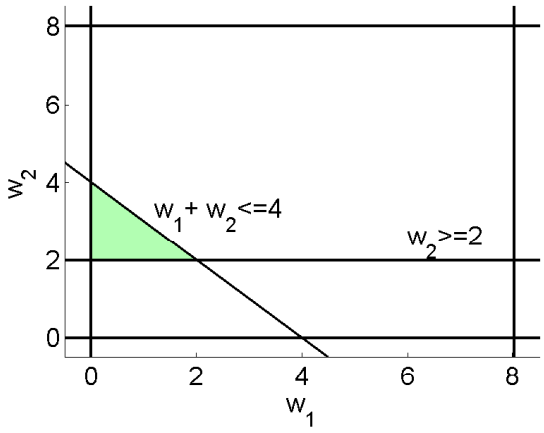


Figure 4.5: Deterministic learning, final iteration iteration. We show the feasible set after three comparison, the solution is now uniquely determined.

Proposition 2 (Completeness)**.** Suppose that we have an admissible policy π. Then Algorithm 1 is complete.

Proof. Let $\boldsymbol{w}^i$ and $\boldsymbol{w}^j$ be non-equivalent weights. From the user model and inequalities (4.5) we derive the following relation when the user prefers P^i over P^j:

$$u_i < u_j \implies \boldsymbol{w}^{\texttt{user}} \notin \Omega(\boldsymbol{w}^j). \tag{4.6}$$

In each iteration where the user chooses one path over the other, we shrink the feasible set by at least one equivalence region. Now we consider that the user does not prefer either path:

$$u_i = u_j \implies \boldsymbol{w}^{\texttt{user}} \in \texttt{span}\left(\Omega(\boldsymbol{w}^i) \cap \Omega(\boldsymbol{w}^j)\right). \tag{4.7}$$

We notice that the interior of $\Omega(\boldsymbol{w}^i)$ and the interior $\Omega(\boldsymbol{w}^j)$ are disjoint. Thus, the intersection of the equivalence regions is of lower dimensionality. In the case that $u_i = u_j$ we reduce the dimension of the feasible set.

As both, the number of equivalence regions and the number of user constraints, i.e., dimensions, are finite, the feasible set eventually either consists of only the *equivalence region* $\Omega(\boldsymbol{w}^{\texttt{user}})$, or $\boldsymbol{w}^{\texttt{user}}$ is uniquely determined by equality constraints. Hence, algorithm terminates after finite iterations. □

Notice that in a worst case the number of iterations of Algorithm 1 equals the number of paths between the start and goal vertex, which can be exponential in the size of the input. Nonetheless, in Section 4.3 we show that in practice the number of iterations is linear in the size of the input.

4.2.5 Vertex Search

We propose the policy $\pi_{\texttt{vertexSearch}}$ to find new vertices, shown in Algorithm 2. The *feasible set* can be understood as an unknown graph, where the vertices are the extreme points of the polyhedron, connected by its edges. We apply depth first search (DFS) to explore the feasible set, starting at the previously best solution $\boldsymbol{w}^{\text{best}}$. Line 5 and lines 8-12 explore neighbouring vertices, which is computationally inexpensive and similar to the pivot step in the simplex algorithm for linear programs [10]. If a new vertex is found, it is added to the set $\mathcal{W}_{\text{new}}$ (line 6-7). The algorithm stops when either k new vertices have been found, the DFS has exhausted all vertices or a maximum number of iterations is reached. The policy is a form of pattern search [75], where the set of search directions consists of the vectors from the current weight to all vertices of the feasible set. Notice that for $i_{\max} \to \infty$ the policy is admissible. Furthermore, DFS explores a given graph in linear time.

Algorithm 2: Policy $\pi_{\texttt{vertexSearch}}$ for finding new weights using depth first search.

Input: $\boldsymbol{A}$, $\boldsymbol{b}$, k, $\mathcal{W}$, $\boldsymbol{w}^{\text{best}}$
Output: $\mathcal{W}_{\text{new}}$

```
1  Initialize set W_new = ∅, openList = {w^best} and maximum iterations i_max
2  for i = 0 to i_max do
3      if |W_new| = k or openList is empty then
4          return W_new
5      w̃ = openList.pop()
6      if w̃ is not equivalent to any w ∈ W_new ∪ W then
7          Add w̃ to W_new
8      if w̃ is not labelled as discovered then
9          Label w̃ as discovered
10     for all w' ∈ getAdjacentVertices(w̃) do
11         if w' ∉ openList and w̃ is not labelled as discovered then
12             openList.insert(w')
13 return W_new
```

4.2.6 Minimal Vertex

As $\pi_{\texttt{vertexSearch}}$ potentially exhausts an exponentially growing graph, we introduce a heuristic policy. The previous approach did not use any information from the shortest path problem and explored the feasible et in a naive way. For instance, a user specification may contain numerous constraints that do not affect the path between a start and goal, which makes them irrelevant for the problem instance.

Therefore, we propose the heuristic policy $\pi_{\texttt{minSearch}}$ that always tries to minimize the weights within the current feasible set, illustrated in Algorithm 3. In lines 2-4, we find the minimal feasible weight and add it to the set $\mathcal{W}_{\text{new}}$ if it is not equivalent to previous weights. If $k = 1$ and the minimal weight was not previously preferred by the user, the algorithm terminates in line 6. Otherwise, we invert the search direction for all constraints that the path based on the minimal weight violates (line 7-11) to generate new weights (line 13-15). Finally, if the heuristic has been unsuccessful, $\pi_{\texttt{vertexSearch}}$ is called (line 18).

Algorithm 3: Heuristic policy $\pi_{\texttt{minSearch}}$ for finding minimal new weights

Input: $\boldsymbol{A}$, $\boldsymbol{b}$, k, $\mathcal{W}$,$\boldsymbol{w}^{\text{best}}$
Output: $\mathcal{W}_{\text{new}}$

$\mathcal{W}_{\text{new}} = \emptyset$
$\boldsymbol{w} = \min \mathbf{1}^T \boldsymbol{w}$ s.t. $\boldsymbol{A}\boldsymbol{w} \leq \boldsymbol{b}$
if $\boldsymbol{w} \notin \mathcal{W}$ **then**
    add $\boldsymbol{w}$ to $\mathcal{W}_{\text{new}}$
if $|\mathcal{W}_{\text{new}}| = k$ **then**
    return $\mathcal{W}_{\text{new}}$
`newSearchDirections` $= \emptyset$
for γ_i *where* $\boldsymbol{\phi}_i(P(\boldsymbol{w})) > 0$ **do**
    $\bar{\boldsymbol{c}} = \boldsymbol{c}$
    $\bar{\boldsymbol{c}}_i = -1$
    add $\bar{\boldsymbol{c}}$ to `newSearchDirections`
for $\bar{\boldsymbol{c}} \in$ `newSearchDirections` **do**
    $\tilde{w} = \min \bar{\boldsymbol{c}}^T \boldsymbol{w}$ s.t. $\boldsymbol{A}\boldsymbol{w} \leq \boldsymbol{b}$
    if $\tilde{\boldsymbol{w}} \notin \mathcal{W}$ **then**
        add $\tilde{\boldsymbol{w}}$ to $\mathcal{W}_{\text{new}}$
    if $|\mathcal{W}_{\text{new}}| = k$ **then**
        return $\mathcal{W}_{\text{new}}$
$\mathcal{W}' \leftarrow \pi_{\texttt{vertexSearch}}(\boldsymbol{A}, \boldsymbol{b}, k - |\mathcal{W}_{\text{new}}|, \mathcal{W} \cup \mathcal{W}_{\text{new}}, \boldsymbol{w})$
return $\mathcal{W}_{\text{new}} \cup \mathcal{W}'$

4.3 Evaluation

In the experiments we used layouts of real industrial facilities. We generate graphs G' by uniform grid-based sampling. We use a graph G' with 3646 vertices and 12456 edges. User feedback in each iteration is simulated using the user cost $c(\cdot)$, *evaluated* by the user as introduced in equation (4.3). Given the learned weight $\boldsymbol{w}^{\texttt{best}}$, let $P^{\texttt{best}}$ be the corresponding optimal path. We use the relative error of the learned path compared to the optimal path from Chapter 3; which is defined as follows:

$$\texttt{Err}(P^{\texttt{best}}, \boldsymbol{w}^{\texttt{user}}) = \frac{c(P^{\texttt{best}}, \boldsymbol{w}^{\texttt{user}})}{c(P^{\texttt{user}}, \boldsymbol{w}^{\texttt{user}})} - 1 \tag{4.8}$$

Table 4.1: Different types of user for Experiment 1.

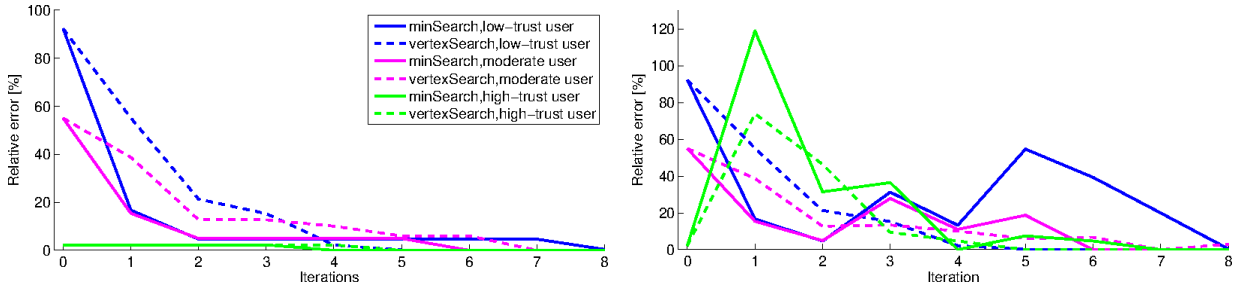
User type	description	# constraints
low-trust	specifies many constraints with a range of weights	20
moderate	specifies fewer constraints with moderate or high weights	10
high-trust	specifies very few constraints, all have high weights	4

If $\boldsymbol{w}^{\texttt{best}}$ is *equivalent* to $\boldsymbol{w}^{\texttt{user}}$, then $\texttt{Err}(P, \boldsymbol{w}^{\texttt{user}}) = 0$. Algorithm 1 does not have access to $\boldsymbol{w}^{\texttt{user}}$. Therefore, alternative paths can be presented even if the optimal solution was already found. A classic path planning approach without learning user preferences only finds paths that do not violate any user constraints, i.e., the path P^0 of our algorithm. Therefore, P^0 is used as a baseline to highlight the benefit of our approach.

We conduct two experiments: In the first, we propose models for three different types of users and their weights, while in the second experiment we randomly generate user weights for a predefined set of constraints. In both experiments, we set the maximum number of user iterations J to 30. As discussed in the previous chapter, the check for convergence is approximated. We choose $i_{\max} = 50$ as the number of iterations for $\pi_{\texttt{vertexSearch}}$. In each iteration the user is presented one new as well as the currently best path.

Experiment 1 In this experiment, we consider three user classes to mimic different realistic users, summarized in Table 4.1. Problem instances are generated from a set of start-goal configurations. In Figure 4.6 we compare $\pi_{\texttt{minSearch}}$ and $\pi_{\texttt{vertexSearch}}$ for learning the preferences of all three users. We show the relative error of the best path so far P^{best} in subfigure (a) as well as the relative error of the new path P^j in subfigure (b), at each iteration j averaged over all trials.

In Figure 4.6 we see the most considerable improvement for the low-trust user. We recall that P^0 respects all user constraints, but as the low-trust user has a relatively low preference for each, the optimal path is likely to be violating some constraints. In contrast, the high-trust user seldom accepts any violation. Therefore, P^0 is already the optimal solution in most cases. Moreover, for the low-trust user $\pi_{\texttt{minSearch}}$ finds relatively good solutions more quickly than $\pi_{\texttt{vertexSearch}}$. For the moderate user we observe an intermediate result: While the error evolves similar to the low-trust user, the improvement compared to P^0 is smaller.



(a) Relative error the current best path P^{best} (b) Relative error of each presented path P^j

Figure 4.6: Comparison of the policies $\pi_{\texttt{minSearch}}$ and $\pi_{\texttt{vertexSearch}}$ for all three users, described in Table 4.1

Although the three users feature different numbers of constraints, the algorithm converges after five to eight iterations. There are two reasons for this. First, additional constraints are not always relevant for each shortest path problem, the number of equivalence regions does not necessarily increase with an additional constraint. Second, even though the number of equivalence regions increases, Algorithm 1 does not necessarily search through them exhaustively. Some hyperplanes according to expression (4.5) reduce the feasible set by more than one equivalence region.

In summary, the user type has little influence on the number of iterations. Nonetheless, especially the low-trust user benefits from our approach compared to classic path planning; the optimal path outperforms P^0 by over 80%.

Experiment 2 User weights are uniformly drawn from $[0, \lambda w^{\max}]$ where $0 < \lambda \leq 1$ and $w^{\max}$ is found as in Lemma 2. High values for λ imply that the user has a widely varying weights, including increasingly more high preferences such that violations are accepted more rarely. Table 4.2 compares the best solutions each policy finds after j iterations. Even though in many cases both policies perform equally well, $\pi_{\texttt{minSearch}}$ still outperforms $\pi_{\texttt{vertexSearch}}$ between iteration two and eight.

Consequently, Table 4.3 shows the results of using $\pi_{\texttt{minSearch}}$. The number of iterations until convergence is growing more slowly than the number of constraints. From the $I_{10\%}$ column we see that relatively good solutions are often found before convergence, however, for $\lambda = 0.5$ this is because P^0 is often the optimal path. The improvement of the optimal path ΔC monotonically increases with the number of constraints. When P^0 has to take large detours to respect all constraints, violations lead to larger improvements encouraging the user to accept an alternative path. Overall, the final solution outperforms P^0 in 57.4%

Table 4.2: . Each row indicates in how many trials (%) each policy achieved a lower $c(P^{\text{best}})$ at iteration j.

Iteration j	1	2	3	4	5	6	7	8	9
$\pi_{\texttt{minSearch}}$	0	27	24	23	20	7	3	1	0
$\pi_{\texttt{vertexSearch}}$	0	10	10	9	2	0	0	0	0
Tie	100	63	66	68	78	93	97	99	100

Table 4.3: Notation: n the number of constraints, λ is the range for sampling weights, $\mathcal{I}$ the number of iterations until convergence, $\mathcal{I}_{10\%}$ the number of iterations until a 10% approximation is found, ΔC the improvement of the final P^{best} compared to P^0, $\#A^*$ the number of A* calls and t_{mean} the mean runtime to find a new weight in each iteration.

n	λ	$\mathcal{I}$	$\mathcal{I}_{10\%}$	$\Delta C[\%]$	$\#A^*$	$t_{\text{mean}}[s]$
5	0.005	7	1	24.9	59	33.1
	0.05	9	9	22.3	66	40.2
	0.5	4	1	8.1	28	27.5
10	0.005	6	1	27.6	72	38.3
	0.05	9	9	26.1	72	34.7
	0.5	12	1	9.7	58	30.6
20	0.005	11	2	38.1	131	54
	0.05	11	8	33.5	113	42.9
	0.5	11	1	15.4	56	22.5

of the trials. The number of shortest path problems Algorithm 1 has to solve ($\#A^*$) increases roughly linearly with the number of constraints. The runtime for finding new non-*equivalent* weights t_{mean} in each iteration generally increases with n, however, only for $\lambda = 0.005$ monotonically.

4.4 Summary and Discussion

We proposed a methodology to learn user preferences for spatial and simple temporal constraints in a shortest path problem via human-robot interaction. Based on a deterministic user model, we have shown how information about feasible weights can be derived from simple user feedback, and introduced equivalence regions to partition the weight space. Using these results a learning algorithm was proposed followed by its proof for completeness. After introducing two policies, we showed simulation results based on real world environments. In our evaluation, the number of iterations grows more slowly than the number of constraints. Moreover, especially for low user weights the performance of the robot task improves significantly by learning the user preferences, allowing for a user-on-the-loop robot task specification in an accessible way.

Nonetheless, the presented framework has three main areas of improvement: First, even though the proposed algorithm is complete, the runtime for finding new weights might increase drastically for high dimensions. Second, our work is based on a deterministic, linear user model. This is a potential shortcoming as real users at least occasionally might violate these assumptions, i.e., contradict their own choices. In Chapter 6 we address both issues: We further study equivalence regions and show that determining the extents of an equivalence region is in general computationally intractable. Moreover, we introduce a probabilistic user model. We propose a new learning algorithm based on equivalence regions and demonstrate its robustness in simulations. Third, in the evaluation we used either manually designed or randomly generated data that do not necessarily reflect real users. In the next chapter, we present the results of a user study and show that the proposed framework helps nearly all users to create better specifications, i.e., improves the task completion time.

Chapter 5

Evaluation of the Deterministic Framework in a User Study

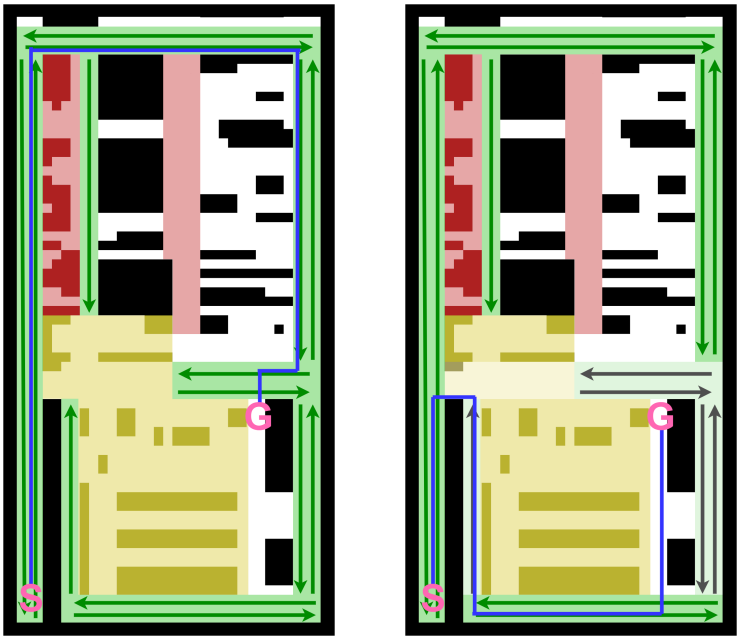
The research in this chapter was conducted in conjunction with a MASc. student of the University of Waterloo, Alexandru Blidaru[1], and was published in the International Journal on Robotics Research (IJRR) in 2020 [107].

5.1 Overview

We extend the framework from Chapter 4 to a multi-task scenario and evaluate its practicality in a user study. We show that, using our proposed framework, most users accept alternative paths and obtain revised specifications, which improve the robot's performance by 14% on average, within at most 20 user interactions. Further, we show that while initially provided specifications vary largely between users, the learning interaction results in specifications that are more similar. Especially those users whose constraints initially drastically affect performance, benefit most from the interaction.

We consider an industrial facility where the work space is shared between pedestrians, human-operated vehicles, and autonomous vehicles. While mobile robots are capable of navigating safely given a description of the environment, their choice of routes might not fit the preferences and established rules of humans. Without a further specification, robots

[1] A. Blidaru and N. Wilde contributed equally unless stated otherwise.



(a) Initial specification. (b) Revised Specification.

Figure 5.1: Example environment (white) with obstacles (black), user defined constraints and a task start and goal locations. Roads are drawn in green with an arrow indicating the direction. Speed limit zones where only half the maximum speed is allowed are drawn in yellow, while areas of avoidance are illustrated in red. In 5.1a we see the initial path respecting all user constraints and preferring roads. Following the user interaction, we obtained the revised specification in 5.1b, in which some of the constraints are less important to the user (faded yellow and green). Thus the shortest path for the given task is significantly shorter, at the cost of a violated speed limit zone, and a violated road zone.

are unaware of the context, e.g. areas that are designated for vehicles or areas where robot traffic is undesired. Additionally, the behaviour of autonomous vehicles can appear unpredictable to humans.

To address these issues, users can specify a set of traffic rules to guide robots in such environments. In current industrial practice such rules for robot behaviour are designed by trained personnel [76]. We employ an extension of the graphical user interface (GUI) from [13] where a user can specify traffic rules such as one and two way roads, areas of avoidance and reduced speed zones by graphically defining polygons on the map of the environment (we synonymously refer to the user defined traffic rules as user constraints).

In Chapter 4 we presented a framework for revising specifications, helping inexperienced users to improve the efficiency of the resulting robot behaviour. An initial specification assumes that no traffic rules are allowed to be violated. This allows for computing a path that strictly follows the defined rules, shown in Figure 5.1 (a). Through active preference learning we try to revise the specification and improve the performance. We extend our work from Chapter 4 to consider multiple start-goal tasks. In each iteration of user interaction the preferred path becomes the best path so far. If the same task is presented to the user in a later interaction, the previously preferred path constitutes one of the two alternatives. This corresponds to a user–on–the–loop framework: the current best path can already be executed as the user approved it previously. The idea of showing the previously preferred solution in the next iteration is also used in [77]. Depending on the user feedback, the violation of some rules might be acceptable for a certain time benefit. In the example in Figure 5.1 we show a revised specification following the active learning process, which decreases the task time. Thereby, one speed limit zone has become less important and three roads are rewarded less, such that the robot traverses through the free space, and a one-way road is effectively removed from the specification.

5.2 Problem Description

The proposed approach contains the following two components: First, having obtained a user specification, we extend the active learning approach from Chapter 4 to a multi-task scenario. Following this, we apply the metrics proposed in [13] to evaluate the impact of the specification, and show how the learning system improves the quality of the robot's task performance.

5.2.1 Learning User Preferences

Problem setup Similar to the problem in Chapter 4, the robot receives a description of the environment and a user specification. The environment is represented as a weighted strongly connected multigraph $G' = (V, E, \Psi, t)$. The weight t on the graph encodes the time a robot requires to traverse an edge. Instead of a single task (as in Chapter 4), we consider a set of tasks given by a set of ordered pairs $\mathcal{Z} = \{(s_1, g_1), (s_2, g_2), \dots\}$ where s_i and g_i are vertices on G'. A single task consists of navigating from a start s_i to a goal g_i.

The user specification is a set of constraints $\Gamma = \{\gamma_1, \gamma_2 \dots, \gamma_d\}$. Each constraint γ_k is a pair $(E_k, w_k^{\texttt{user}})$, where E_k is a subset of the edges of G' and $w_k^{\texttt{user}}$ is a hidden user

weight for the constraint. We slightly change our convention and allow these weights to be positive or negative: A positive weight $w_k^{\texttt{user}}$ expresses a penalty for using edges in E_k while a negative weight expresses a reward. Notice that a road on the interface entails two constraints: A reward for using the road in the direction of travel and a penalty for moving the wrong way. Consequently, a two way road maps to four constraints.

We incorporate the user specification by creating a doubly weighted graph $G = (V, E, \Psi, t, w^{\texttt{user}})$. For each edge e in G the second weight $w^{\texttt{user}}(e)$ is defined as the sum of all $w_k^{\texttt{user}}$ that belong to a constraint containing e. Our objective is to find paths $P_i^{\texttt{user}}$ between all s_i and g_i that are optimal with respect to

$$\min_{P_i} \sum_{e \in P_i} w^{\texttt{user}}(e) + t(e). \tag{5.1}$$

The true user weights $w_k^{\texttt{user}}$ are latent, i.e., we do not ask the user to define $w_k^{\texttt{user}}$ during the specification. Nonetheless, given estimates $\hat{w}_k$ of the true user weights, we can also construct a doubly weighted multigraph $\hat{G}$. Moreover, the weights are defined in units of time, allowing us to pose the multi-objective optimization as an unweighted sum. To learn about the weights, we can query the user. In a query we present them with a pair of paths (P_i^1, P_i^2) for a selected start-goal pair (s_i, g_i). Considering only pairs instead of more than two paths at a time is motivated by reducing the burden on the user, as choosing between numerous alternatives is more demanding [49].

Linear learning model We distinguish *penalty* and *reward* constraints. Penalty constraints include the edges within an avoid zone, edges within a speed-limit zone where the traversal time does not correspond to obeying the speed limit and the edges going against the defined direction of travel in a one-way road. Reward constraints describe the edges that follow the direction of traffic on a road. Thus, for any penalty constraint the weight w_i is non-negative while for a reward constraint the weight is non-positive. As $\boldsymbol{w}^{\texttt{user}}$ is hidden, we initially only know that its values are finite, i.e., $l_i \leq w_i^{\texttt{user}} \leq u_i$ for some real number lower and upper bounds l_i and u_i.

Let P^i and P^j be two paths. If the user prefers path P^i it implies that $c(P^i) \leq c(P^j)$. We can write this as a half-space in $\mathbb{R}^d$ containing $\boldsymbol{w}^{\texttt{user}}$

$$\{\boldsymbol{w} \in \mathbb{R}^d | (\boldsymbol{\phi}^i - \boldsymbol{\phi}^j)\boldsymbol{w}^{\texttt{user}} \leq t^j - t^i\}. \tag{5.2}$$

Thus, obtaining user feedback allows us to iteratively learn inequality constraints on the user weights. We write the intersection of the learned half-spaces as a polyhedron

$$\mathcal{F} = \{\boldsymbol{w} \in \mathbb{R}^d | l_i \leq w_i \leq u_i,\ \boldsymbol{A}\boldsymbol{w} \leq \boldsymbol{b}\}, \tag{5.3}$$

which we refer to as the *feasible set*. The half-spaces obtained from user feedback can be used to iteratively shrink the feasible set.

5.2.2 Active Learning Algorithm

The learning algorithm employed in the user study is the same as described in Chapter 4, using the $\pi_{\texttt{minSearch}}$ policy (Algorithm 3). In this section we briefly discuss some important details of the practical application.

After receiving the initial user specification but before the interactive learning, we want to choose an initial path that follows all user constraints. First, we set the lower and upper bounds for all weights: For penalty constraints, $l_i = 0$, while u_i is the sum of all t_i for all edges e_i on the graph G'. For reward constraints we have $u_i = 0$. However, the lower bound l_i needs to guarantee that there are no negative cycles such that that the optimization problem is well-defined. Let γ_i be a constraint containing edges that follow a road. To obtain the lower bound we choose the negative of the length of the shortest edge in the constraint, denoted by $t_i^{\min}$. Further, we subtract a small amount such that a path planner breaks ties in favor of paths using fewer edges: $l_i = -(1-\epsilon)t_i^{\min}$ where $0 < \epsilon \ll 1$. We then pick the initial path P^0 that is optimal for some weight $\boldsymbol{w}^0$ where $w_i^0 = u_i$ if γ_i is a penalty constraint and $w_i^0 = l_i$ if γ_i is a reward constraint. Hence, P^0 follows the user specification, i.e., it does not violate any *avoid*- or *speed*-zones, does not traverse roads in the wrong direction and uses roads as much as possible. In each iteration we then present the user with the current best path and one alternative. If the user prefers the alternative it becomes the new current best path P^{best}.

5.2.3 Multiple Tasks

The motivation for a multi-task scenario is two-fold: In practice, an autonomous robot usually has to perform more than one task, making the multi-task setting more relevant. Furthermore, learning about several tasks in parallel has potential computational advantages as it allows for an additional degree of freedom in the active learning: Choosing what task to learn about in the next iteration.

We now discuss how Algorithm 1 from Chapter 4 can be adapted to multiple tasks. A multi-task scenario is set up from a set of points of interest in the environment yielding multiple start-goal pairs. We learn about the constraints from each interaction by obtaining feedback for a single task. We can combine the information from multiple rounds by intersecting the *feasible set* of all individual tasks. This leads to a *passive* learning effect:

Algorithm 4: Active Selection of the new task for learning.

Input: $\boldsymbol{A}$, $\boldsymbol{b}$, L
Output: $l_{\max}$

```
1  time_saving = −∞
2  l* = ∅
3  for (s_i, g_i, w_i^best) in L do
4      Pick P^1 as the optimal path for ŵ_i^best
5      w_i^new = minVertex(A, b, 1, W_i, ŵ_i^best)
6      Compute new path P^2 for w_i^new
7      if t(P^1) − t(P^2) > time_saving then
8          time_saving = t(P^1) − t(P^2)
9          l* = l_i
10 return l*
```

Obtaining feedback about a task (s_1, g_1) potentially affects the learning for another task (s_2, g_2), as some weights corresponding to paths for (s_2, g_2) might no longer lie in the *feasible* set.

In the multiple task setting we additionally have to pick a start-goal pair for which we want to present new paths. We propose a simple policy for this in Algorithm 4. Let a learning instance l_i be the collection $(s_i, g_i, \boldsymbol{w}_i^{\text{best}})$ for a task i where $\boldsymbol{w}_i^{\text{best}}$ is the weight vector corresponding to the current best path. Further, let L be the set containing all l_i for all tasks in the scenario. Given L and the current feasible set described by $\boldsymbol{A}\boldsymbol{w} \leq \boldsymbol{b}$, the algorithm iterates over all l_i and computes a new alternative path with the `minVertex` policy (line 4). Then, it selects the task, i.e., start-goal pair, where the time difference between the current best path and the tentative alternative is maximized (line 7). As a result the user is usually presented with those tasks for which the alternatives consist of very different paths in the first few iterations. After some user feedback is obtained, fewer paths are feasible and the respective weights are less different. Hence, in later iterations the two paths presented to the user become more similar.

Remark. The evaluation of Algorithm 4 can take significant computation time in practice. We can approximate the selection of l_i by sampling a random subset L' of L and iterate over the elements in L' in line 2 of the algorithm. Further improvements can be achieved by parallel execution of the for-loop.

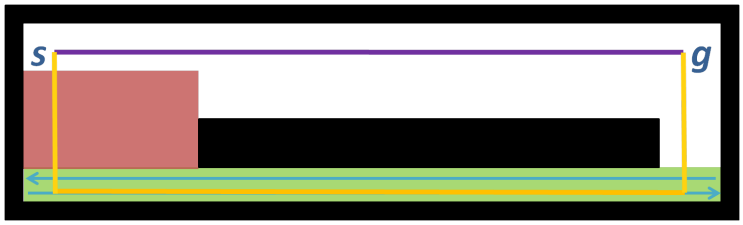


Figure 5.2: Example for increase in task completion time. The initial specification results in path P^{init}, shown in purple. An alternative solution (yellow) might have a longer traversal time, but correspond better to the user preferences if they value the avoid zone as less important and prefer the use of the road.

Impact of learning on performance Note that the interactive learning does not guarantee improvements in the completion time of paths. Users accept alternative paths if they have a lower cost with respect to equation (5.1), which does not necessarily imply a lower time. Consider the simple example in Figure 5.2. Following the specification leads to the direct path P^{init}, shown in purple. However, a possible alternative enters into the avoid zone (shown in red), but also traverses along a user specified road (shown in green). The alternative path (shown in yellow) might have a lower cost with respect to the user preference. This effect becomes especially relevant in the multi-task setting due to *passive learning*: Obtaining feedback for paths between some s_1 and g_1 adds inequality constraints to the *feasible set*, which then affects the optimal path between s_2 and g_2. Relating back to the example from Figure 5.2, the user might not be presented with these two paths as the learning system might infer about the importance of the avoid zone from a different task. However, in the results of the user study we analyze the performance in detail and show that in practice, the interactive learning improves the performance for most users.

5.2.4 Metrics[2]

To quantitatively measure the quality of specifications, two metrics are employed: *Entropy Ratio* and *Time Ratio*. Similar metrics were originally introduced and applied to user specifications in [13].

Entropy Ratio Given a graph G' and a specification Γ, the entropy quantifies the complexity of the robot's action space, generated by the combination of the environment and

[2]The research in this section was conducted entirely by the collaborating student Alexandru Blidaru.

user specification, by considering the number of outgoing edges available at each node, taking their cost into account. The entropy ratio is expressed as the ratio of entropies between graph G and graph G' (See Section 5.2), i.e., the constrained and unconstrained environment. We measure complexity using entropy, defined similarly to previous work in [13] and [5]. Given an estimate $\hat{\boldsymbol{w}}$ of the user weights, let the cost of an edge be the sum of time and the estimated weight: $\hat{c}(e) = t(e) + \hat{w}(e)$. Further, let $\hat{c}^{\min}(v_i, v_j)$ be the minimal cost between all parallel edges from v_i to v_j.

For a given vertex v_i on a graph and the set of its neighbours $\mathcal{N}(v_i)$, the entropy of v_i is given by

$$H(v_i) = -\sum_{v_j \in \mathcal{N}(v_i)} p(v_i, v_j) \log_2 p(v_i, v_j), \tag{5.4}$$

where we define $p(v_i, v_j)$ as

$$p(v_i, v_j) = \frac{1/\hat{c}^{\min}(v_i, v_j)}{\sum_{k, v_k \in \mathcal{N}(v_i)} 1/\hat{c}^{\min}(v_i, v_k)}. \tag{5.5}$$

To obtain the entropy of a graph, we take the sum over the individual vertex entropies:

$$H_G = \sum_{v_i \in V} H(v_i), \tag{5.6}$$

where V is the set of vertices of a graph G, and H_G is the entropy of a graph. The entropy ratio is then denoted by $\eta = H_G / H_{G'}$. The entropy is maximized for $H_{G'}$, when there are no user specifications the robot can move freely in any obstacle-free regions of the environment. Adding constraints always decreases entropy as the robot's movement becomes more restricted. Thus, small entropy ratios indicate rigorous specifications where the robot behaves in a more predictable way.

Time Ratio Given a graph G', a specification Γ and a set of start and goal pairs V' where each start and goal is a vertex on G', the *time ratio* metric describes the effect of the constraints Γ on the average duration of the shortest paths with respect to equation (5.1), i.e., the ratio between the average optimal path durations in graph G and in graph G'. Thereby, paths for all pairs in V' are considered. Similarly to our previous work [13], we distinguish two forms of the metric: The *global* time ratio considers all vertices on the graph, i.e., $V' = V \times V$ where V is the set of vertices on G', while the *task* time ratio considers only a defined set of start and goal pairs, i.e., $V' = \{(s_1, g_1), (s_2, g_2), \dots\}$.

5.3 User Study

In this section we detail the study scenario and procedure and propose our main hypotheses.

5.3.1 Scenario

The study scenario describes a simulated industrial environment adapted from the layout of a real world facility. An autonomous mobile robot is required to fulfill material transport tasks; a single task consists of navigating from a given start to a given goal location.

Users are provided with a description of the environment detailing different zones as illustrated in Figure 5.3. This includes areas with high pedestrian traffic, loading docks, storage space, robot parking and charging, dedicated human work and break areas and, an assembly line. The tasks consist of navigating between the labeled areas, for instance traversing from the robot charging zone to the upper end of the assembly line. The user is asked to generate a specification such that the robots are able to reach any of the areas, excluding the human break rooms. Further, robots are never allowed to cross through the assembly line.

Before starting the specification task, users receive an explanation of the traffic rules and instructions on how to use the interface (For more details see Section 5.3.2). One-way roads are described as encouraging the robot to traverse them in the direction of traffic and discouraging the robot from traversing in the opposite direction. Two-way roads function as two adjacent and opposing one-way roads. Areas of avoidance simply define a part of the environment where robot traffic is undesired, while speed-limits express that the robot is required to drive with a reduced speed in a specific area.

Finally, users are told that "the robot can navigate freely in the environment without any traffic rules" and has fundamental safety features such as obstacle avoidance. The traffic rules are "meant to guide the robot's behaviour" in a way preferable for the user, and users are free to specify as few or many rules as they deem necessary.

5.3.2 Procedure

Structure The study was approved by the Office of Research Ethics at the University of Waterloo. Each study session took approximately 1 hour. The study process is structured in three parts: overview and introduction, training and main study. In the first part, the scenario and the role of the participant are briefly explained. A video introduces the

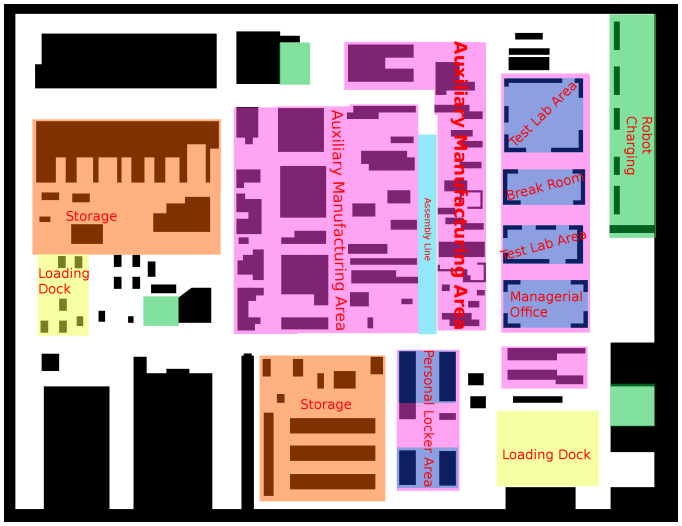


Figure 5.3: The scenario described in the study. Black corresponds to physical obstacles while white is the free space. The described areas in the environment are labeled as follows: High pedestrian traffic – purple, loading docks – yellow, storage – orange, robot parking and charging –green, human work and break areas – dark blue, assembly line – light blue.

user interface and demonstrates how to create traffic rules in detail. Further, written information about the traffic rules and the robot's capabilities is provided.

In the training phase the objective is to familiarize participants with the interface. They are presented with a smaller example environment including a similar description as in Figure 5.3 and are asked to create traffic rules until they feel confident in using the interface. At the end of the training, participants tele-operate the robot in the simulated environment.

The main study has three phases: specification, interaction and tele-operation. The first two phases are illustrated in Figure 5.4. In the specification phase, participants are presented with the environment from Figure 5.3 and the written instructions on the traffic rules and the robot's capabilities. It is once more stated that the robot is required to navigate between all marked areas on the environment (with the exception of the dedicated human break areas) and that the robot is able to navigate without any traffic rules present. Participants are then asked to define the traffic rules they find appropriate to achieve the desired robot behaviour. Once they are satisfied with their set of traffic rules, the first phase is concluded.

In the interaction phase, users are iteratively presented with two alternative paths for a

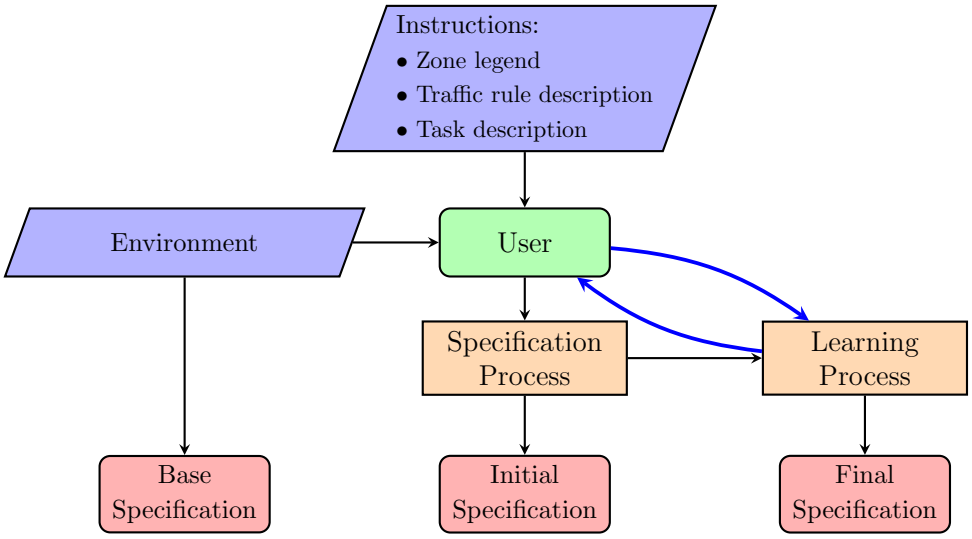


Figure 5.4: Flowchart of the study with the resulting specifications, black arrows are only executed once while blue arrows are executed multiple times. Participants initially receive an instruction set and a description of the environment. The environment yields a base specification, only including obstacles. Using the traffic rules they create the initial user specification for the robot. During the learning interaction users provide feedback, leading to a revised, final specification.

task. First, a brief instruction explains the interface: On the map of the environment both paths are shown simultaneously, a simple menu allows participants to select a path, which is then highlighted in color. Further, if a path is highlighted and violates a *penalty* constraint, the perimeter of the constraint in question is also highlighted. Finally, the menu features information about the duration of the two paths and lists the violated penalty constraints. All participants go through 20 iterations of the interaction process, unless the learning algorithm cannot find a new path for any of the tasks and terminates earlier. The task for which they are presented with two alternatives is selected by Algorithm 4 using an approximated set L' containing five tasks, sampled randomly.

In the last phase of the study, participants are asked to tele-operate the robot from a given start to a goal location. Thereby, they can choose freely if they follow their own traffic rules or violate them.

Questionnaires During the study, participants are also presented with several questionnaires. Before providing the specifications, users indicate their trust in the robot's capabilities to fulfill the described tasks. After each of the main steps users are presented with a questionnaire where they rate how well they specified the traffic rules and how confident they feel about using the system. Further, during each step of the interaction participants are asked how acceptable both paths were and, in every third iteration, what their reasoning for their choice was. Finally, the study concluded with a longer questionnaire where users evaluate the overall system. We use the standardized system usability score (SUS)[8] for evaluating the interface design and additional questions focusing on the warehouse scenario.

Evolution of specifications We define the specifications corresponding to different stages of the process, detailed in Figure 5.4. Before a user specification is provided, the environment including the obstacles yields a base specification where the optimal paths P_i^{base} for each task only minimize time. After receiving the traffic rules but before the learning we obtain the initial specification. The optimal paths P_i^{init} are the optimal paths corresponding to $\boldsymbol{w}^{\max}$, i.e., the paths that categorically follow the initial specification if possible. After learning, the algorithm has not necessarily converged to the optimal solution due to the limited number of iterations. Therefore, the feasible set might contain weights that are not equivalent to one another. In our learning model, all paths that are optimal for a feasible weight are equally good solutions, so the final path is not uniquely defined. Hence, we need to pick some $\boldsymbol{w}^{\text{final}}$ from the resulting *feasible set* $\mathcal{F}_{\text{final}}$ after learning (See Section 5.2.1). We propose a conservative approach for determining the final specification by choosing P_i^{final} to be the optimal paths for the maximum feasible weight, i.e., $\boldsymbol{w}^{\text{final}} = \underset{\boldsymbol{w} \in \mathcal{F}_{\text{final}}}{\arg\max} \{\mathbf{1}^T \cdot \boldsymbol{w}\}$.

Types of evaluation Further, we categorize different evaluations: Interaction-specific evaluation only considers tasks that were presented to the user during interaction. As we directly observed the user choosing between given pairs of paths, we have access to the path characteristics, i.e., time and violation, for each user choice. Task-specific and global evaluations are based on the metrics from the Definitions in Section 5.2.4. Task-specific evaluations consider the shortest paths for all tasks defined in the scenario, while global evaluations are based on the shortest paths between all pairs of vertices on the graph. As the interaction is limited to 20 iterations, not every user is necessarily presented

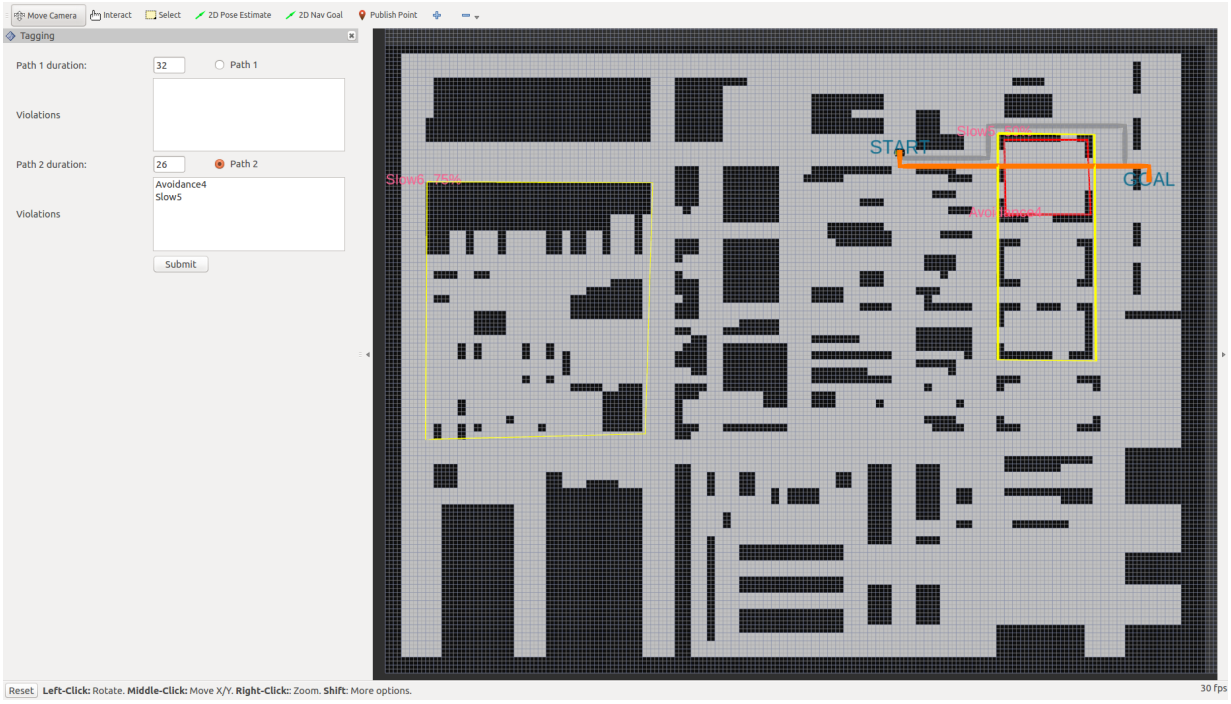


Figure 5.5: Preference learning UI displaying duration and traffic rule violations for each path. Additionally, the violated traffic rules are also highlighted on the map.

with alternative paths for all start-goal pairs. Therefore, task-specific changes between the initial and final specification might result from passive learning.

5.3.3 Interface Design

The interface employed in this study is an evolution of the one used in [13]. It allows users to create a set of robot constraints by defining polygons on the map of the environment. The interface allows for the creation of roads, areas of avoidance and reduced speed zones. Additionally, the interface also accommodates the interaction phase, as well as the embedding of study questionnaires that automatically pop-up at predetermined sections of the study. Integrating all the elements of system interaction in a single interface results in a more compact and easy to manage interface, which has been previously correlated to an increase in the resulting human-robot team performance [113, 94].

During each iteration of the interaction phase, the interface presents the user with a pair of labeled start and goal points, two alternative paths between these points, the time

duration of the two paths, and the constraints that each path violates. The interaction phase interface elements are illustrated in Figure 5.5. The violated constraints are shown both in list form, highlighted on the map.

In order to measure how users evaluate the system, they are asked several questions throughout the study. To minimize the effect that constant interruptions could potentially have on the users' performance, the questionnaires were integrated directly into the interface in the form of dialog boxes.

5.3.4 Hypotheses

Finally, we propose the two main hypotheses for the user study.

Hypothesis 1 (H1)**.** The learning process has the following properties: (a) user accept alternative paths that violate some of the constraints they specified over the course of the learning process and (b) the task performance improves through the interaction process.

Hypothesis 2 (H2)**.** Users find the specification process, and the interaction with the learning system intuitive and efficient.

Hypothesis 1 focuses on quantitative analysis of the user interaction. For (a) we analyze the user feedback from the interaction while (b) is based on the metrics described in Section 5.2.4. To validate Hypothesis 2 we conduct quantitative analyses based on the questionnaire, including the SUS score, as well as a qualitative analysis using the free form user comments.

5.4 Results

5.4.1 Participants

For the study we recruited 31 participants (21 male and 10 female) via mailing lists. In total 24 participants are affiliated with the Faculty of Engineering at the University of Waterloo. Moreover, 22 participants are pursuing or have completed a graduate education while 8 are pursuing or have completed an undergraduate degree. Finally, 6 participants stated that they have background knowledge in robot motion or urban planning.

The population of the study consists of two groups: 21 novice users and 10 repeat users. The novice users had never interacted with the presented framework before, while

Table 5.1: Characteristics of the initial user specifications. We show the individual mean, median, min and max for each type of constraint and the number of traffic rules. Further we report the characteristics of the overall smallest and largest specification as well as the example of participant 5 (P 05), shown in Figure 5.6.

	Roads	Avoidance	Speed	Traffic Rules
mean	15	4	8	21
median	13	5	7	21
[min, max]	[0, 40]	[1, 9]	[0, 19]	[10, 38]
EXAMPLES				
smallest	10	1	5	10
largest	39	1	13	38
P 05	21	1	17	31

the repeat users had previously used the system once (e.g., during the pilot phase of the study). No participant is part of both groups. In Sections 5.4.2–5.4.4 we present results for all users while Section 5.4.5 focuses on differences between the two groups.

5.4.2 Specifications

The initial specifications provided by the users vary in their complexity. We summarize the characteristics of the initial specifications in Table 5.1. Recall that the number of user defined roads does not correspond to the number of constraints for the planning problem as roads constitute a reward and a penalty constraint for each lane. We show three example specifications, the smallest and largest with respect to the number of traffic rules as well as the specification from participant 5 that is illustrated in Figure 5.6.

5.4.3 Hypothesis 1

(a) Acceptance of alternative paths For each user we define $\alpha^j_{\texttt{all}}$ to be the percent of iterations in which user j accepted the alternative path. Further, we introduce $\alpha^j_{\texttt{tasks}}$ as the percentage of the tasks presented to the user where user j accepted at least one alternative, i.e., where they rejected the initial path at some point.

Overall 30 out of the 31 participants accepted at least one alternative path. On average we found that $\alpha_{\texttt{all}} = 0.44$, meaning that users accepted alternatives in 44% of the

Figure 5.6: Example specification from participant 5. Reduced speed rules are marked in yellow, road rules in green, and avoidance rules in red.

interactions. The task related acceptance has a mean of $\alpha_{\texttt{tasks}} = 0.62$; thus, for roughly 2 out of 3 tasks users preferred an alternative path over the initial one.

Further, we investigate the correlation of $\alpha_{\texttt{tasks}}$ and the richness of user specifications. We characterize the richness of a specification in two ways: The number of traffic rules that the user defined and the number of resulting constraints for the planner. We found that the Spearman rank correlation of the number of traffic rules and the acceptance rate is 0.51 while the correlation of constraints and acceptance is 0.60, both with a confidence of $p < 0.005$. This corresponds to a moderate correlation, indicating that users who define a larger set of constraints are more likely to accept alternative paths.

Together with the task specific acceptance rate of $\alpha_{\texttt{tasks}} = 0.62$ we find strong support for our first hypothesis: Users are unaware of the impact of their specification and thus allow robots to violate traffic rules (or use roads less frequently) when presented with different possible solutions. Further, the obtained revised specification leads to paths where users chose an alternative path over the initial one in 62% of cases. Moreover, the correlation of complexity and acceptance shows that this effect becomes more apparent for users defining many traffic rules. In Chapter 4 we postulated three types of users for the simulations: A low trust user with many constraints for which the importance varies drastically, a high trust user with few constraints that all are relatively important and an

intermediate user. In the user study we do not observe a discrete separation but rather a continuous distribution for the user behavior. From the difference in the correlation we can conclude that users defining many traffic rules are more likely to accept deviations from the initial path.

(b) Increased performance To evaluate the changes in the performance, we compare the *time ratio* metric of the initial and the final specification, illustrated with violin plots in Figure 5.7. Further, we compare the metric for global and task-dependent evaluation.

For both evaluations we observe a decrease in the time ratio after the learning process as well as a reduction in the standard deviation. A paired-samples one-sided t-test was conducted on the task time ratio between initial and final specifications. The task time ratio of initial specifications ($M = 1.81,\ SD = 0.43$) was found to be significantly different ($p < 0.01$) from that of final specifications ($M = 1.55,\ SD = 0.22$).

Unsurprisingly, the initial specifications vary largely in their impact on the performance as the number of traffic rules users defined range from 10 to 38. However, the decrease in the population standard deviation following interaction indicates that the learning reduces the variation in the performance impact of user input and thus helps users to create more efficient task specifications.

Further, we notice that the task-dependent time ratios are higher than the global ones for the initial and final specifications. The global metric takes into account locations that are less relevant in the scenario, e.g., the lower left corner of the environment (shown in Figure 5.6) is not part of a robot task and therefore neglected by most users. Moreover, as the global evaluation considers all vertices on the graph, many close-by pairs of vertices are considered, where the specification often has little influence. While the task specific performance is worse, the relative change in time ratio is higher in the task specific case (14.4%) compared to the global metric (11.8%). Hence, the learning effectively improves the performance of the tasks in the scenario.

Figure 5.8 illustrates the change of the task-dependent time ratio and the entropy ratio for all user specifications. From the plot, we observe that while the time ratio decreases the entropy ratio increases. Entropy corresponds to how predictably the robot behaves. It captures the robot's degree of freedom with respect to the edge cost on G, which is the sum of time and user weight. As the learning process initially assumes high weights on the constraints and thus only reduces weights after obtaining feedback, the entropy increases. After learning, the robot might be allowed to violate some constraints, which enables more options to navigate in the environment. This leads to fewer restrictions on robot behavior,

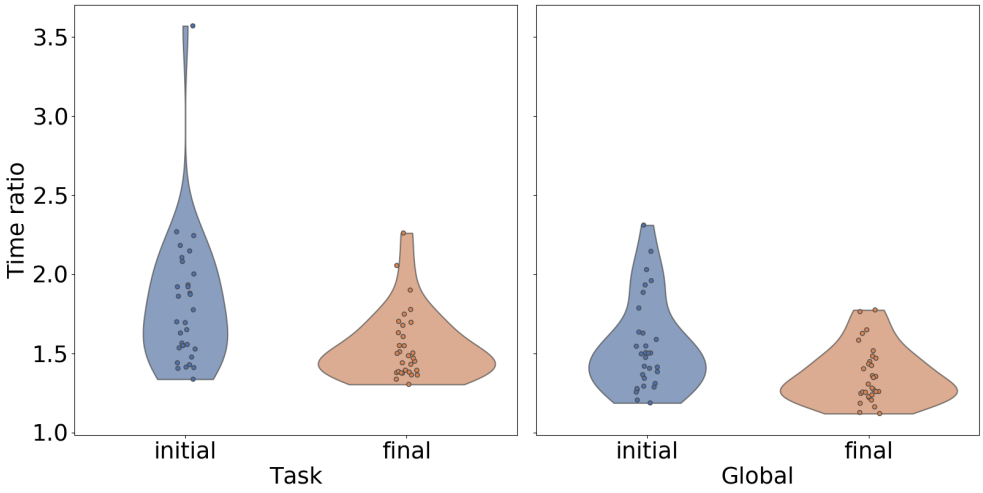


Figure 5.7: Change in the task-dependent and global time ratio metric of the specification due to active learning. In both plots the left bar shows the time ratio of the initial specification, averaged over all users. The right bar illustrates the time ratio of the final specification, also averaged over all users.

which may not always be desirable. However, this relaxation of the specification is traded-off with the increase in performance.

Running a paired-samples one-sided t-test, we found the entropy ratio of initial specifications ($M = 0.881,\ SD = 0.064$) to be significantly different ($p < 0.01$) from the entropy ratio of final specifications ($M = 0.9142,\ SD = 0.046$). Moreover, we notice that while the mean entropy increases, the standard deviations of the time and entropy ratios decrease due to the learning. With respect to the metrics, the specifications become more similar during the learning. Two sample f-tests between the initial and final time ratios show a significant difference ($p < 0.01$) in the variances, both in the case of global and task versions of the metric. However, no significant difference in the variances was found when performing the two sample f-test between the initial and final entropy ratios of the specifications. Additionally, Figure 5.8 also suggests that the specifications with low initial values of entropy and high initial task time ratios generally see more improvement following the preference learning process. This is verified by a strong Pearson correlation between the initial values and the difference between the final and initial values, resulting in $\rho = -0.88$ ($p < 0.01$) for time ratio, and $\rho = -0.85$ ($p < 0.01$) in the case of entropy ratio.

In summary, the learning system leads to a significant improvement in the time ratio metric, especially when measured for the tasks in the scenario. Further, the learning

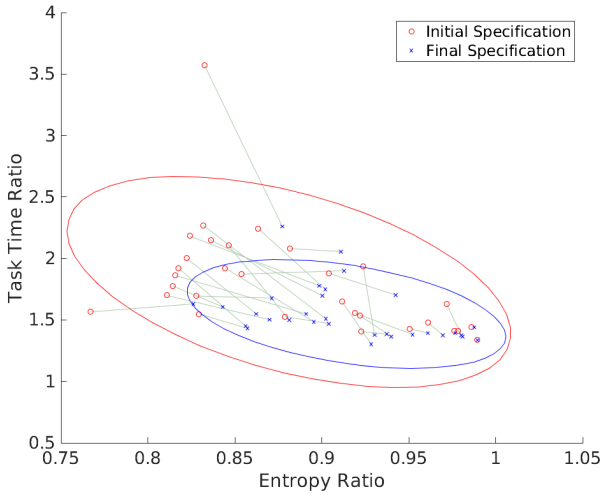


Figure 5.8: Change in the task time ratio and entropy ratio metrics of the specifications due to active learning. Red indicates the metrics for the initial specification, blue shows the metrics for the final one, and the lines associate the initial and final specifications of each individual users. The elipses represent the 95% confidence intervals.

revision reduces the variance between the performance of specifications. Moreover, specifications that are initially more inefficient benefit more from the learning process.

5.4.4 Hypothesis 2

We now report on the users' assessment of the usability of our framework, based on the system usability score (SUS). While the SUS does not provide a grade for the usability itself, the work of [8] provides a reference frame based on $2,324$ surveys using the SUS. In our study, users gave a mean SUS score of 69 while the median is 75. The difference arises from two outliers in the data set with a difference from the mean of over 2 and over 3 standard deviations. The mean corresponds to the second highest quarter of all surveys examined in [8]. Specifically for computer based GUIs, [8] reports a mean of SUS of 75.

After the three main parts of the study – constraint specification, learning interaction and tele-operation – participants were asked to asses how well they specified the robot behaviour on a 1 to 10 scale. On average, users reported similar ratings at each step, varying between 7.5 and 7.9, with standard deviations between 1.1 and 1.6. Hence, users felt relatively confident about how they used the framework. Interestingly, we observe

an inverse correlation (Spearman coefficient -0.65, $p < 0.01$) between the second self assessment and the richness of the specification, i.e., the number of constraints. Users defining a larger set of constraints tended to view their specification more critically after the learning. Thus, the interactive framework helps users to better understand the impact of their specification on the robot's performance.

5.4.5 Differences in the Population

Acceptance rates When splitting the data into the two populations, we observe only a minor increase in the acceptance rates for the novice users compared to the repeat ones. However, the correlation of acceptance rate and complexity of the specifications disappears for the repeat users while it is stronger for novices. Repeat users are more aware of the impact of their specifications while novice users benefit from the interaction to improve the robot's behaviour.

Time ratio metric Between novice and repeat users the *time ratio* metric varies. We recall that the task-dependent metric better reflects the effect of specifications on the task performance and thus we show the results for the task-dependent time ratio in Figure 5.9. We observe that the initial specifications provided by the novice users show a larger variance compared to repeat users. While the median values are relatively similar, the distribution for novice specifications spreads out to higher time ratios. However, the time ratios of final specifications are much more similar.

A two-sample one-tailed t-test was performed on the difference in time ratio between the initial and final specifications of novice and repeat users, which revealed that the two populations are significantly different ($p < 0.01$). In other words, the changes in the time ratios differ between the two groups.

We conclude that novice users create more diverse specifications with respect to the impact on performance. However, the learning process helps them to improve the specification and obtain better specifications. Repeat users seem to have a better understanding of the effect of the traffic rules and thus design specifications more carefully. Consequently, they allow for fewer violations that effectively render constraints insignificant and therefore obtain a smaller time benefit.

Entropy ratio Although small differences in the mean entropy ratios across the two populations were observed for both the initial (0.890 for novice and 0.870 for repeat) and

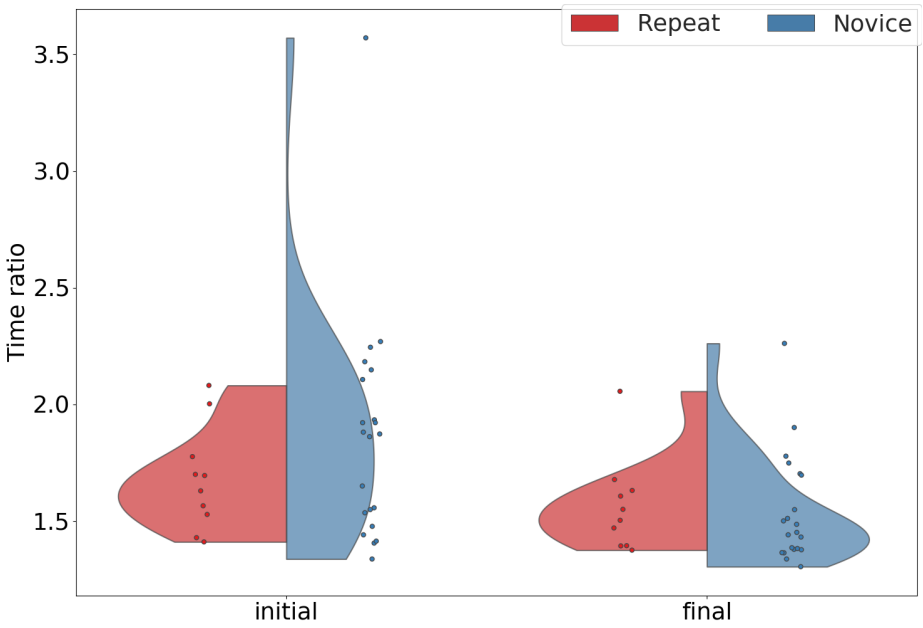


Figure 5.9: Change in the task-dependent time ratios of the specification, comparing novice and repeat users.

final (0.922 for novice and 0.901 for repeat) specifications, these differences were not found to be statistically significant.

SUS score The mean SUS of repeat users is 70 (median 77) while the mean of novice users is 68 (median 74). Naturally, participants who have interacted with the UI before are likely to find it more easy to use. Nonetheless, the reported difference is less than half of the standard deviation among all users scores and thus is not statistically significant. This supports our claim that the presented framework enables inexperienced users.

5.5 Discussion

In this chapter we investigated the practicality of the deterministic learning framework. In a user study we observed that all but one participant accepted alternative paths during the interaction and therefore the task-specific performance improve on average by 14%. Further, the specifications became more similar between users with respect to the time and entropy ratio. In general, users were generally positive regarding the usability of the

GUI. In this section we discuss additional findings in the study that do not directly relate to our introductory claims or the proposed hypotheses.

5.5.1 User Feedback

While most users ranked the interaction with our system as positive, participants provided several suggestions for improving the framework in the questionnaire feedback. Almost half the users expressed the desire to change their specification during the learning process. This indicates that although instructed on general robot behaviour, participants found it somewhat difficult to envision how all of the created traffic rules affect the behaviour of the robot. They could be well served by visualizations showing robot behaviour during the specification phase or by allowing them to change their specification later during the interaction.

Another aspect that could be improved is how the user feedback is incorporated into the learning. Currently users express their traffic rules preferences by selecting the preferred path. While this approach is intuitive and simple to use, users occasionally expressed frustration when both paths presented to a user contain undesirable behaviour, and so users have to select the lesser of two evils. As a result, future work should investigate additional forms of feedback that might better reflect a user's preference, and could potentially lead to a more efficient learning process. The work of [9] investigates richer forms of feedback in active preference learning. In addition to ask for the user's preference, feature queries give the user the opportunity to express their reasoning, i.e., "Which feature is most responsible for the difference in your preference between these two trajectories?". This aligns with feedback from the questionnaire in our study: Some participants stated that they rejected alternative paths as they violated both a minor and a major constraint at the same time; the violation of only the minor constraint would have been acceptable, but that was unknown to the learning system. In this case richer user feedback would help in two ways, allowing users to express the reasoning for their path selection, and potentially reducing the number of iterations of the learning. On the other hand, a drawback of this approach is the increased complexity of the interaction. A different form of richer feedback could allow users to manually indicate, and potentially correct, the undesired sections of presented paths. This idea is investigated by [27] where users segment a robot's trajectory into good and bad parts. In Chapter 9 we propose a framework for learning preferences when the user *corrects* the current path of the robot.

5.5.2 Repeat and Novel Users

In Section 5.4.5 we have shown that specifications originating from novice and repeat users differ in the time ratio metric. This indicates that there are differences in how the two groups of users specify the robot constraints, and that these metrics could be used to identify the expertise of a user based on the specification provided. In multi-user systems, this could be used to combine multiple specifications, emphasizing those of expert users. Despite the observed differences, the iterative preference learning system was shown to be capable of improving the specification performance of both types of users. This leads us to hypothesize that even in the case of specifications designed by domain experts, the learning framework could still be used to help increase specification performance. To further investigate this, we currently develop a web-based user interface for remote study participation, allowing us to recruit more domain experts, i.e., people working in industrial facilities.

5.5.3 Learning Framework

In Chapter 4 we evaluated the active learning framework in simulation. Validating the extended algorithm proposed here in the user study allows us to make additional observations about the practicality of the approach. Unlike the work of [87, 28, 40, 35], our learning framework is currently based on a deterministic user model. The major drawback is that our model does not consider users who behave differently than described in the assumed cost function. Nonetheless, we were able to demonstrate that using a simplified linear user model, the framework proposes alternative paths that users accept over the initial paths and revises the specification to improve the task performance within a small number of iterations. While the resulting final specification does not necessarily correspond to the optimal solution with respect to the hidden user preferences, the deterministic model allows for quick learning, yielding substantial improvements within only 20 iterations. A more complex, potentially probabilistic user model would make fewer assumptions about the user's behaviour and thus be more robust; however, usually at the cost of performance, i.e., the number of iterations required for learning in a comparable setting.

Further, due to the multitask scenario we were able to observe some inaccuracies in the user feedback with respect to our user model. When learning about a single task, the *feasible set* can never be empty as the algorithm stops when all feasible weights are equivalent. However, intersecting the feasible sets for different tasks can lead to an empty set. In that case, the user feedback to different tasks contradicts one another, assuming the linear cost function. Notice that an empty intersection of the feasible sets is not a

necessary but a sufficient condition for inaccurate user feedback. In the study we observe this phenomenon for a total of 4 out of the 31 users. A more expressive user model could be used to avoid this issue of converging to a suboptimal solution, however, a richer model is likely detrimental to the efficiency of the learning.

Chapter 6

Bayesian Preference Learning using Equivalence Regions

A version of this chapter was published in the IEEE Robotics and Automation Letters (RA-L) in 2019 [109].

We extend the theoretical work presented in Chapter 4 in two ways: First, the assumptions on the user are relaxed in order to capture more realistic behaviour. By considering noisy user feedback and introducing a probabilistic learning approach, we allow users to not always behave consistently with our user model. Second, based on our formulation of the problem as an optimization problem, i.e., a shortest path search on a graph, we further investigate equivalence regions for possible solutions. We propose a new learning algorithm that exploits the notion of equivalence regions, which are sets of constraint weights that are indistinguishable to the user. From this we obtain a greedy algorithm that allows for highly efficient learning, outperforming other state-of-the-art techniques [87, 9].

Previously, we proposed a deterministic user model for learning about weights from a ranking feedback and proposed a complete algorithm. Using the same framework for combining path planning with user constraints, we extend the user model. To capture user feedback inconsistent with our assumed cost function, we propose a Bayesian learning approach (Section 6.2.1). Thereby, we exploit the discrete properties of our problem, introducing a partitioning of the weight space based on equivalence regions. We prove almost sure convergence of the algorithm (Section 6.2.2) and derive a greedy approach (Section 6.2.3). Finally, we show the performance and robustness of our approach in comparison with another state-of-the-art technique in extensive simulations (Section 6.3).

6.1 Problem Formulation

As in Chapter 4 and 5, we are interested in solving the specification revision problem (Problem 3). In contrast to Chapter 5 we only consider a single task scenario; however, the presented algorithm could also be adapted to multi-task scenarios. The problem setup is summarized in Figure 4.2. From the environment and a set of user constraints we construct an initial specification for the robot. As users might allow the violation of some of their constraints for sufficient time benefit, we present them with alternative paths during the interaction and ask for feedback. From this feedback we learn the weights of the constraints and obtain a revised specification that corresponds to the user preferences.

As before, we consider a fully known, static environment, represented as a weighted strongly connected multigraph $G' = (V, E, \Psi, t)$. The weight t on the graph encodes the time a robot requires to traverse an edge. We use parallel edges with different times to model speed. A robot task consists of navigating from a start vertex v_{start} to a goal vertex v_{goal} on G'. On the environment, a user specifies a set Γ containing d constraints. Each constraint is a pair $(E_i, w_i^{\texttt{user}})$, where E_i is a subset of the edges of G' and $w_i^{\texttt{user}}$ is a hidden user cost for the constraint. To incorporate the user specification, we create a doubly weighted graph $G = (V, E, \Psi, t, w^{\texttt{user}})$. For each edge e in G the second weight $w^{\texttt{user}}(e)$ is defined as the sum of all $w_i^{\texttt{user}}$ that belong to a constraint containing e. The problem is to find a path from v_{start} to v_{goal} that minimizes the following objective:

$$\min_{P} \sum_{e \in P} w^{\texttt{user}}(e) + t(e). \tag{6.1}$$

The true user weights $w_i^{\texttt{user}}$ are latent. Moreover, they are defined in units of time, allowing us to pose the multi-objective optimization as an unweighted sum. To learn about the weights, we can query the user by presenting them with a set of paths $\{P^0, P^1, \dots P^k\}$. The feedback is a vector of length k representing a ranking, i.e., a partial ordering, of the presented paths. Without loss of generality we focus only on pair-wise comparisons, as the ranking of additional elements can be expressed with a set of pair-wise relations. This is also well motivated with respect to the user; ranking more than two alternatives might be unnecessarily challenging [49].

6.2 Probabilistic Learning

In this section we propose a probabilistic model of user behaviour where the user feedback may be noisy and thus not deterministic.

6.2.1 Bayesian Learning

Using definitions for Bayesian inference from [105] we set up a learning model for gaining information about the hidden (latent) parameter $\boldsymbol{w}$. We model the user weights to be positive and finite: $w_i \in [0, w^{\max}]$. The cost of a path P is $c(P) = \boldsymbol{\phi}\boldsymbol{w}^{\texttt{user}} + t$, where the violation vector $\boldsymbol{\phi}$ describes how many edges of each constraint are traversed by P, $\boldsymbol{w}^{\texttt{user}}$ is a column vector containing all latent user weights and t is the time to traverse P. From each user feedback for a pair (P^i, P^j) we can derive a hyperplane of the form $(\boldsymbol{\phi}^i - \boldsymbol{\phi}^j)\boldsymbol{w} = t^j - t^i$. This hyperplane defines two subsets of the weight space, Λ^{ij} and Λ^{ji}, where $\Lambda^{ij} = \{\boldsymbol{w} \in [0, w^{\max}]^d | (\boldsymbol{\phi}^i - \boldsymbol{\phi}^j)\boldsymbol{w} \leq t^j - t^i\}$. Thus Λ^{ij} is the set of all weights for which P^i has lower cost than P^j.

Probabilities of halfpspaces For any pair of paths (P^i, P^j), the parameter $\boldsymbol{w}^{\texttt{user}} \in \Lambda^{ij}$ holds iff $c(P^i) \leq c(P^j)$. Adopting a Bayesian perspective, we treat $\boldsymbol{w}^{\texttt{user}} \in \Lambda^{ij}$ as a random variable and assign an uninformed prior $\mathbb{P}(\boldsymbol{w}^{\texttt{user}} \in \Lambda^{ij}) = 1/2$. Notice that the volumes of Λ^{ij} and Λ^{ji} do not correspond to the probability of a path being preferred over another path. From the user feedback about two paths P^i and P^j we obtain binary observations. We denote observations with a random variable U^{ij}, indicating whether the user prefers path P^i or P^j. A deterministic user always provides feedback where $U^{ij} = 1 \iff c(P^i) \leq c(P^j)$, i.e., $U^{ij} = 1 \iff \boldsymbol{w}^{\texttt{user}} \in \Lambda^{ij}$. A probabilistic user is consistent with this model with some probability p^{ij}. Hence, the probability of U^{ij} given $\boldsymbol{w}^{\texttt{user}} \in \Lambda^{ij}$ is

$$\begin{aligned} \mathbb{P}(U^{ij} = 1 | \boldsymbol{w}^{\texttt{user}} \in \Lambda^{ij}) &= \quad p^{ij} \\ \mathbb{P}(U^{ij} = 1 | \boldsymbol{w}^{\texttt{user}} \notin \Lambda^{ij}) &= \quad 1 - p^{ij}. \end{aligned} \tag{6.2}$$

We refer to p^{ij} as the accuracy of the user and assume $p^{ij} > 1/2$, i.e., that our user model fits the user's decision making better than a random guess. If the parameter p^{ij} is hidden, we can evaluate equation (6.2) with an estimate $\hat{p}$. To simplify notation we write $\mathbb{P}(U^{ij} = 1)$ as $\mathbb{P}(U^{ij})$. In general, p^{ij} is a function of P^i and P^j. This allows us to model different levels of the user's accuracy depending on how similar the paths are.

Probabilities of equivalence regions Equation (6.2) describes an observation model for a pair of paths, assigning probabilities to halfspaces. We now assign probabilities to paths instead.

In Chapter 4 we introduced the notion of *equivalent* weights: If the same path is optimal for two weights $\boldsymbol{w}^i$ and $\boldsymbol{w}^j$, we call $\boldsymbol{w}^i$ and $\boldsymbol{w}^j$ *equivalent*. An *equivalence region* of a weight $\boldsymbol{w}^i$ is then the set of all weights that are *equivalent* to the weight: $\Omega(\boldsymbol{w}^i) = \{\boldsymbol{w}^j \in$

$\mathbb{R}^n_{\geq 0} | \boldsymbol{w}^j$ is *equivalent* to $\boldsymbol{w}^i\}$. We can use equivalence regions to discretize the weight space $[0, w^{\max}]^d$. Given a comparison of two paths (P^i, P^j), we introduce a second observation model that describes the probability of user feedback given that the true user weight $\boldsymbol{w}^{\texttt{user}}$ lies in the equivalence region Ω' of some path P' as

$$\mathbb{P}(U^{ij} | \boldsymbol{w}^{\texttt{user}} \in \Omega') = \begin{cases} p^{ij}, & \text{if } \Omega' \subseteq \Lambda^{ij} \\ 1 - p^{ij}, & \text{if } \Omega' \subseteq \Lambda^{ji} \\ 1/2, & \text{otherwise.} \end{cases} \tag{6.3}$$

If an equivalence region lies in both halfspaces Λ^{ij} and Λ^{ji}, we obtain no information from the feedback U^{ij}, since not all weights in Ω' are either feasible or infeasible with the user feedback; expressed in the third case. Let $\mathbb{O}$ be the set of all equivalence regions for a given problem instance. The observation model allows us to express a probability for $\boldsymbol{w}^{\texttt{user}} \in \Omega'$ given an observation U^{ij} as a Bayesian posterior

$$\mathbb{P}(\boldsymbol{w}^{\texttt{user}} \in \Omega' | U^{ij}) = \frac{\mathbb{P}(U^{ij} | \boldsymbol{w}^{\texttt{user}} \in \Omega') \mathbb{P}(\boldsymbol{w}^{\texttt{user}} \in \Omega')}{\sum_{\Omega \in \mathbb{O}} \mathbb{P}(U^{ij} | \boldsymbol{w}^{\texttt{user}} \in \Omega) \mathbb{P}(\boldsymbol{w}^{\texttt{user}} \in \Omega)}. \tag{6.4}$$

Following [105], we write the Bayesian posterior for a series of n observations U for arbitrary pairs of paths as

$$\mathbb{P}(\boldsymbol{w}^{\texttt{user}} \in \Omega' | U) = \frac{\prod_{U^{ij} \in U} \mathbb{P}(U^{ij} | \boldsymbol{w}^{\texttt{user}} \in \Omega') \mathbb{P}(\boldsymbol{w}^{\texttt{user}} \in \Omega')}{\sum_{\Omega \in \mathbb{O}} \prod_{U^{ij} \in U} \mathbb{P}(U^{ij} | \boldsymbol{w}^{\texttt{user}} \in \Omega) \mathbb{P}(\boldsymbol{w}^{\texttt{user}} \in \Omega)}. \tag{6.5}$$

Remark. Notice that this general model does not depend on the exact form of the likelihoods p^{ij}, we only require $p^{ij} > 1/2$. Therefore, our model could use the likelihood function from [9]. Alternatively, one could fix all p^{ij} to a constant. Then, in contrast to [87, 9], the accuracy of the user does not depend on the scaling of the features in the cost function. Moreover, our model increases the robustness towards user feedback that appears inaccurate because the user is considering context that is not described by our features. For instance, in a warehouse an operator might have different preferences for different weekdays or wants a robot to temporarily avoid certain regions. This can not be covered with the current cost function and thus this user would appear erratic to the learner. Finally, when the accuracy is set to one, the deterministic learning model from Chapter 4 is recovered. The key advantage of using equivalence regions in equation (6.5) is that it reduces the complexity of the probability distribution since we now have a discrete distribution over regions rather than a continuous one This allows for a significantly faster solving of the problem, as we will show in Section 6.3.

Algorithm 5: Probabilistic learning using equivalence regions with presampled paths.

Input: G', Γ, N
Output: $\hat{\boldsymbol{w}}^{\text{curr}}$

1 $\hat{\boldsymbol{w}}^{\text{curr}} = \boldsymbol{w}^{\max}$, $U = \emptyset$
2 Calculate $\mathbb{O}$
3 **for** $n = 1$ *to* N **do**
4 Update $\mathbb{P}(\boldsymbol{w}^{\texttt{user}} \in \Omega' | U)$ for all $\Omega' \in \mathbb{O}$
5 $P(\hat{\boldsymbol{w}}^{\text{new}}) \leftarrow \texttt{getNewPath}(\mathbb{O}, \hat{\boldsymbol{w}}^{\text{curr}}, \{\mathbb{P}(\boldsymbol{w}^{\texttt{user}} \in \Omega^1 | U), \mathbb{P}(\boldsymbol{w}^{\texttt{user}} \in \Omega^2 | U), \dots\})$
6 Get user feedback $U^{\text{curr,new}}$ for paths $P(\hat{\boldsymbol{w}}^{\text{curr}})$ and $P(\hat{\boldsymbol{w}}^{\text{new}})$
7 $U = U \cup U^{\text{curr,new}}$
8 **if** $U^{\text{curr,new}} = \text{new}$ **then**
9 $\hat{\boldsymbol{w}}^{\text{curr}} = \hat{\boldsymbol{w}}^{\text{new}}$
10 **return** $\hat{\boldsymbol{w}}^{\text{best}} = \underset{\boldsymbol{w}' | \Omega' \in \mathbb{O}}{\arg\max}\, \mathbb{P}(\boldsymbol{w}^{\texttt{user}} \in \Omega' | U)$

6.2.2 Probabilistic Learning Algorithm

In Algorithm 5 we propose a general procedure to iteratively learn about user preferences from pairwise user feedback with inaccurate users. Initially, we compute the set of all equivalence regions $\mathbb{O}$ (line 2). After updating our current belief about all equivalence regions (line 4), we iteratively generate new paths (line 5) similar to the deterministic algorithm from Chapter 4. Then, we request user feedback for the pair $(P^{\text{curr}}, P^{\text{new}})$ and add the user feedback to a set (6-7). After adding the new observation to our set, we update the weight space and, if necessary, the current weight (8-9). The procedure is repeated until we reach the iteration budget N in line 2, at the end we return the weight $\hat{\boldsymbol{w}}^{\text{best}}$ where the posterior belief is maximized. We discuss an implementation of the function $\texttt{getNewPath}(\cdot)$ in Section 6.2.3.

Convergence We now establish almost surely convergence of Algorithm 5. Let $\boldsymbol{w}^{\texttt{user}}$ be the true user weight and $p^{ij} > 0.5$ for all pairs of paths (P^i, P^j). Without loss of generality, we only consider i, j pairs that are ordered such that $c(P^i) \leq c(P^j)$; hence $\boldsymbol{w}^{\texttt{user}} \in \Lambda^{ij}$ always holds (but this is not known to the algorithm). Moreover, l is the number of equivalence regions in $\mathbb{O}$ and m the number of all pair-wise comparisons. For the following definition we change our notation and denote the optimal path $P^{\texttt{user}}$ as P^1.

Definition 6 (Asymptotically completely informative sequence)**.** Let X_n be a sequence of pairs of paths presented to the user in n iterations, and for each j, let $X^{1j} = \{X_1^{1r_1}, X_2^{1r_2}, \ldots, X_{n_j}^{1r_{n_j}}\}$ be the longest subsequence of X_n for which $\Omega^j \subseteq \Lambda^{1r_1}, \Omega^j \subseteq \Lambda^{1r_2}, \ldots, \Omega^j \subseteq \Lambda^{1r_{n_j}}$. Then the sequence is asymptotically completely informative if as n goes to ∞, we have $n_j \to \infty$ for all j.

In other words, if a sequence of pairs of paths contains observations about every $(P^{\texttt{user}}, P^j)$, and the number of observations for each $(P^{\texttt{user}}, P^j)$ pair goes to infinity as the length of the sequence goes to infinity, it is called *asymptotically completely informative*. Notice that such a sequence of paths is not required to contain subsequences for all pairs of paths $(P^{\texttt{user}}, P^j)$; it is sufficient if the feedback to the pairs $(P^{\texttt{user}}, P^j)$ contains information about all other equivalence regions according to (6.3). Finally, we call U *asymptotically completely informative* if the corresponding sequence of paths is *asymptotically completely informative* and treat the probability that the true user weight $\boldsymbol{w}^{\texttt{user}}$ or an estimate $\hat{\boldsymbol{w}}$ lie in an equivalence region Ω as a random variable.

Proposition 3 (Convergence)**.** Let $\Omega^{\texttt{user}}$ be the equivalence region containing $\boldsymbol{w}^{\texttt{user}}$. Given *asymptotically completely informative* user feedback U, the probability that the best estimate $\hat{\boldsymbol{w}}^{\text{best}}$ of Algorithm 1 lies in $\Omega^{\texttt{user}}$ converges almost surely to 1 as all n_j go to infinity:

$$\mathbb{P}(\hat{\boldsymbol{w}}^{\text{best}} \in \Omega^{\texttt{user}} | U) = 1, \; n_j \to \infty, \text{ for all } j. \tag{6.6}$$

Proof. At first consider the comparison of an arbitrary pair (P^i, P^j) and fix $P^i = P^{\texttt{user}}$. Let U be a sequence of user feedback of length n^{ij} and k^{ij} be the number of times the user chooses P^i, i.e., chooses accurately. Moreover, let $\hat{p}^{ij}$ be our estimate of p^{ij}. For simplification we drop the ij superscript. From $p > 1/2$, we can conclude that $k > n/2$ as $n \to \infty$, using Hoeffding's inequality [44]. We notice that the probability for a sequence of user feedback given $\boldsymbol{w}^{\texttt{user}} \in \Lambda^{ij}$ depends on p, while our belief about $\boldsymbol{w}^{\texttt{user}} \in \Lambda^{ij}$ given some user feedback is based on $\hat{p}$. Given user feedback U with known n and k, the posterior probability is

$$\begin{aligned} \mathbb{P}(\boldsymbol{w}^{\texttt{user}} \in \Lambda^{ij} | U) &= \frac{\hat{p}^k (1-\hat{p})^{n-k}}{\hat{p}^k (1-\hat{p})^{n-k} + \hat{p}^{n-k}(1-\hat{p})^k} \\ &= \frac{1}{1 + \frac{\hat{p}}{1-\hat{p}}^{n-2k}}. \end{aligned} \tag{6.7}$$

We take the limit as n goes to ∞:

$$\lim_{n\to\infty} \mathbb{P}(\boldsymbol{w}^{\texttt{user}} \in \Lambda^{ij} | U) = 1 = \lim_{n\to\infty} \frac{1}{1 + \frac{\hat{p}}{1-\hat{p}}^{n-2k}}. \tag{6.8}$$

Using $\hat{p} > 1/2$ leads to $\frac{\hat{p}}{1-\hat{p}} > 1$. The term $n - 2k$ is strictly negative if $k > n/2$. Hence, $\hat{p}/(1-\hat{p})^{n-2k}$ approaches zero as n goes to infinity. We conclude that $\lim_{n\to\infty} \mathbb{P}(\boldsymbol{w}^{\texttt{user}} \in \Lambda^{ij}|U) = 1$. As we only have two paths, we only have two equivalence regions. Following our ordering of i and j, $\Omega^{\texttt{user}} = \Lambda^{ij}$. Hence, $\mathbb{P}(\hat{\boldsymbol{w}} \in \Omega^{\texttt{user}}|U) = 1$, as $n^{ij} \to \infty$ for a single, fixed pair of paths P^i and P^j.
Finally, we extend the result to comparisons for multiple pairs. Equation (6.5) expresses the probability of $\boldsymbol{w}^{\texttt{user}}$ lying in a given equivalence region. Notice that $\bigcap_{j\neq i}\Lambda^{ij} \subseteq \Omega^{\texttt{user}}$, as well as $\lim_{n\to\infty} \mathbb{P}(\boldsymbol{w}^{\texttt{user}} \in \Lambda^{ij}|U) = 1$. As $\mathbb{P}(\boldsymbol{w}^{\text{best}} \in \Lambda^{ij}|U) \to 1$ for all $j \neq i$, we have $\mathbb{P}(\boldsymbol{w}^{\text{best}} \in \bigcap_{j\neq i}\Lambda^{ij}|U) \to 1$. Hence, $\mathbb{P}(\hat{\boldsymbol{w}}^{\text{best}} \in \Omega^{\texttt{user}}|U) \to 1$ and the statement holds. □

From Proposition 3 we conclude that Algorithm 5 always elicits the true user weight if an *asymptotically completely informative* sequence of paths is presented to the user for feedback, and if the user's accuracy with respect to our model is greater than 1/2. However this does not include any guarantees on the speed of convergence. In the next section we derive a greedy approach to maximize convergence speed.

6.2.3 Greedy Policy

We now show how to find new paths in each iteration of Algorithm 5, i.e., the function `getNewPath`. We notice that computing the set of all equivalence regions for a given problem instance is computationally intractable, as characterized in the following proposition.

Proposition 4 (Hardness of finding all paths)**.** Finding the number of non-*equivalent* paths between v_{start} and v_{goal} on the graph G is #P hard.

Proof. To prove the statement we reduce the S-T-paths problem to our problem. The S-T paths problem finds the number of all paths from a start to a goal vertex on an unweighted graph and is known to be #P complete [101]. Let (G^{ST}, S, T) be an instance of S-T-paths where $G^{ST} = (V^{ST}, E^{ST}, \Psi^{ST})$ is a non-weighted graph without parallel edges and Ψ^{ST} maps a pair of vertices to each edge. We construct an instance of our problem where $G' = (V, E, \Psi)$ with $V = V^{ST}$, $E = E^{ST}$, $\Psi = \Psi^{ST}$ and $v_{\text{start}} = S$ and $v_{\text{goal}} = T$. We choose t_i for all $e_i \in E$ such that G' is metric. We then pick a user specification consisting of $|E|$ user constraints, each defining a weight for exactly one edge such that each edge e_i in E is associated with a single hidden weight $w_i^{\texttt{user}}$. We obtain a doubly weighted graph $G = (V, E, \Psi, t, w^{\texttt{user}})$. Every path on G^{ST} then has a corresponding path on G and vice versa. Moreover, any path P between v_{start} and v_{goal} on G is a shortest path for some realization of all w_i as we can choose $w_i^{\texttt{user}} = 0$ if $e_i \in P$ and $w_j^{\texttt{user}} = \sum_k t_k$ for all $e_j \notin P$

and $k = 1, \ldots, |E|$ otherwise. Hence, the number of *equivalence* regions of our problem equals the number of all paths on G, which corresponds to the number of S-T paths on G^{ST}. We conclude that a solution to finding all equivalent paths solves S-T paths. □

The complexity class of #P includes problems such as counting the number of solutions of NP-hard problems; however, even for many problems solvable in polynomial time, counting the solutions is nonetheless #P hard [101]. If a polynomial time algorithm for solving a #P hard problem exists, it would imply P = NP. Furthermore, in Appendix A we study the sensitivity of shortest path problems on graphs, which is a generalization of equivalence regions. In addition to Proposition 4 we show that describing one single equivalence region is in general #P complete.

Nonetheless, we propose a greedy algorithm for finding new paths. Since the set of all paths can be of exponential size in relation to the number of constraints, this approach is only practicable for a an approximated set of paths $\mathbb{O}'$ where the size is bounded by a polynomial of the number of constraints. We define $q(\boldsymbol{w}^{\mathtt{user}} \in \Omega | U)$ as the unnormalized posterior, i.e., the numerator of equation (6.5). As q is not a probability we refer to it as the *posterior measure*. The decrease in the posterior measure is captured as

$$f(X_n) = 1 - \sum_{\Omega^i \in \mathbb{O}} q(\boldsymbol{w}^{\mathtt{user}} \in \Omega^i | U_n). \tag{6.9}$$

Our primary motivating application is one in which the user is "on-the-loop". We do not require them to constantly provide feedback and already execute the current solution P^{curr} (as in Chapter 4. Therefore, we keep the current best path and fix P^{curr} to be one of the two alternative paths comprising the next query (Algorithm 5). Thus, our greedy algorithm returns the path maximizing the posterior measure

$$P_n^{\text{new}} = \arg\max_{P^j} \quad \mathbb{E}\left[f\left((P^{\text{curr}}, P^j) \bigcup X_{n-1} \right) \right]. \tag{6.10}$$

In this optimization we only need to consider one P^j for each equivalence region. However, if we would present two new paths in each iteration, i.e., we do not fix one path in each query to be P^{curr}, it can be shown that equation (6.10) is an adaptive submodular function.

A similar approach is presented in the active learning framework of [87], where the objective is the reduction of the unnormalized integrated posterior of the weight space, referred to as the removed volume. The authors show that this volume removal function

is adaptive submodular. In contrast, equation (6.10) sums over the posterior measure of all equivalence regions. This indicates how the belief over paths changes instead of over all weights. When P^i is not fixed to be P^{curr}, our greedy objective function $f(X_n)$ can also be shown to be normalized, adaptive monotone, and adaptive submodular. Adaptive monotonicity follows from $q(\boldsymbol{w}^{\texttt{user}} \in \Omega|U)$ being multiplied with p^{ij}, $1-p^{ij}$ or $1/2$ when user feedback is observed. Further, adaptive submodularity follows since the marginal reward of an element X^{ij}, as defined in [34], is smaller for a set X_m than for a set X_n, where $|X_m| \geq |X_n|$ as the decrease in the posterior measure has an upper bound of $q(\boldsymbol{w}^{\texttt{user}} \in \Omega^i|U_n, U^{ij})$. Adaptive submularity provides strong performance guarantees for a greedy approach: At any given iteration, the greedy solution achieves $1 - 1/e \approx 0.63$ times the optimal solution and is the best polynomial time approximation [34].

Nonetheless, we can establish convergence for the greedy approach. For this, we use the notion of asymptotically completely informative sequences from Definition 6:

Lemma 3 (Convergence of the greedy algorithm)**.** The greedy algorithm equation (6.10) returns an *asymptotically completely informative* sequence of paths if the number of iterations goes to infinity and thus the probability of the true user weight converges to one almost surely.

Proof. To prove the statement we show two properties: 1) The greedy algorithm eventually returns $P^{\texttt{user}}$ and 2) if $P^{\text{curr}} = P^{\texttt{user}}$ it eventually returns all paths necessary to constitute an *asymptotically completely informative* sequence. To show the first statement let $P^{\text{curr}} \neq P^{\texttt{user}}$. If a path $P^j \neq P^{\texttt{user}}$ is returned we either do not learn about $P^{\texttt{user}}$ and the posterior of either Ω^{curr} or Ω^j decreases relatively to the posterior of $\Omega^{\texttt{user}}$ (see equation (6.3)). Otherwise, the comparison of (P^{curr}, P^j) contains information about $\Omega^{\texttt{user}}$ and thus is expected to increase the posterior of $\Omega^{\texttt{user}}$. While $P^{\text{curr}} \neq P^{\texttt{user}}$, the expected marginal reward of presenting $P^{\texttt{user}}$, i.e., $\mathcal{E}\left[f\left((P^{\text{curr}}, P^{\texttt{user}}) \bigcup X_{n-1}\right)\right] - f(X_{n-1})$, increases monotonically, relatively to the reward of any other path. Thus, $P^{\texttt{user}}$ will eventually be the maximizer of equation (6.10). Then, the greedy algorithm returns $P^{\texttt{user}}$ and the user will prefer $P^{\texttt{user}}$ over the current P^{curr} in expectation.

For the second statement, assume we already have $P^{\text{curr}} = P^{\texttt{user}}$. Due to inaccurate user feedback another path P^i becomes P^{curr}. However, as shown above, the algorithm eventually presents $P^{\texttt{user}}$ again. Consider the path P^j where n_j is minimal among all paths (n_j is defined in Definition 6). Case 1: $q(\boldsymbol{w}^{\texttt{user}} \in \Omega^j|U)$ is the maximizer of equation (6.10), thus P^j is presented and n_j increments. Case 2: Some $q(\boldsymbol{w}^{\texttt{user}} \in \Omega^r|U)$ is the maximizer and (P^{curr}, P^r) is presented. If the corresponding feedback contains information about P^j, n_j increments as well. According to equation (6.3), if no information about P^j is obtained,

$q(\boldsymbol{w}^{\texttt{user}} \in \Omega^j|U)$ increases by a ratio of $^{0.5}/_{(1-p)}$ (where p is the user accuracy) relative to all $q(\boldsymbol{w}^{\texttt{user}} \in \Omega'|U)$, where Ω' gets rejected by the feedback to (P^{curr}, P^r). As this holds for all other paths and the number of paths is finite, $q(\boldsymbol{w}^{\texttt{user}} \in \Omega^j|U)$ increases relative to all other posterior measures until, after a finite number of iterations, either information about P^j is obtained and n_j increments, or case 1 applies. Hence, we are guaranteed to increment the minimal n_j and thus all n_j must go to infinity. □

Performing an exact greedy step is hard, as finding the set of all equivalence regions $\mathbb{O}$ is intractable. In practice, a polynomial sized estimate $\mathbb{O}'$ can be found via sampling which allows for an approximate greedy step.

6.3 Evaluation

To generate realistic simulations, we recruited users to create specifications, using the same graphical user interface as employed in the user study in Chapter 5. Given the layout of a real industrial facility, users defined constraints as described in [13]. To systematically evaluate our approach, we simulate user feedback in the active learning. This allows us to pick different ground truths $\boldsymbol{w}^{\texttt{user}}$ and generate the user feedback with varying accuracy levels. Further, for an outdoor scenario we conducted experiments using graphs generated by a probabilistic method [60] and random specifications.

Our primary interest is how the posterior belief about $\boldsymbol{w}^{\texttt{user}}$ evolves. In the evaluation we have two objectives: Showing the robustness of our user model and comparing our work with [87]. We refer to their approach as the *Maximum Volume Removal (`MVR`)* and name our approach the *Maximum Equivalence Region Removal (`MERR`)* [1].

In our implementation of `MVR`, we modify the query selection: As we consider a discrete space, queries are found by iterating over $\mathbb{O}'$ rather then solving a continuous optimization problem. To ensure comparability with our framework, we fix one path in the pair that comprises a query as the current path. Moreover, `MVR` requires a scaling of the features. The user's accuracy is modelled as an exponential function of the difference in the cost of two paths [87]. Thus the user's accuracy depends on the scaling of the cost function. The model is extended by a linear parameter β in [9] to describe different levels of accuracy. However, as no restriction on the scale of the features is made, these β values do not yield similar results for different scenarios. In our experiments we manually determined β for each scenario such that the user's accuracy is approximately 0.9.

[1] Note: In both approaches neither volume nor equivalence regions are removed, we rather assign a lower posterior probability to the rejected items.

To investigate the performance we used three different specifications that vary in complexity. The first specification consists of 26 constraints, covering 33% of the free space, the second has 41 constraints covering 40% while the third consists of 52 constraints covering 73%[2]. For each specification we varied between two start and goal pairs and three randomly selected true user weights $\boldsymbol{w}^{\texttt{user}}$. As finding the set of all equivalence regions $\mathbb{O}$ is intractable, we generate estimates $\mathbb{O}'$ via sampling. The graph G' for these experiments is based on a grid layout and has of 5185 vertices. `MERR` and `MVR` differ in three components: The model for the simulated user feedback, the user model assumed by the learning system and the strategy for presenting new paths, i.e., the query selection.

6.3.1 Performance of `MERR` and `MVR` Query Selections

Experiment 1 In the first experiment, we compare the performance for the different active query selections - either `MVR` or `MERR` - for a user following the `MVR` model. Figure 6.1 shows the result for a total of 180 trials (10 repetitions for each configuration of user, start-goal pair and true user weight) with a budget of 30 iterations.

The left plot of Figure 6.1 illustrates the percentage of trials that achieved a given final posterior value for the true user weight $\boldsymbol{w}^{\texttt{user}}$ within the iteration budget. A critical threshold for the posterior is 0.5, as then $\hat{\boldsymbol{w}}^{\text{best}} = \boldsymbol{w}^{\texttt{user}}$ and $\hat{\boldsymbol{w}}^{\text{best}}$ is the unique maximizer of the posterior distribution. The `MERR` query selection has a higher success rate: 70% of the trials converge within 30 iterations, while only 40% do so for `MVR` queries. Interestingly, both approaches always converge once the posterior of $\boldsymbol{w}^{\texttt{user}}$ surpasses 0.5. In the right sub plot we illustrate the evolution of the median posterior of $\boldsymbol{w}^{\texttt{user}}$ over the iterations. Further, at iterations 10, 20 and 30 we show the distribution of the data. We observe that between iterations 10 and 20 the `MERR` posterior median starts to increase quickly, passing the 0.5 threshold at iteration 17 and getting past 0.9 after 21 iterations. `MVR` shows a slower increase and does not pass 0.2 within the 30 iterations. The violin plots at three stages of the process illustrate a further detail: The distributions are nearly bimodal. Both approaches succeed for some instances very quickly while the posterior stays low for harder instances. However, we observe that the low end of the distribution shrinks more quickly for `MERR` and eventually becomes completely bimodal. In some hard instances, the algorithm takes longer to initially show $P^{\texttt{user}}$ to the user and does not learn about $\boldsymbol{w}^{\texttt{user}}$ until then. However, once $P^{\texttt{user}}$ is shown, the belief about $\boldsymbol{w}^{\texttt{user}}$ is maximized quickly, leading to the bimodal distribution.

[2]When a user specifies a road on the interface it counts as 2 constraints for the planner: A reward for following the road, i.e., a constraint with a negative weight, and a penalty for going against the direction of travel.

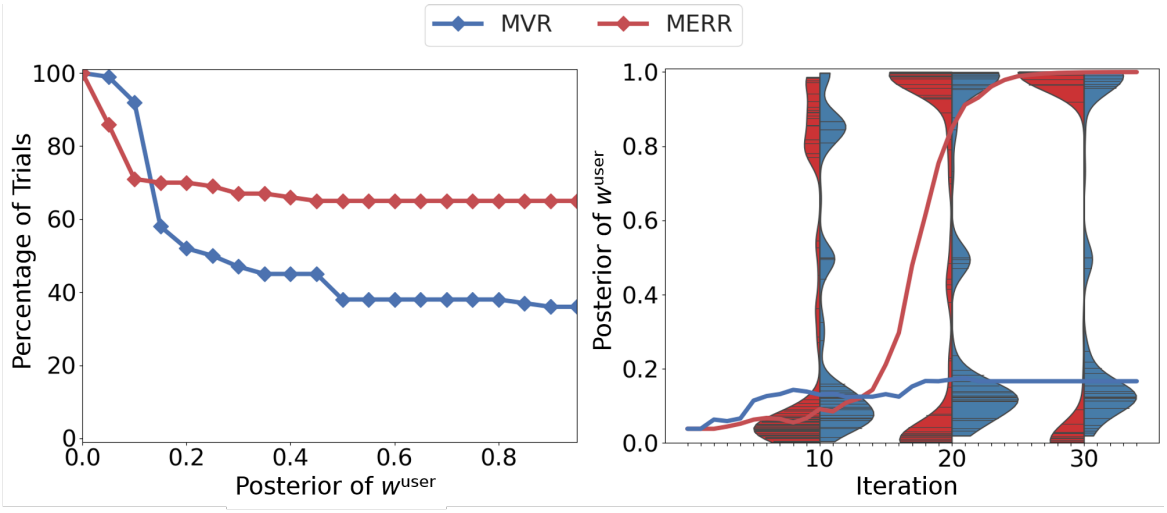


Figure 6.1: Results from experiment 1. For different values of the posterior of the optimal weight $\boldsymbol{w}^{\texttt{user}}$ the left plot shows the percentage of trials that achieved that value within 30 iterations. In the right plot we show median values of the posterior over all 30 iterations together with violin plots of the distribution of the posterior of $\boldsymbol{w}^{\texttt{user}}$ at iterations 10, 20 and 30.

To explain the better performance of `MERR`, we recall that equivalence regions vary drastically in volume. `MVR` often proposes queries that reduce the integral of the posterior but do not significantly change the posterior of equivalence regions and thus makes little progress.

Experiment 2 The second experiment focuses on the performance of both query selection methods, assuming the user generates responses according to the `MERR` model, i.e., equation (6.3). In Experiment 1 we simulated the user according to [87, 9]. In that model, the accuracy depends on how different the presented paths are and therefore is influenced by the query selection. In order to ensure comparability with our user model, we fixed the accuracy to the average of Experiment 1, thus $p = 0.9$. Additionally, a second dataset shows the results for lower accuracies of $p = 0.8$. Notice that the range for meaningful values is $(0.5, 1]$; users with $p = 0.5$ act completely independently of our model. Further, $p = 0.9$ corresponds to 10% misleading feedback, $p = 0.8$ doubles the error rate to 20% of cases. In Figure 6.2 we summarize the data as done for Experiment 1.

In contrast to Experiment 1, both query selection methods achieve a lower convergence

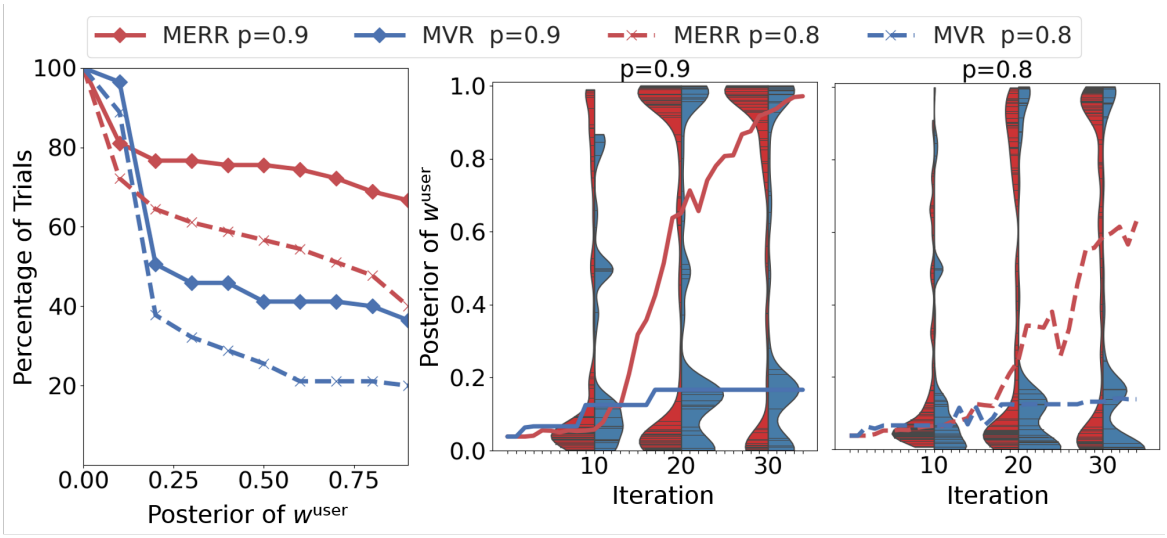


Figure 6.2: Results for experiment 2. The same analysis as in Figure 6.1 is shown, but for the MERR user two different values for accuracy, $p = 0.9$ and $p = 0.8$, are depicted.

rate for $p = 0.9$. MERR reaches a posterior median of 0.6 in 80% of the trials, while with MVR less than 45% of trials reach the 0.5 threshold. For the lower accuracy of $p = 0.8$, both query selection methods perform worse: MERR converges only in 42% and MVR 20% of cases. The graphs illustrating the mean posteriors over the iterations show a similar result for $p = 0.9$ compared to Experiment 1. For $p = 0.8$, the median of the posterior for MERR makes substantially better progress than MVR. The distributions confirm this observation: MERR shrinks the bottom lobe and gains on the upper end more quickly.

In summary, the first two experiments highlight the performance benefit when maximizing the decrease in the posterior summed over equivalence regions compared to the decrease in the integral of the posterior (i.e., the removed volume), irrespective of the user model.

6.3.2 Robustness of MERR

We further investigate how sensitive the proposed approach is to knowledge of the user's accuracy. We simulate the user according to the MERR model with a constant accuracy p, but the learning system only has access to an estimate $\hat{p}$. We fixed $p = 0.7$ and $p = 0.85$ and picked either $\hat{p} = p$, or $\hat{p} = p \pm 0.1$. The experiment is based on the smallest and

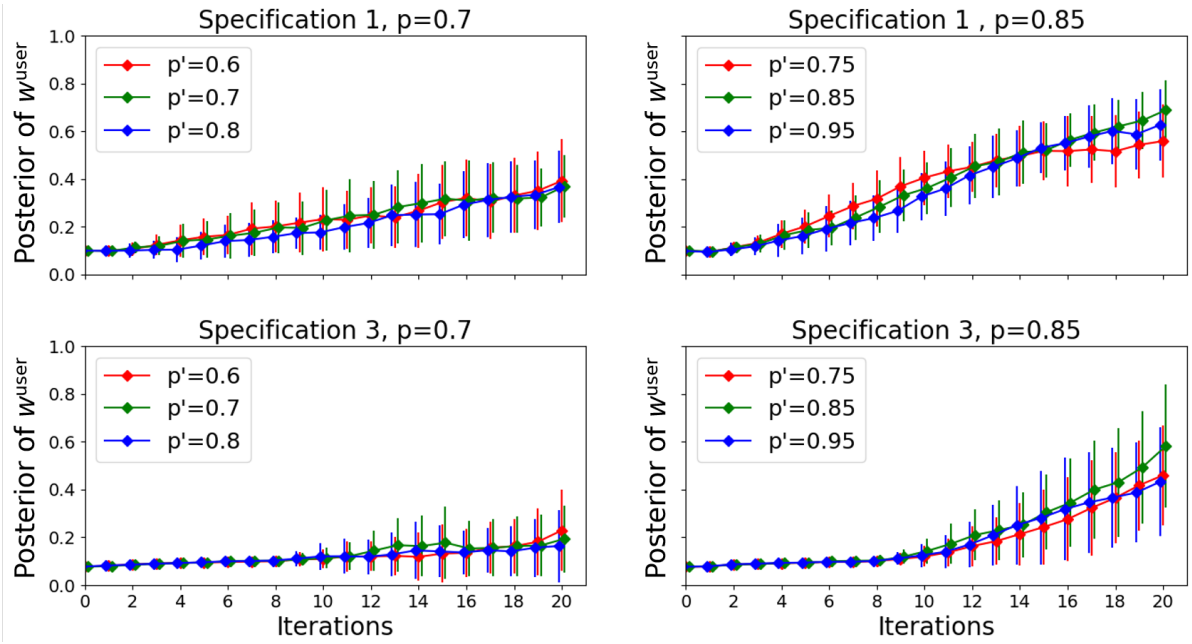


Figure 6.3: Experiment 3: Robustness of the MERR user model for two specification (26 and 52 constraints) and different accuracies p in the simulated user and different estimates p' for the learning.

largest specifications with 26 and 52 constraints, respectively. For each $p, \hat{p}$ configuration we average over 20 trials.

Figure 6.3 shows that an estimation error of 0.1 for the user accuracy has very little influence on the performance. In all 4 plots the over- and underestimates behave similarly to when the learner knows the user's accuracy exactly. As expected, the accuracy itself has an impact on the performance: For $p = 0.7$ the learning system performs worse for both specifications. Especially for the large specification, there is only little progress over the 20 iterations. In a more complicated setting additional feedback is needed to elicit the true user weight, the higher amount of inaccurate feedback then has a larger impact. On the other hand, for $p = 0.85$ the final result is relatively similar for both specifications. We conclude that a richer specification has a smaller impact on the performance when the user feedback is more accurate.

6.3.3 Extension to other Scenarios

Finally, we applied the approach to a different setting: An environment described by a graph based on a k-nearest probabilistic roadmap (PRM) [60]. User specifications are generated randomly by sampling polygons in the environment; for each region a constraint is formulated over all edges incident with a vertex in the region. Using the layout of the campus of the University of Waterloo, we generated a PRM graph with 2000 vertices and $k = 10$, the number of sampled constraints is 50. In Figure 6.4 we show the map together with the generated graph. Further, we compare the initial path of the learning which does not violate any constraints, and the user optimal path $P^{\texttt{user}}$ that is learned through interaction. We conducted a similar analysis as in Experiments 1 and 2, averaged for 20 different specifications. Overall 50% of the trials achieve a posterior of at least 0.9 within 50 iterations. Moreover, after 15 iterations the median passes the 0.5 threshold, while 0.9 is reached after 40 interactions.

6.4 Discussion

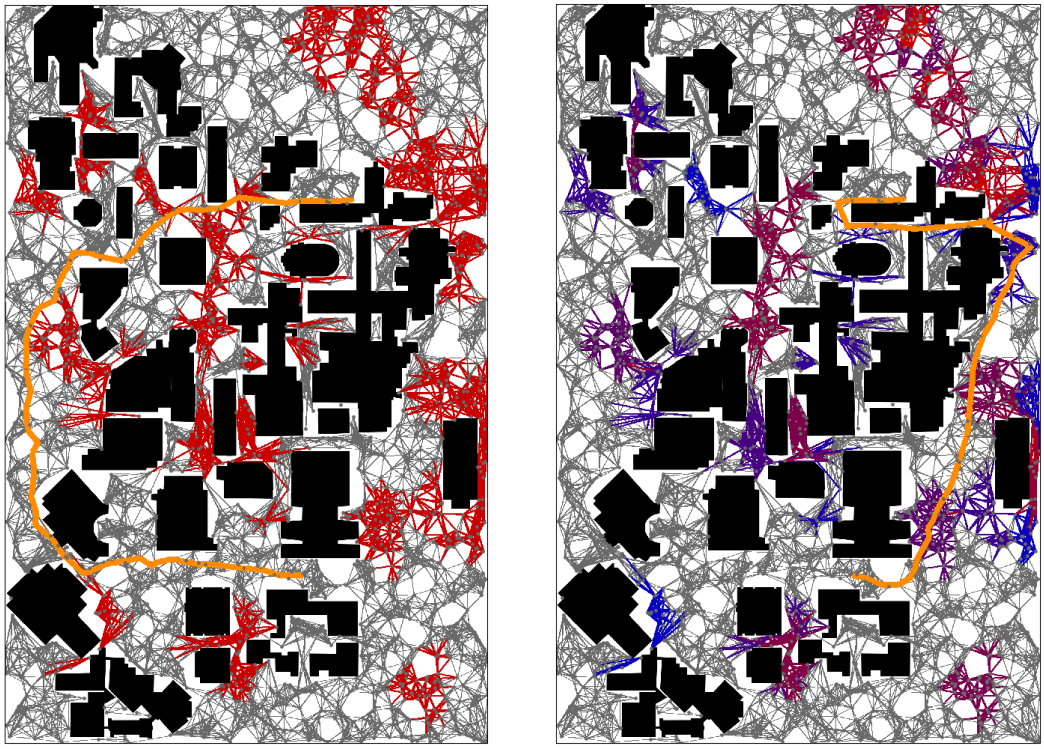
We presented an interactive framework for robot task specification. Based on Bayesian active leaning we derived a greedy algorithm for generating queries. Our approach exploits the fact that different weights for constraints do not necessarily lead to different optimal paths. Using equivalence regions allows for a discrete Bayesian learning model that does not require the user to always provide feedback consistent with the assumed cost function. The probability of an inconsistent user feedback is of a general form and scale-invariant. In simulations, we demonstrated that our approach outperforms a related state-of-the-art technique [87] and showed robustness of our user model.

User models Generally, both models, `MVR` and `MERR`, assume that the user evaluates paths based on a weighted sum of features. `MVR` uses model where the user's accuracy depends on how similar the presented paths are. This approach is well motivated and promises good performance when the features are adequate. On the other hand, it has disadvantages with the scaling of features and lacks robustness when users do not follow the model. In contrast, our approach generally models inaccuracies as a random noise and is agnostic towards their exact form. In the simulations we fixed p to a constant and demonstrated the robustness. Therefore, our learning system is less dependent on our user model exactly capturing the real user behaviour. A limitation of the `MERR` user model with constant accuracy values is that accurate user feedback is potentially not exploited

efficiently as the noise then is query-independent. This approach is investigated in [45]. Moreover, both learning models depend on sampling to perform the greedy step. Even though the accuracy can be increased arbitrarily with more samples, finding the optimal solution is computationally intractable.

Performance in experiments In the first two experiments we showed data that was collected for different user weights and different user specifications (and thus features). Both have a direct influence on the algorithm's performance. Usually, more complex specifications have a larger set of equivalence regions, which affects the convergence. Maybe surprisingly, the true user preference $\boldsymbol{w}^{\texttt{user}}$ can also influence the performance, especially in the `MERR` model: The learning system makes little progress if most of the hyperplanes learned from a sequence of user feedback intersect $\Omega^{\texttt{user}}$. Moreover, in single trials both models can perform relatively poorly by random chance. An inaccurate user feedback for a query containing $P^{\texttt{user}}$ leads to decrease in the posterior of $\boldsymbol{w}^{\texttt{user}}$. Then, the learning system might need multiple iterations to present $P^{\texttt{user}}$ again and thus elicit $\boldsymbol{w}^{\texttt{user}}$. This effect is enhanced when the accuracy of the user is low. Nonetheless, the `MERR` learning model is still guaranteed to converge. In Chapter 7 we propose a different way of measuring the quality of the learning: We consider the path that is optimal for the expected weight, and compute its relative error, compared to the optimal solution.

Extension to continuous space We studied an active preference learning technique based on equivalence regions, i.e, the discretization of the weight space into sets of weights where a motion planner returns the same path for all weights in each set. The concept of equivalence regions is based on the sensitivity of optimization problems and thus is not limited to discrete motion planners, i.e., graphs. However, for a continuous space planner the number of equivalence regions can become arbitrarily large while the regions themselves become very small. Thus, the presented algorithm might learn less efficiently. In an extreme case where all equivalence regions are singletons, the presented approach is equivalent to the maximum volume removal approach from [87]. The concept of equivalence regions could be better adapted to continuous spaces by introducing a notion of path similarity instead of strict equivalence. This is related to the work of [14], which proposed a formalism for probability distributions over a continuum of trajectories based on the trajectories' feature similarity.



(a) (b)

Figure 6.4: Results for Experiment 4. (a) shows the outdoor environment (buildings in black and freespace in white) with the generated PRM-graph. Red indicates edges that belong to a randomly generated constraint, orange shows the optimal path between a start and goal location of a task when not violating any constraints. (b) shows the optimal path $P^{\texttt{user}}$ and the updated weights on the constraints – red indicates high and blue low weights.

Chapter 7

Maximum Regret Learning

The work in this chapter was accepted for publication in the proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS) 2020 [110].

Thus far, we studied the specification revision problem (Problem 3) with deterministic and a probabilistic user models, showing strong results in simulations and demonstrating the practicality of the deterministic setting in a user study. Generally, the proposed techniques are not limited to Problem 3 and could be extended to different settings.

In this chapter we further investigate the dichotomy of active preference learning techniques that we discovered with the equivalence regions in Chapter 6: Learning about weights of the cost function, or learning about robot paths that are computed with these weights. We study two different measures for learning user preferences: 1) the *path error* which we introduced in our initial problem statement in equation (3.4), i.e., the error ratio between the learned and the optimal (hidden) path, and 2) the *weight alignment error* used in [87, 9, 20] which is described by the cosine of angle between the learned weights of the user cost function and the user optimal weights $\boldsymbol{w}^{\texttt{user}}$.

In Chapter 6 we compared a query selection method that purely learned about the weight space (Maximum Volume Removal) with our approach that groups weights for which the motion planner returns the same path (Maximum Equivalence Region Removal). We further build on our insights into the relationship of weights and paths to derive another novel query selection method that directly tries to remove worst case solutions, i.e., paths, from the belief space. Users are presented with the pair of solutions that has the maximum error ratio among all paths, discounted by the learned posterior belief. We demonstrate the performance of our approach by comparing it to a competing state of the art technique and show that our proposed method learns the desired behaviour more efficiently. Moreover,

the queries the user is presented with are easier to answer and thus lead to more reliable user feedback. Finally, we demonstrate how our measure based on solutions gives better predictions about the behaviour of the robot in different scenarios that were not part of the learning.

7.1 Problem Statement

Preliminaries We now study active preference learning in a more general setting, i.e., solving Problem 2, instead of the specification revision problem. Thus, there is no user specification and thereby features are no longer bound to user defined constraints. Furthermore, we do not pose our problem on a discrete space motion planner specifically, removing the motion planning graph from the problem statement. Therefore, we briefly refresh our mathematical conventions.

Let $\mathcal{X}$ be the state space of a robot and the environment it is acting in and x_0 some start state. Further, we have an action space $\mathcal{A}$ where each $a \in \mathcal{A}$ potentially only affect parts of the state, i.e., there might be static or dynamic obstacles unaffected by the robot's actions. Further let P be a path of finite length starting at x_0. A path is evaluated by a column vector of predefined features $\boldsymbol{\phi}(P) = [\phi_1(P), \phi_2(P), \ldots, \phi_d(P)]^T$. Together with a row vector of weights $\boldsymbol{w}$ we define the cost of a path as

$$c(P, \boldsymbol{w}) = \boldsymbol{\phi}(P)\boldsymbol{w}. \tag{7.1}$$

Given some weight $\boldsymbol{w}'$ let the *optimal path* be $P' = \arg\min_P c(P, \boldsymbol{w}')$. The optimal cost for a weight is

$$c^*(\boldsymbol{w}') = c(P', \boldsymbol{w}'). \tag{7.2}$$

For any other weight $\boldsymbol{w}$, we call $c(P', \boldsymbol{w})$ the cost of P' *evaluated* by $\boldsymbol{w}$.

Problem Formulation We consider a robot's state and action space $(\mathcal{X}, \mathcal{A})$ and some start state x_0. Further, let a vector of weights $\boldsymbol{w}^{\texttt{user}}$, describing a user's preference for the robot's behaviour and the corresponding optimal path $P^{\texttt{user}}$. Each weight $w_i^{\texttt{user}}$ has a lower and upper bound l_i and u_i. However, $\boldsymbol{w}^{\texttt{user}}$ itself is hidden. We can learn about $\boldsymbol{w}^{\texttt{user}}$ by presenting the user with pairs of paths (P, Q) over k iterations. The objective is to find an estimated path P^k that reflects the user preferences $\boldsymbol{w}^{\texttt{user}}$, i.e., is as similar to $P^{\texttt{user}}$ as possible.

To evaluate the result of learning $w^{\texttt{user}}$ the authors of [87] propose the *weight alignment metric*, i.e., the cosine of the angle between the learned weight vector $\boldsymbol{w}^k$ and $\boldsymbol{w}^{\texttt{user}}$. We

adapt this metric and transform it to a normalized error between 0 and 1, which we call the *weight error*:

$$\texttt{Err}_{\texttt{Weight}}(\boldsymbol{w}^k, \boldsymbol{w}^{\texttt{user}}) = \frac{1}{2}\left(1 - \frac{\boldsymbol{w} \cdot \boldsymbol{w}^{\texttt{user}}}{||\boldsymbol{w}^k||_2||\boldsymbol{w}^{\texttt{user}}||_2}\right). \tag{7.3}$$

The alignment metric was also used in [9, 20]. However, this metric has two potential shortcomings: 1) It does not consider the sensitivity of the optimization problem that finds an optimal path for a given weight vector. Thus, an error in $\texttt{Err}_{\texttt{Weight}}$ might actually not result in a different optimal path. Moreover, even if the learned weight has a relatively small error, the corresponding path might be suboptimal to the user. 2) The weight error is not suitable as a test error (i.e., to test whether the learned user preferences generalize well to new task instances not encountered during learning) since it does not consider the robot's resulting behaviour: $\texttt{Err}_{\texttt{Weight}}(\boldsymbol{w}, \boldsymbol{w}^{\texttt{user}})$ is equal for all training and test instances. Hence, the weight error gives no insight into how well the estimated preferences translate into different scenarios, unless $\texttt{Err}_{\texttt{Weight}}(\boldsymbol{w}, \boldsymbol{w}^{\texttt{user}}) = 0$, i.e,. the optimal weights are found. Therefore, we choose a different metric for evaluating the learned behaviour: Instead of the learned weight $\boldsymbol{w}^k$ we consider the learned path P^k. We compare the cost of P^k, *evaluated* by the user's true cost $\boldsymbol{w}^{\texttt{user}}$ to the optimal cost path of $\boldsymbol{w}^{\texttt{user}}$:

$$\texttt{Err}_{\texttt{Path}}(P^k, \boldsymbol{w}^{\texttt{user}}) = \frac{c(P^k, \boldsymbol{w}^{\texttt{user}})}{c^*(\boldsymbol{w}^{\texttt{user}})} - 1. \tag{7.4}$$

We refer to $\texttt{Err}_{\texttt{Path}}(P^k, \boldsymbol{w}^{\texttt{user}})$ as the *path error*, as proposed in Chapter 3. It is related to the spanning error of minimal control sets for state-lattice planners [15, 83]. Similarly, the authors of [18] use a policy error in risk-aware inverse reinforcement learning, the difference is that the error is a difference between the cost of the computed and the optimal solution instead of a ratio. Based on this metric we can now formally pose the learning problem.

Problem 4. Given $(\mathcal{X}, \mathcal{A})$ and x_0, and a user with hidden weights $\boldsymbol{w}^{\texttt{user}}$ who can be queried over k iterations about their preference between two paths P and Q, find a weight $\boldsymbol{w}^k$ with the corresponding optimal path P^k starting at x_0 that minimizes $\texttt{Err}_{\texttt{Path}}(P, \boldsymbol{w}^{\texttt{user}})$.

7.2 Active Preference Learning Framework

We introduce the user model and learning framework of our active preference learning approach and then discuss several approaches for selecting new solutions in each iteration.

7.2.1 User Model

To learn about $\boldsymbol{w}^{\texttt{user}}$ and thus find P', we can iteratively present the user with a pair of paths (P, Q) and they return the one they prefer:

$$\begin{aligned} c(P, \boldsymbol{w}^{\texttt{user}}) \leq c(Q, \boldsymbol{w}^{\texttt{user}}) &\implies \text{the user returns } P, \\ c(P, \boldsymbol{w}^{\texttt{user}}) > c(Q, \boldsymbol{w}^{\texttt{user}}) &\implies \text{the user returns } Q. \end{aligned} \tag{7.5}$$

However, a user might not always follow this model exactly. For instance, they might consider features that are not in the model or they are uncertain in their decision when P and Q are relatively similar. Thus, we extend equation (7.5) to a probabilistic model, similar to Chapter 6. Let $I^{P,Q}$ be a binary random variable where $I^{P,Q} = 1$ if the user prefers path P over Q, and -1 otherwise. Then we have

$$\begin{aligned} \mathbb{P}\left(I^{P,Q} = 1 | c(P, \boldsymbol{w}^{\texttt{user}}) \leq c(Q, \boldsymbol{w}^{\texttt{user}})\right) &= p, \\ \mathbb{P}\left(I^{P,Q} = -1 | c(P, \boldsymbol{w}^{\texttt{user}}) \leq c(Q, \boldsymbol{w}^{\texttt{user}})\right) &= 1 - p, \end{aligned} \tag{7.6}$$

where $1/2 < p \leq 1$. If $p = 1$ we recover the deterministic case from equation (7.5). In this very simple model the user's choice does not depend on how similar the paths P and Q are. In the experiments we simulate the user with the more complex model from [20], which poses the user's error rate as a function of the similarity between alternatives, and show that equation (7.6) nonetheless allows us to achieve strong performance.

7.2.2 Learning Framework

Over multiple iterations, equation (7.5) yields a collection of inequalities of the form $(\boldsymbol{\phi}^P - \boldsymbol{\phi}^Q)\boldsymbol{w} \leq 0$. We write the feedback obtained after k iterations as a sequence $U^k = \{(P^1, Q^1), (P^2, Q^2), \ldots, (P^k, Q^k)\}$. Without loss of generality, we assume that for any pair (P, Q) in U^k the path P was preferred over the path Q. We then summarize the left-hand-sides $(\boldsymbol{\phi}^P - \boldsymbol{\phi}^Q)$ for all k iterations using a matrix $\boldsymbol{A}^k$. Based on the sequence U^k we can compute an estimate $\boldsymbol{w}^k$ of $\boldsymbol{w}^{\texttt{user}}$ using a maximum likelihood estimator.

Deterministic case In the deterministic case, i.e., $p = 1$, the estimate $\boldsymbol{w}^k$ must satisfy $\boldsymbol{A}^k\boldsymbol{w}^k \leq 0$ to be consistent with the user feedback obtained thus far. The set of all such weights constitutes the *feasible set* $\mathcal{F} = \{\boldsymbol{w} \in \mathbb{R}^d | l_i \leq w_i \leq u_i,\ \boldsymbol{A}^k\boldsymbol{w} \leq 0\}$.

7.2.3 Active Query Selection

In active preference learning we can choose a pair of paths (P, Q) to present to the user in each iteration k. Throughout this work we only consider paths P and Q that are optimal for some weights $\boldsymbol{w}^P$ and $\boldsymbol{w}^Q$. Given the user feedback obtained until iteration k, a new pair (P^k, Q^k) is then found by maximizing some measure $f(\boldsymbol{w}^P, \boldsymbol{w}^Q, U^{k-1})$ describing the expected learning effect from showing (P^k, Q^k) to the user.

In the literature, several approaches have been discussed: Removing the *Volume*, i.e., minimizing the integral of the unnormalized posterior over the weights [87, 9], maximizing the information entropy over the weights [20] and removing *equivalence regions*, i.e., sets of weights where for each weight has the same optimal path [109].

Parameter space and solution space The first two approaches ([87, 9] and [20]) maximize information about the parameter space, i.e., the weights $\boldsymbol{w}$, instead of the solution space, i.e., the set of all possible paths P. Despite its motivation based on inverse reinforcement learning, this has a major drawback: The difference in the parameters does not map linearly to the difference in the features of corresponding optimal solutions. Given some $\boldsymbol{w}^P$ and $\boldsymbol{w}^Q$, we can compute optimal paths P and Q with features $\boldsymbol{\phi}^P$ and $\boldsymbol{\phi}^Q$, respectively. Then $||\boldsymbol{w}^P - \boldsymbol{w}^Q|| \propto ||\boldsymbol{\phi}^P - \boldsymbol{\phi}^Q||$ does *not* necessarily hold. Thus, learning efficiently about $\boldsymbol{w}$ does not guarantee efficient or effective learning about paths. Moreover, learning about $\boldsymbol{w}$ might allow for disregarding a large number of weights. However, the corresponding optimal paths might be very similar and thus the learning step is potentially less informative in the solution space.

Example 4. We consider the autonomous driving example from [20] which is posed in a continuous state and action space, illustrated in Figure 7.2. In Figure 7.1 we compare the weight error $\texttt{Err}_{\texttt{Weight}}(\boldsymbol{w}, \boldsymbol{w}^{\texttt{user}})$ and the path error $\texttt{Err}_{\texttt{Path}}(P, \boldsymbol{w}^{\texttt{user}})$ of 200 uniformly random samples, where every weight has a *unique* optimal path, i.e., there are no two weights where the corresponding optimal paths have identical features. While the weight error is distributed uniformly, the path error distribution takes nearly a discrete form, despite the continuous action space. This illustrates how different weights do not lead to different solutions, making the solution space more structured than the parameter space.

In our previous work [109] we proposed a query selection based on a discretization of the weight space: Sets of weights that have the same optimal path are labeled as *equivalence regions*. The objective then is to maximally reduce the posterior belief over

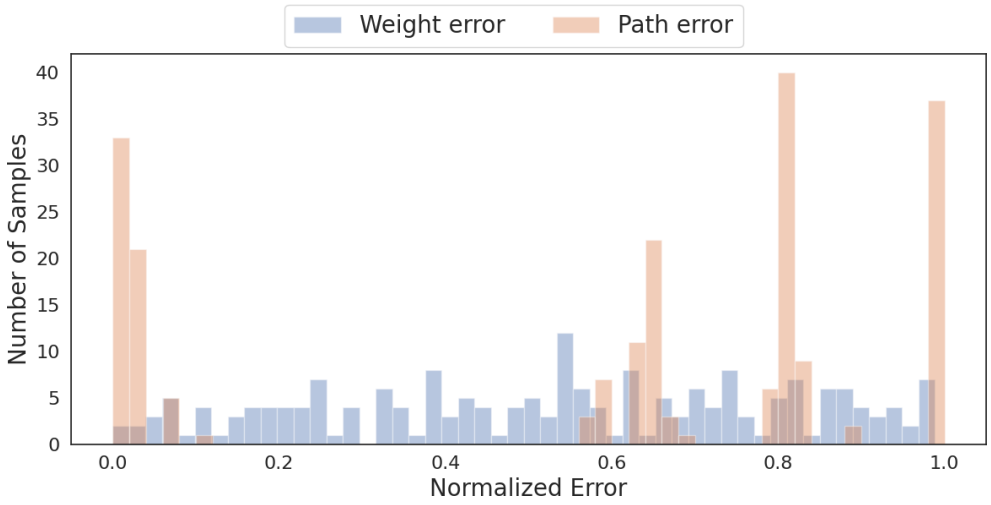


Figure 7.1: Example of the sensitivity of a continuous motion planning problem. We show the histogram of the normalized weight error $\texttt{Err}_{\texttt{Weight}}$ and the normalized path error $\texttt{Err}_{\texttt{Path}}$ for 3000 uniformly sampled random weights.

equivalence regions, i.e, to reject as many equivalence regions as possible. A drawback of this approach is that there exists cases where any query only allows for updating the belief of few equivalence regions, resulting in slow convergence. Because of these limitations of the existing approaches we study a new measure $f(\boldsymbol{w}^P, \boldsymbol{w}^Q, U^k)$ based on the solution space.

7.3 Min-Max Regret Learning

We now propose a new measure $f(\boldsymbol{w}^P, \boldsymbol{w}^Q, U^k)$ called the maximum regret, which we seek to minimize.

Definition 7 (Regret of a path relative to a weight)**.** Given a path P that is optimal for $\boldsymbol{w}^P$ and some weight $\boldsymbol{w}^Q$, the regret of P under $\boldsymbol{w}^Q$ is

$$r(\boldsymbol{w}^P, \boldsymbol{w}^Q) = \frac{c(P, \boldsymbol{w}^Q)}{c^*(\boldsymbol{w}^Q)}. \tag{7.7}$$

Regret expresses how sub-optimal a path P is when evaluated by some weights $\boldsymbol{w}^Q$. In active learning, this can be interpreted as follows: *If P is the final estimate, but Q is the*

optimal solution, how large is the ratio between the cost of P, evaluated by $\boldsymbol{w}^Q$, and the optimal cost? We now formulate an approach for selecting which alternatives to show to the user by using regret.

7.3.1 Deterministic Regret

When assuming a deterministic user, we need to assure that $\boldsymbol{w}^P \in \mathcal{F}^k$ and $\boldsymbol{w}^Q \in \mathcal{F}^k$, such that the presented paths reflect the user feedback obtained so far. Given $\boldsymbol{w}^P$ we pose the *Maximum Regret under Constraints Problem (MRuC)* as

Problem 5 (Maximum Regret under Constraints Problem (MRuC))**.** Given weights $\boldsymbol{w}^P = [w_1^P, w_2^P, \ldots, w_n^P]$ with a corresponding optimal path P, and a feasible set $\{\boldsymbol{w} \in \mathbb{R}^n | l_i \leq w_i \leq u_i,\ \boldsymbol{A}^k\boldsymbol{w} \leq \boldsymbol{0}\}$, find a weight that maximizes the following:

$$\begin{aligned} \max_{\boldsymbol{w}^Q}\ & r(\boldsymbol{w}^P, \boldsymbol{w}^Q) \\ s.t.\ & \boldsymbol{A}^k\boldsymbol{w}^Q \leq \boldsymbol{0}, \\ & l_i \leq w_i^Q \leq u_i. \end{aligned} \tag{7.8}$$

The objective can be written in the form $\max_{\boldsymbol{w}^Q, \boldsymbol{\phi}^Q} \boldsymbol{\phi}^Q\boldsymbol{w}^Q - c^*(\boldsymbol{w}^P)$. This is a bi-linear program, which are a generalization of quadratic programs. Unfortunately, in our case the objective function is non-convex; generally, such problems are hard to solve. In Appendix B we study the special case that paths are computed with a discrete motion planner and show that finding an exact solution is computationally intractable.

Symmetric Regret In equation (7.8) we have defined the maximum regret problem when one path is given. While presenting users with a new pair of paths (P, Q), we want to find paths where the regret of $\boldsymbol{w}^P$ under $\boldsymbol{w}^Q$ is maximized *and* vice versa. Thus, we rewrite the objective in (7.8) to $r(\boldsymbol{w}^P, \boldsymbol{w}^Q) + r(\boldsymbol{w}^Q, \boldsymbol{w}^P)$, which we call the *symmetric regret*. The maximum symmetric regret of a feasible set $\mathcal{F}^k$ can be found with the following bi-linear program:

$$\begin{aligned} \max_{\boldsymbol{w}^P, \boldsymbol{w}^Q}\ & r(\boldsymbol{w}^P, \boldsymbol{w}^Q) + r(\boldsymbol{w}^Q, \boldsymbol{w}^P) \\ s.t.\ & \boldsymbol{A}^k\boldsymbol{w}^P \leq \boldsymbol{0}, \\ & \boldsymbol{A}^k\boldsymbol{w}^Q \leq \boldsymbol{0}, \\ & l_i \leq w_i^P \leq u_i, \quad l_i \leq w_i^Q \leq u_i. \end{aligned} \tag{7.9}$$

Similar to equation (7.8) this is a non-convex optimization problem. In the evaluation we solve this problem by sampling a set of weights and pre-computing the corresponding optimal paths, following the approach in [87].

7.3.2 Probabilistic Regret

We now formulate regret with consideration of the user's uncertainty when choosing among paths. Taking a Bayesian perspective we treat $\boldsymbol{w}^{\texttt{user}}$ as a random vector. This allows us to express a posterior belief over $\boldsymbol{w}^{\texttt{user}}$ given an observation $I^{P,Q}$. Let $c^P = c(P, \boldsymbol{w}^{\texttt{user}})$ and $c^Q = c(Q, \boldsymbol{w}^{\texttt{user}})$, respectively. Further, we assume a uniform prior over $\boldsymbol{w}^{\texttt{user}}$. For any estimate $\boldsymbol{w}$ where $(\boldsymbol{\phi}^P - \boldsymbol{\phi}^Q)\boldsymbol{w} \leq 0$ we have

$$\mathbb{P}(\boldsymbol{w}^{\texttt{user}} = \boldsymbol{w} | I^{P,Q}) \propto \mathbb{P}\left(I^{P,Q} | c^P \leq c^Q\right). \tag{7.10}$$

Let $\mathbb{P}(\boldsymbol{w}|I^{P,Q})$ denote $\mathbb{P}((\boldsymbol{w}^{\texttt{user}} = \boldsymbol{w}|I^{P,Q} = 1)$. We calculate the posterior given a sequence of user feedback $U^k = \{(P^1, Q^1), (P^2, Q^2), \ldots, (P^{k-1}, Q^{k-1})\}$ as

$$\mathbb{P}(\boldsymbol{w}|U^k) \propto \prod_{(P,Q)\in U^k} \mathbb{P}(\boldsymbol{w}|I^{P,Q}). \tag{7.11}$$

We formulate the symmetric regret in the probabilistic case by weighting the regret by the posterior of $\boldsymbol{w}^P$ and $\boldsymbol{w}^Q$:

$$\begin{aligned} &\mathcal{R}^k(\boldsymbol{w}^P, \boldsymbol{w}^Q | U^k) \\ &\quad = \mathbb{P}(\boldsymbol{w}^P|U^k)\mathbb{P}(\boldsymbol{w}^Q|U^k)\left(\frac{c(P, \boldsymbol{w}^Q)}{c^*(\boldsymbol{w}^Q)} + \frac{c(Q, \boldsymbol{w}^P)}{c^*(\boldsymbol{w}^P)}\right). \end{aligned} \tag{7.12}$$

That is, we discount the *symmetric regret* such that we only consider pairs (P, Q) where both $\boldsymbol{w}^{\texttt{user}} = \boldsymbol{w}^P$ and $\boldsymbol{w}^{\texttt{user}} = \boldsymbol{w}^Q$ are likely given the user feedback U^k.

Finally, we adapt the problem of finding the maximum symmetric regret from equation (7.9) to the probabilistic case. As we cannot formulate a feasible set $\mathcal{F}$ for a probabilistic user, we consider a finite set Ω where each $\boldsymbol{w} \in \Omega$ is uniformly randomly sampled from the set $\{\boldsymbol{w} \in \mathbb{R}^d | l_i \leq w_i \leq u_i\}$. We then take the maximum over all $\boldsymbol{w} \in \Omega$ to compute the *probabilistic maximum regret*

$$\mathcal{R}^k_{\max}(U^k) = \max_{\boldsymbol{w}^P, \boldsymbol{w}^Q \in \Omega} \mathcal{R}^k(\boldsymbol{w}^P, \boldsymbol{w}^Q | U^k). \tag{7.13}$$

In min-max regret learning, we choose the pair of paths (P, Q) that is the maximizer of equation (7.13).

Algorithm 6: Maximum Regret Learning with presampled paths.

Input: $(\mathcal{X}, \mathcal{A}), x_0, K$
Output: $P^{\texttt{best}}$

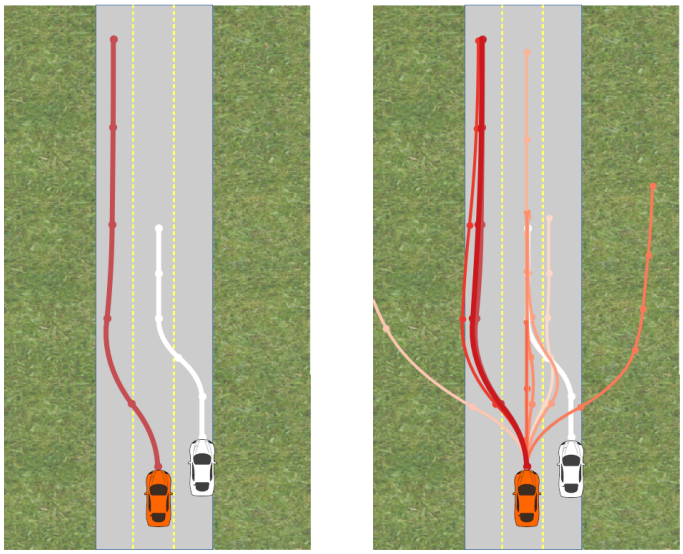
1 Initialize $U^0 = \emptyset$
2 Sample a set of weights Ω
3 **for** $k = 1$ *to* K **do**
4 $\quad \boldsymbol{w}^P, \boldsymbol{w}^Q \leftarrow \texttt{max\_regret}((\mathcal{X}, \mathcal{A}), x_0, U^{k-1}, \Omega)$
5 $\quad P \leftarrow \texttt{shortest\_path}(\boldsymbol{w}^P)$
6 $\quad Q \leftarrow \texttt{shortest\_path}(\boldsymbol{w}^Q)$
7 $\quad I^k \leftarrow \texttt{user\_feedback}(P, Q)$
8 $\quad$ **if** $I^k = -1$ **then**
9 $\quad\quad U^k = U^{k-1} \cup (P, Q)$
10 $\quad$ **else**
11 $\quad\quad U^k = U^{k-1} \cup (Q, P)$
12 $\boldsymbol{w}^{\texttt{best}} \leftarrow \mathbb{E}_{\boldsymbol{w}}[\boldsymbol{w}|U^k]$
13 **return** $P^{\texttt{best}}$

7.3.3 Preference Learning with Probabilistic Maximum Regret

Our proposed solution for active preference learning using *probabilistic maximum regret* is summarized in Algorithm 6. In each iteration we find the pair $(\boldsymbol{w}^p, \boldsymbol{w}^Q)$ that maximizes the probabilistic symmetric regret as in equation (7.13) over a set of samples Ω (line 4). We then obtain user feedback $I_k = 1$ if the user prefers path P and $I_k = -1$ otherwise (line 7) and add the feedback to a sequence (line 8-11). After K iterations, we choose $\boldsymbol{w}^{\texttt{best}}$ as the expected weight and return the corresponding shortest path (line 11,12). Using the maximum regret in the query selection is a greedy approach to minimize the maximum error. Given the current belief over the weights, we choose the pair P, Q with the maximum error ratio, discounted by the likelihoods of $\boldsymbol{w}^P$ and $\boldsymbol{w}^Q$.

7.4 Evaluation

We first evaluate the proposed approach using the simulation environment from [20], allowing us to compare our approach to theirs in the same experimental setup. We use the label `Entropy` to denote the maximum entropy learning from [20] and `Regret` for our maximum regret learning.



(a) Optimal behaviour. (b) Learned behaviour.

Figure 7.2: 'Driver' simulation scenario with different robot trajectories. Behaviour of an autonomous car (red) in the presence of another vehicle (white). In (a) we show the optimal behaviour for some user. In (b) we show alternative paths presented during active preference learning. Darker shades of red indicate behaviour that was presented later. The figure was created using code from [20].

Learning experiments First, we consider one of the experiments in [20]: The autonomous driving scenario (*Driver*) where an autonomous car moves on a three lane road in the presence of a human-driven vehicle as shown in Figure 7.2. Paths are described by four features: Heading relative to the road, staying in the lane, vehicle speed, and the distance to the other car. Every feature is averaged over the entire path. Furthermore, we introduce the *Extended Driver* experiment with additional features to create a more complex scenario. In addition to the above features we add the distance travelled along the road, the summed lateral movement, summed and maximum lateral and angular acceleration, the minimal speed, and the minimum distance to the other vehicle. We choose the Driver example because the entropy approach from [20] showed strong results and this scenario was already previously investigated in [87]. The extension aims to show how the learning techniques behave in higher dimensions.

Next we consider the specification revision problem, as described in Problem 3 and in Chapters 4 to 6. Here the feature vector $\boldsymbol{\phi}$ is of length $n+1$, where the first n entries are the features describing the length of path P in the i-th constraint for all $i = 1, 2, \ldots, n$.

The $n+1$-th feature is the time it takes the robot to execute path P. We will refer to this experiment as *Mobile*. The instance of the problem used for evaluation consists of 18 areas; thus, the dimensionality of the feature and weight space is 19.

Optimal paths Given a weight $\boldsymbol{w}$ we need to find the corresponding optimal path P in order to evaluate the path error (and to compute regret in Algorithm 6). In [20] no motion planner is given; in the experiments we rely on the generic non-linear optimizer `L-BFGS` [6] in the *Driver* and *Extended Driver* experiments. However, depending on the problem, this solver can return suboptimal solutions. To mitigate this effect, we presample paths, which are then used as a look-up-table, i.e., given a path that was found using `L-BFGS`, we iterate over all presampled paths; if a sampled path yields a better cost for the given weight, we use that path instead. In both experiments, the entropy approach uses the implementation provided by [20] where queries are chosen from $500,000$ presampled pairs of random paths. Since regret requires optimal paths, the regret approach uses a set of 200 weights with their corresponding optimal paths, yielding $40,000$ possible pairs of paths[1]. Finally, in these experiments the behaviour is actually captured by a reward and not a cost. Thus, an optimal path is found by minimizing the negative cost and we change the definition of regret to $r(\boldsymbol{w}^P, \boldsymbol{w}^Q) = 1 - {}^{c(P,\boldsymbol{w}^Q)}/_{c^*(\boldsymbol{w}^Q)}$.

In the *Mobile* experiment the robot moves using a state lattice planner [83]; given a weight $\boldsymbol{w}$ we can always find an optimal path in polynomial time. We varied the problem set up by choosing three different start and goal locations for the robot to navigate between. For each start goal pair we need to presample paths individually. The discrete state space led to a significantly smaller set of presamples, varying between 5 and 32. However, using randomly generated paths as in [20] for `Entropy` led to very poor performance. Therefore, we slightly modified the `Entropy` approach for this experiment, such that the same presamples were used as for the `Regret` approach.

Simulated users We simulate user feedback using the probabilistic user model from [20]. Given two paths, the user's uncertainty depends on how similar the paths are with respect to the cost function evaluated for $\boldsymbol{w}^{\texttt{user}}$:

$$\mathbb{P}\left(I = 1 | (P, Q), \boldsymbol{w}^{\texttt{user}}\right) = \frac{e^{c(P,\boldsymbol{w}^{\texttt{user}})}}{e^{c(P,\boldsymbol{w}^{\texttt{user}})} + e^{c(Q,\boldsymbol{w}^{\texttt{user}})}}. \tag{7.14}$$

[1] Using sampled optimal paths for the entropy approach did not lead to different results in the experiments, therefore we show the results using the original implementation.

The probabilistic regret is computed using pre-sampled weights as described in Algorithm 6, with an uncertainty of $p = 0.85$ in equation (7.6). Similar to [20] for each experiment we sample a user preference $\boldsymbol{w}^{\texttt{user}}$ uniformly randomly from the unit circle, i.e., $||\boldsymbol{w}^{\texttt{user}}||_2 = 1$. We notice that this can include *irrational* user behaviour: A negative weight on heading for instance would encourage the autonomous car to *not* follow the road.

7.4.1 Learning Error

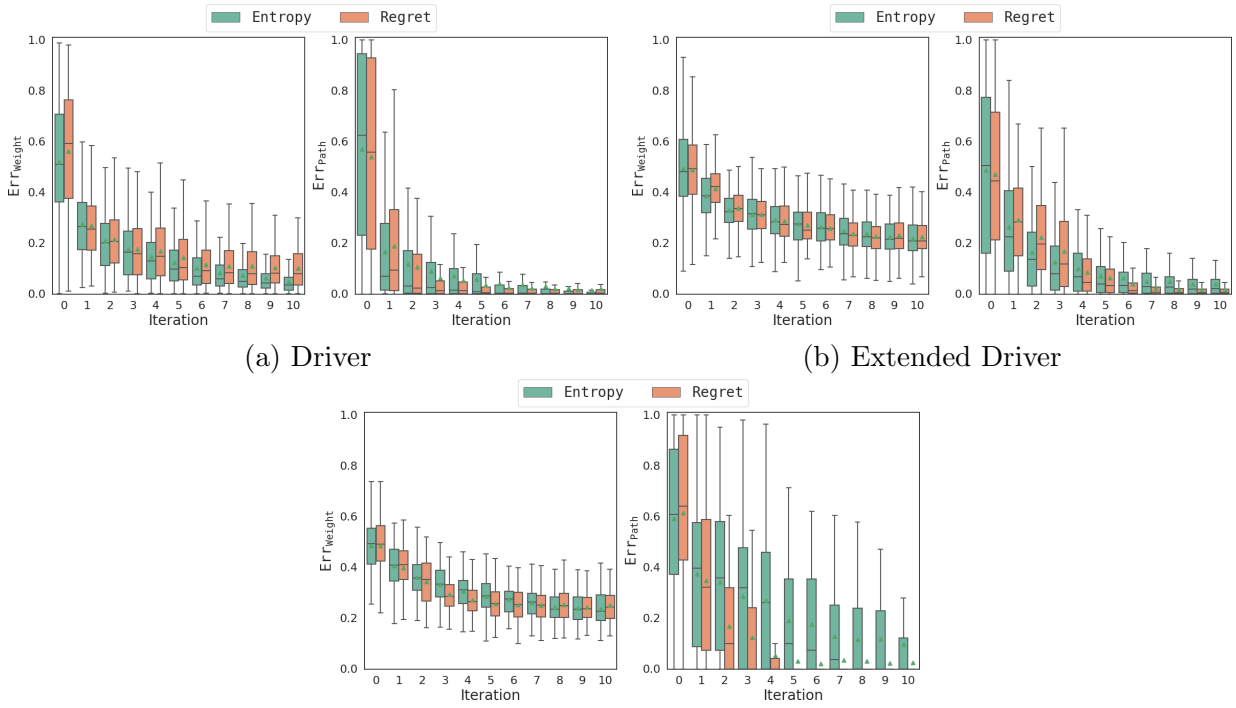
In Figure 7.3 we compare `Entropy` to `Regret` on both metrics over 10 iterations for the two experiments, each repeated 200 times. In the boxplots the center line shows the median and the green triangle shows the mean.

In the driver example, `Entropy` overall achieves a smaller weight error and smaller deviations from the mean, reproducing the results from [20]. In the path space we observe that `Entropy` achieves a slightly better result in the last two iterations. However, between iteration 2 and 8, the `Regret` approach performs better, i.e., learns more quickly. Overall, both approaches perform equally well with `Regret` showing a small benefit in convergence speed.

For the *Extended Driver* example in Figure 7.3b, both approaches make limited progress on the weight metric and exhibit large deviations. For the path error we observe that `Entropy` performs better initially, but makes little progress after iteration 6. The final median lies at ≈ 0.05 at iteration 10, but the highest quartile still reaches up to 0.2. The `Regret` approach achieves a lower mean and median error in iteration 4 and subsequently improves further. At iteration 8 the mean and median are close to 0, the box plot also shows that three quarters of all trials are very close to convergence.

Figure 7.3c illustrates the result for the *Mobile* experiment. Here, the weight error shows no difference between the two approaches; both perform equally poorly and inconsistently. At the same time the path error shows a large difference. `Regret` achieves convergence for nearly all trials after just 5 iterations (some outliers cause the mean value to still be at ≈ 0.5). At the same time the performance of `Entropy` is inferior: Even though the median error becomes 0 in iteration 8, the mean value is still ≈ 0.15 with large deviations.

In conclusion, `Regret` achieves an equally good result as `Entropy` on the path error for the *Driver* experiment, despite having a larger weight error. That is, while the weights found by `Entropy` are more similar to $\boldsymbol{w}^{\texttt{user}}$ based on the alignment metric, the resulting behaviour of `Regret` and `Entropy` are equally good. Moreover, the *Extended Driver* and *Mobile* experiments have shown that the performance of `Entropy` deteriorates for higher dimensions, i.e., larger sets of features. In contrast, `Regret` still achieves a very strong



(a) Driver

(b) Extended Driver

(c) Mobile robot

Figure 7.3: Comparison weight and path error in active preference learning with minimizing entropy or minimizing regret.

performance on the path error. In Section 7.4.3 we compare the two metrics in terms of robustness under different scenarios.

7.4.2 Easiness of Queries

A major contribution of [20] is the design of queries that are *easy* for the user to answer, i.e., the probability that the user choice is inconsistent with the assumed cost function from equation (7.14) is low. In maximum regret learning we do not directly consider the user's uncertainty when choosing a new pair of paths. However, as the paths maximizing the probabilistic symmetric regret have a large difference on cost, our approach implicitly selects paths P, Q that are easy for a user to answer, where $\boldsymbol{w}^{\texttt{user}} = \boldsymbol{w}^P$ or $\boldsymbol{w}^{\texttt{user}} = \boldsymbol{w}^Q$. To compare the easiness of the queries presented to the user, we consider the probability that the user would choose the path with lower cost, evaluated by $\boldsymbol{w}^{\texttt{user}}$ (7.14). In Figure 7.4 we compare the probability of correct user answers for `Entropy` and `Regret`.

In the *Driver* experiment, we recorded 84% correct answers for `Entropy`, which is slightly worse than reported for the strict queries in [20], where correct answers occurred in 87% of cases (3.5 wrong answers over 30 iterations on average). Nonetheless, with `Regret` the simulated answers were correct in 94% of cases, outperforming `Entropy`. In Figure 7.4, we observe that both approaches achieve very high probabilities for correct user answers in the first iteration, i.e., ask an *easy* question. Afterwards, the probabilities get smaller: The median of `Entropy` decreases to 0.9 in iteration 3 and the deviations increase significantly. The `Regret` approach maintains higher median values for all iterations. Interestingly, we observe cyclic decreases of the mean (and increases for the deviations) in iterations 4, 8 and 10. According to the user model the presented paths were very similar, indicating that the learning already might have gotten close to convergence. This aligns with the small errors of the expected weight reported in Figure 7.3a.

In the Extended Driver experiment the user behaviour is much more accurate for both approaches, indicating that the sampled paths differ more in cost. The user accuracy was 92% for `Entropy` and 96% for `Regret`. From iteration 7 onwards, `Entropy` starts to show larger deviations, i.e., questions become more difficult to answer, implying that the presented paths are very similar. Together with the very small decrease in path error the we observed in Experiment 1 (Figure 7.3b) at the same iteration, this leads to the conjecture that `Entropy` is converging to a local optimum.

Finally, the *Mobile* experiment did not show any difference between the two approaches, both achieving a very high accuracy of 99%. Overall, these results strongly support our claim that maximizing regret implicitly creates queries that are easy for the user to answer.

7.4.3 Generalization of the Error

Finally, we investigate how the two error metrics generalize to different scenarios, independent of whether the error is a result of learning with `Entropy` or `Regret`. That is, we investigate how useful each error metric is for predicting the learning performance when the robot needs to generalise to a new instance of the problem not encountered during active preference learning. Therefore, we consider the driver scenario from Figure 7.2 as a training case and construct 5 test cases by changing the initial state of the human driven vehicle (white). The weight error is scenario independent, it directly describes how similar the estimated weight is to $\boldsymbol{w}^{\texttt{user}}$. Thus, the weight error is the same in training and test cases and cannot be used as test error, as this would contain no additional information about performance on the test case. Hence, we use the path error as the test error. Further,

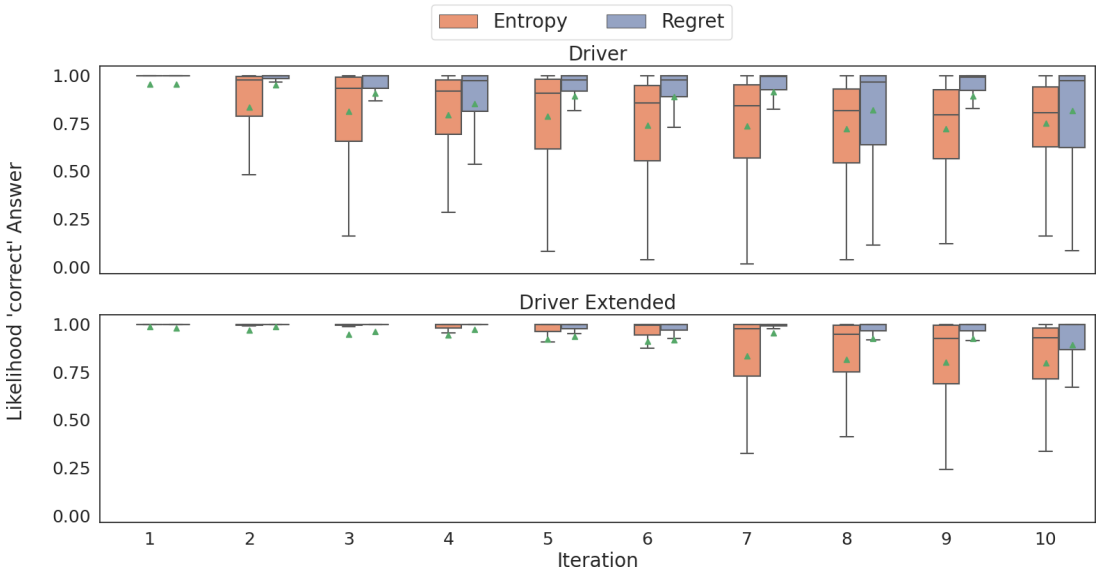


Figure 7.4: The likelihood that the simulated user gives the 'correct' answer, i.e., the probability in equation (7.14).

we notice that if the weight error is zero, i.e., the weights have been learned perfectly, then the path error is zero in all scenarios. However, as shown in Figure 7.3 and in [87, 20] the weights typically do not converge to the true user weight within a few iterations. Given some weight error, the path errors will differ for every test scenario. We are now interested in how well the weight error and the path error of the training scenario predict the path error of the test scenario.

We generate 40 different random user weights $\boldsymbol{w}^{\texttt{user}}$ and then generate 200 estimates of each of these weights. For every estimate we find the optimal path and compute the path and weight error which are used as training errors for the estimate. In Figure 7.5 we show how these training errors relate to the test error. We compare the path and weight error as a measure of generalisation performance (i.e., how well the weight and path errors predict the test case performance).

We observe that the path error translates linearly between training and test scenarios: Given a weight with a certain path error in the training scenario, the weight yields paths in the test scenarios that have a similar path error, on average. The relationship between weight error and test error is more complex. For a weight error of $0 - 0.01$ during training, we observe a test error of 0, i.e., if the weights are very close to the optimum, the optimal solution is found in every scenario. However, the test error shows large deviations, implying

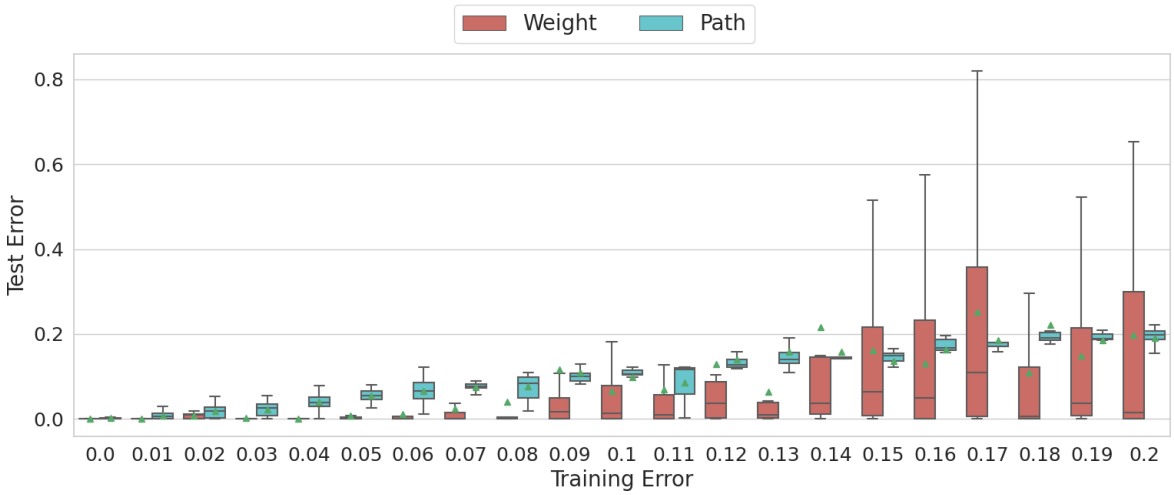


Figure 7.5: Relationship between training errors measured by the path and weight metric to test errors in the path metric.

that a low weight error in training is not a robust measure of how good the resulting behaviour is in test cases. The observation is supported by a strong Pearson correlation of $p = 0.92$ between training and test error for the path error, but a much weaker correlation of $p = 0.28$ for the weight error. This lends support to the claim that the path error is better suited for making predictions of the performance in scenarios that were not part of the training.

We conducted a the same experiment for the *Extended Driver* scenario. The correlation of the path error is weaker, but with $p = 0.80$ still substantially stronger than for the weight error where we observed $p = 0.28$. To conduct the same experiment for the *Mobile* scenario, we defined multiple start goal pairs in the environment, with one pair serving as the training task and the others as test tasks. However, we observed no correlation for both path error and weight error. The features in this scenario are local, i.e., describe if the robot visits a certain part of the environment. Learning about one task does only gain information about some features, and thus does not always provide sufficient information to find a good path for a different task.

In summary, we observe that the path error is more suitable than the weight error for reliable predictions of the test performance in scenarios with global features. However, higher dimensions can weaken the reliability, and local features may not allow for any predictions.

7.5 Discussion

Summary We investigated a new technique for generating queries in active preference learning for robot tasks. We have shown that competing state of the art techniques have shortcomings as they focus on the weight space only. As an alternative, we introduced the regret of the cost of paths as a heuristic for the query selection, which allows to greedily minimize the maximum error. Further, we studied the differences of our error function based on the cost of paths and an alternative error function based on the difference on weights from the literature. In simulations we demonstrated that using regret in the query selection leads to faster convergence than entropy while the queries are even easier for the user to answer. Moreover, we have shown that the path error allows for better predictions for other scenarios.

Computation time considerations Both query generation methods used in the evaluation, `Entropy` and `Regret`, require a discrete approximation of the set of all possible paths, which is obtained via pre-sampling. This is a major hindrance for practical application, since pre-sampling can take significant time. In Section 10.3 we further discuss this issue and outline a regret based learning method that does not rely on computing sample paths prior to learning and thus allows for practical use in new scenarios.

Nonetheless, even with pre-sampling we already observed a substantial computational advantage of the regret approach: An `Entropy` query takes $\approx 6s$ on average while a `Regret` query runs in $\approx 0.3s$ on average [2]. Calculating the expected change in entropy for a query candidate (P, Q) requires a tentative update of the posterior probability density functions (pdf) for the case that the user would like P over Q and vice versa. For each of these two tentative pdfs the information entropy is then approximated via sampling. For a large set of sampled queries (P, Q) this can result in a significant computational effort. In contrast, `Regret` is much simpler: For a query candidate (P, Q), we only require the regrets $r(\boldsymbol{w}^P, \boldsymbol{w}^Q)$ and $r(\boldsymbol{w}^Q, \boldsymbol{w}^P)$, which are just dot products in the dimension of the number of features, multiplied with the posterior of $\boldsymbol{w}^P$ and $\boldsymbol{w}^Q$. That is, `Regret` does not require a look-ahead for the possible new posterior distributions over $\boldsymbol{w}$ for the two possible outcomes of user feedback, resulting in the observed computation time benefit.

Theoretical guarantees Unlike the `Entropy` approach, our `Regret` approach does not come with a theoretical guarantee for the learning efficiency, i.e., given some iterations K,

[2] Computation times for single threaded execution on a i7-4710 2.5GHz.

how well is the measure for learning $f(\boldsymbol{w}^P, \boldsymbol{w}^Q, U^k)$ optimized compared to an optimal (but computationally intractable) algorithm. In [20] a sub-optimality bound for `Entropy` learning was established by showing that information entropy is an adaptive sub-modular set function [34]. However, information entropy is a surrogate measure for learning, i.e., the weight error in equation (7.3); thus, the sub-optimality bound for minimizing entropy gives only indirect information about how well the weight error is minimized.

In the proposed regret learning approach, we greedily minimize the maximum regret. Therefore, a sub-optimality bound would relate directly to the path error. However, minimizing the maximum discounted symmetric regret (7.12) is, unfortunately, not an adaptive submodular function.

Alternatively, the objective could be to minimize the *expected* discounted symmetric regret. This would be adaptive submodular and thus a greedy algorithm would be the best possible approximation algorithm for minimizing the expected regret over K iterations. Nonetheless, using the expected regret comes with two major drawbacks: (1) The maximum regret, i.e., path error, is an attractive measure since the user feedback to such a query will eliminate one of the worst possible paths from the set of candidates (or in a probabilistic sense decrease the belief over it). This property gets lost when using expectation: For example the expected regret can be relatively low if the current belief space contains some very good solutions, but also at least one that is highly suboptimal. (2) The question is whether the expectation is taken over weights or paths: If taken over weights, the sensitivity of the optimization problem can result in misleading results, since one path can be optimal for a large set of weights while another path is optimal for only a small set of weights. Thereby, some path that leads to high regret and still has a high posterior can be further disguised. On the other hand, taking the expectation over paths requires pre-sampling these paths. As discussed above, a major improvement to the presented work would be a query generation without pre-sampling, which conflicts with taking the expectation over paths.

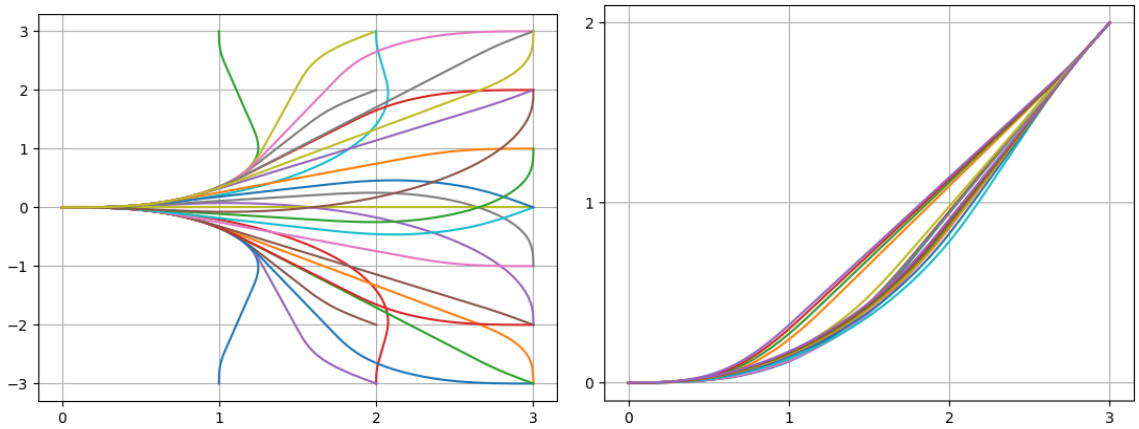
Chapter 8

User Preferences on State lattices

The research in this chapter was conducted in conjunction with another PhD. student of the University of Waterloo, Alexander Botros, and was published in the proceedings of The 14th International Workshop on the Algorithmic Foundations of Robotics (WAFR) 2020 [16]. A. Botros and N. Wilde contributed equally to all parts of the presented work.

In the previous chapters, we studied the problem of learning user preferences *given* a motion planner, i.e., the robots state and action space is fixed. Especially in the specification revision problem this is well justified: The features (expect path time) were based on user constraints on the environment and thus applied locally. However, trajectories features can also be global, i.e., apply on all parts of the environments. Such features can include passenger comfort in autonomous driving or avoiding high accelerations in autonomous material transport to reduce risks.

In this chapter, we investigate the design of a motion planner that represents user preferences. Broadly speaking, instead of just learning a cost function that represents user preferences, we also find a subset of the robot's action space that is optimized for this cost function. Only considering these optimized robot actions gives a motion planner a substantial performance benefit. We focus on lattice planners, where the problem becomes one of learning a control set. Lattice planners are widely used in autonomous driving [82, 83, 63, 69, 99]. The key idea is to pre-compute feasible trajectories between a discrete set of robot states. A pair of states together with a trajectory between the states is called a *motion primitive*; a collection of motion primitives forms a *control set*. Thus, the robot's dynamics do not need to be accounted for during the actual path planning. In Figure 8.1 we illustrate an example of a state lattice using clothoid trajectories [33], as well as a set of different trajectories between two fixed states obtained by varying the maximum



(a) Clothoid trajectory control set. (b) Sample clothoid trajectories.

Figure 8.1: Examples of a lattice control set and different trajectories.

curvature and maximum derivative of curvature. There are two aspects to consider when designing a control set: 1) the set of states the motion primitives should connect (i.e., the endpoints of each primitive), and 2) the actual trajectory used to make the connections. We address the second aspect by estimating the trajectories that a user would take to each goal configuration in a discretization of the configuration space. Generally, the time to compute a motion plan and the quality of the resulting plan increase with the number of primitives [82, 15]. Thus we address the first aspect by selecting a *subset* of the estimated trajectories with a given size. To estimate the user trajectories, we consider that the quality of a plan is evaluated by a user who has preferences with respect to the robot's motion. In autonomous driving, some users may prefer the vehicle to drive more aggressively, making sharp turns and maintaining high speeds, while others may prefer slower speeds and smoother turns [37, 39]. To find trajectories that reflect user preferences, we assume that users evaluate a robot's behaviour based on a weighted sum of features, similar to [87, 37].

The preferred behaviour of an autonomous robot often depends on situational context, like type of road or obstacles. Thus, a lattice planner based on trajectories that reflect the user preferences might only be applicable in certain scenarios. Complete planners that encompass many scenarios often use specialized lattices [1, 37, 29]. Our methodology is not limited to a specific scenario like driving on a highway or navigating in close proximity to an obstacle: for any of these scenarios we can learn a respective control set that captures the user preferences. Thus, we only consider global features such as trajectory length or maximum jerk, which are common in autonomous driving [37]. We learn the relative

importance of features from demonstrations from a single user, i.e., estimate the weights and the user's cost function. The cost function is then used as an input to a trajectory planner, e.g., for a ground vehicle a Dubins path, clothoid [33] or polynomial spiral [70], that computes the optimal trajectories between lattice states. Thus, our approach is agnostic to the employed local planner.

A major restriction for the performance of lattice planners in high-dimensional states is the branching factor (the number of connections in the control set). Higher dimensional states are common in autonomous driving applications where the set of states can extend to (x, y) position, orientation, curvature, and speed [69]. To improve runtime of the motion planner, we set an upper bound on the branching factor. Hence, there might not be a precomputed trajectory to every state in the lattice; such states are reached by executing a sequence of trajectories. A lattice should then be constructed such that over all states for which no precomputed direct trajectory is in the control set, the sub-optimality of the best trajectory sequence is minimized. This problem is closely related to the minimal t-spanning control set problem (MTSCSP) [82, 15]. Here, we find an estimate of the user cost function and compute a lattice with a given maximum branching factor best suited for the user. Restricting the size of the control set vastly improves planning time but at the expense of path quality. We introduce a novel methodology for designing a state lattice planner that considers user preferences. We pose the problem of simultaneously learning trajectories and a connection set of fixed size given demonstrations from a single user. The major contribution of this work is showing that an optimal solution to this problem can be found by applying a separation principle: First, we find the best estimate of the user preferences given data, which defines trajectories for all states in the lattice. Then we compute a connection set of given size that minimizes the error compared to using all connections in the lattice. We show that the proposed approach minimizes the maximum relative error of the expected optimal trajectories. Finally, we demonstrate the performance in simulations where the average cost of the paths computed with the learned control set is 1.4 of the optimum. Moreover, we obtain a substantial planning time speedup of more than factor 10 when using the learned connection set compared to planning with the entire lattice.

8.1 Problem Statement

In the preliminaries, we discussed the problem of computing a minimal t-spanning control set of a lattice $G_T^{\mathcal{B}} = (V, B, c)$. However, the formulation of this problem requires that the set of trajectories T, and thus the costs c, are known. Different users might have individual preferences for trajectories, which can be described by a respective cost function

c. We model users who evaluate a trajectory T_j from s to j as a weighted sum of features $\boldsymbol{\phi}(T_j) = \begin{bmatrix}\phi_1(T_j) & \phi_2(T_j) & \dots & \phi_n(T_j)\end{bmatrix}$:

$$c(T_j, \boldsymbol{w}) = \boldsymbol{\phi}(T_j) \cdot \boldsymbol{w} \tag{8.1}$$

where $\boldsymbol{w}$ is a vector of weights $\begin{bmatrix}w_1 & w_2 & \dots & w_n\end{bmatrix}$. The features can represent properties such as travel time, lateral acceleration, maximum jerk, etc. Similar user cost functions have been used in [71, 2, 87, 77] Without loss of generality, we assume that $\boldsymbol{w} \in [0,1]^n$. Given a user weight $\boldsymbol{w}$, we denote the cost of the optimal trajectory from s to j with respect to the weights $\boldsymbol{w}$ as

$$c_j^*(\boldsymbol{w}) = \min_{T_j} \ \phi(T_j) \cdot \boldsymbol{w}. \tag{8.2}$$

For a given set of weights $\boldsymbol{w} \in [0,1]^n$, and a given set of connections $\mathcal{E} \subseteq \mathcal{B}$, let $G^{\mathcal{E},\boldsymbol{w}} = (V, E, c^*)$. Here, E is the set of all pairs (i,j) such that there exists a connection $b = (s,k) \in \mathcal{E}$ taking i to j, and $c^*(i,j) = c_k^*(\boldsymbol{w})$. For any weight $\boldsymbol{w} \in [0,1]^n$, let $P_j^{\mathcal{E}}$ be a path from s to j on graph $G^{\mathcal{E},\boldsymbol{w}}$. For every edge e in a path $P_j^{\mathcal{E}}$ there is an associated connection $b \in \mathcal{E}$, and for every connection $b \in \mathcal{E}$, there is an associated trajectory $T_b \in T$. Thus a path $P_j^{\mathcal{E}}$ from s to j defines a trajectory from s to j which is the sequence of trajectories associated with each edge in the path. Let $(T_1, T_2, \dots)$ be the sequence of trajectories associated with the edges $(e_1, e_2, \dots)$ in a path $P_j^{\mathcal{E}}$. We extend the cost function c for trajectories to a cost function u of paths: For each feature ϕ_l there exists a feature function

$$f_l(P_j^{\mathcal{E}}) = f_l(\{\phi_l(T_1), \phi_l(T_2), \dots\}). \tag{8.3}$$

Here f_l depends of the trajectories associated with the edges in $e_1, e_2, \dots \in P_j^{\mathcal{E}}$. Further, we only consider functions $f_l(\cdot)$ that are monotone, i.e., $f_l(P^1) \leq f_l(P^2)$ if $P^1 \subseteq P^2$. For example, let ϕ_l be the length of a trajectory, then f_l describing the length of a path sums over all $\phi_l(T_1), \phi_l(T_2), \dots$. Furthermore, for other features we might require a non-linear mapping, e.g., the maximum curvature of a path is the maximum curvature among its trajectories. The user cost function of a path is then given by the weighted sum of all features

$$u(P_j^{\mathcal{E}}, \boldsymbol{w}) = w_1 f_1(P_j^{\mathcal{E}}) + \dots + w_n f_n(P_j^{\mathcal{E}}). \tag{8.4}$$

We notice that u is a generalization of the cost function c, i.e., c evaluates a single trajectory while u is a function of a set of trajectories which form a path. The user cost function u is similar to a reward function in reinforcement learning. However, it is challenging for users – especially non-experts – to specify weights for such a reward function [87]. Summarizing all weights in a row vector $\boldsymbol{w}$ and all features in a column vector $\boldsymbol{f}(P_j^{\mathcal{E}})$ allows us to write $u(P_j^{\mathcal{E}}, \boldsymbol{w}) = \boldsymbol{w}\boldsymbol{f}(P_j^{\mathcal{E}})$. Given weights $\boldsymbol{w}$, we define the optimal path as

$$P_j^{\mathcal{E},\boldsymbol{w}} = \arg\min_{P_j^{\mathcal{E}}} u(P_j^{\mathcal{E}}, \boldsymbol{w}). \tag{8.5}$$

Observe that this definition implies that

$$u(P_j^{\mathcal{B},\boldsymbol{w}}, \boldsymbol{w}) = c_j^*(\boldsymbol{w}). \tag{8.6}$$

Indeed, for connection set $\mathcal{B}$ there exists a direct connection taking s to j for all $j \in V$, and the cost of the minimal trajectory given $\boldsymbol{w}$ is given by $c_j^*(\boldsymbol{w})$.

Given a set of connections $\mathcal{E}$, and weights $\boldsymbol{w}$, let $T = \{T_b \ : \ b \in \mathcal{E}\}$ where T_b solves (8.2) for each $b \in \mathcal{E}$. That is, T is the set of minimal cost trajectories with respect to $\boldsymbol{w}$ for each connection in $\mathcal{E}$. We observe that a control set $(\mathcal{E}, T)$ can thus be determined by the tuple $(\mathcal{E}, \boldsymbol{w})$. For the remainder of this chapter, we focus on computing the tuple $(\mathcal{E}, \boldsymbol{w})$. Based on our cost function, the user-optimal behavior between all connections in $\mathcal{B}$ can be described by a control set $(\mathcal{B}, \boldsymbol{w}^*)$ where $\boldsymbol{w}^*$ denotes the *true user weights*. Users cannot provide $\boldsymbol{w}^*$ directly, and we learn about $\boldsymbol{w}^*$ from demonstrations. The true user weights may be dependent on the situation (e.g. highway vs city driving). This work focuses on a single situation but could be extended to account for multiple situations with the result being several control sets, one for each situation. For a given situation, we treat $\boldsymbol{w}^*$ as a hidden parameter which we have to estimate. Consider a path $P_j^{\mathcal{E},\boldsymbol{w}_1}$ that is optimal for a control set $(\mathcal{E}, \boldsymbol{w}_1)$, and a second weight $\boldsymbol{w}_2$. The cost of path $P_j^{\mathcal{E},\boldsymbol{w}_1}$ can be *evaluated* by $\boldsymbol{w}_2$, which we write as:

$$u_j(\mathcal{E}, \boldsymbol{w}_1 | \boldsymbol{w}_2) = \boldsymbol{w}_2 \cdot \boldsymbol{f}(P_j^{\mathcal{E},\boldsymbol{w}_1}) = u(P_j^{\mathcal{E},\boldsymbol{w}_1}, \boldsymbol{w}_2). \tag{8.7}$$

We read this as *the cost to j using control set $(\mathcal{E}, \boldsymbol{w}_1)$, evaluated by weights $\boldsymbol{w}_2$, i.e., the cost of the path that is optimal given weights $\boldsymbol{w}_1$, for a user whose weights are $\boldsymbol{w}_2$*. This error is similar to the relative error from Problem 2. Based on this notation, we can now pose the main problem statement:

Problem 6 (User control set)**.** Given a finite set of lattice states V with start state $s \in V$, hidden user preferences $\boldsymbol{w}^*$, and a budget on the number of allowable connections, $k \in \mathbb{Z}_{>0}$, find a control set $(\mathcal{E}, \boldsymbol{w})$, with $\mathcal{E} \subseteq \mathcal{B}, \boldsymbol{w} \in [0,1]^n$ such that

$$\begin{aligned}(\mathcal{E}, \boldsymbol{w}) = \arg\min_{\mathcal{E}', \boldsymbol{w}'} \ \max_{j \in V} \ & \frac{u_j(\mathcal{E}', \boldsymbol{w}' | \boldsymbol{w}^*)}{c_j^*(\boldsymbol{w}^*)} \\ & s.t. \ |\mathcal{E}'| \leq k, \ \boldsymbol{w}' \in [0,1]^n.\end{aligned} \tag{8.8}$$

The control set $(\mathcal{E}, \boldsymbol{w})$ minimizes the maximum ratio of path costs *evaluated* by $\boldsymbol{w}^*$ to the cost of the optimal direct trajectory to j, given a budget of k connections. In essence $(\mathcal{E}, \boldsymbol{w})$ is the control set with the minimal t-error and is therefore the most robust control set.

8.2 Approach

Problem 6 consists determining a control set $(\mathcal{E}, \boldsymbol{w})$ that minimizes the maximum ratio between path cost and optimal cost. We introduce a model for how users provide demonstrations and then analyse how the unknown parameter $\boldsymbol{w}^*$ in equation (8.8) can be substituted by an estimate of the user weights given data. We show that the optimal solution is a pair $(\mathcal{E}, \boldsymbol{w})$ where $\boldsymbol{w}$ is the best available estimate of $\boldsymbol{w}^*$ and $\mathcal{E}$ can be computed via a mixed-integer linear program. This allows for a simple, yet effective method for solving Problem 6.

8.2.1 User model

We consider a user with a preference $\boldsymbol{w}^*$. Let d be a demonstrated trajectory from a start state s to a goal state $j \in X$. We do not require j to be a lattice point as we can still evaluate features $\boldsymbol{f}(d)$ and thus assign a cost $u_j(d, \boldsymbol{w})$ as in (8.4). We denote $d^{\boldsymbol{w}^*}$ the minimal-cost demonstration from s to j given weights $\boldsymbol{w}^*$. Users cannot perfectly demonstrate $d^{\boldsymbol{w}^*}$. Thus we use the features of demonstrations to formulate a probabilistic user model:

Assumption 1 (User Model)**.** A user with a preference $\boldsymbol{w}^*$ provides a demonstration d from s to j with features $\boldsymbol{f}(d)$ where the density $p(\boldsymbol{f}(d)|\boldsymbol{w}^*)$ is:

$$p(\boldsymbol{f}(d)|\boldsymbol{w}^*) \sim \mathcal{N}(\boldsymbol{f}(d^{\boldsymbol{w}^*}), \boldsymbol{\sigma}). \tag{8.9}$$

Similar user models have been used in [78, 71]. Thus, users provide demonstrations such that the features follow a normal distribution centred at the features of the optimal demonstration. That is the user demonstrations are unbiased and, given a large enough data set, the average features of demonstrations equal the optimal features. Following (8.4), two demonstrations with equal features have the same cost for any user and thus are indistinguishable with respect to the cost function. Hence, we consider only features of demonstrations. Let $\mathcal{D} = (d_1, d_2, \dots)$ be a sequence of demonstrations. We then find the conditional expectation of $\boldsymbol{w}$ given $\mathcal{D}$ by taking the Bayesian posterior:

$$\begin{aligned} \mathbb{E}_{\boldsymbol{w}}[\boldsymbol{w}|\mathcal{D}] &= \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} \mathbb{E}[\boldsymbol{w}|\boldsymbol{f}(d)] p(\boldsymbol{f}(d)) \\ &= \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} \int_{\boldsymbol{w} \in [0,1]^n} \boldsymbol{w}\ p(\boldsymbol{f}(d)|\boldsymbol{w})\ p(\boldsymbol{w}) d\boldsymbol{w}. \end{aligned} \tag{8.10}$$

Finally, we can approximate the integral by summing over a set of N samples:

$$\mathbb{E}_{\boldsymbol{w}}[\boldsymbol{w}|\mathcal{D}] \approx \frac{1}{|\mathcal{D}|}\sum_{d\in\mathcal{D}}\frac{1}{N}\sum_{i=1}^{N}\boldsymbol{w}_i\ p(\boldsymbol{f}(d)|\boldsymbol{w}_i)\ p(\boldsymbol{w}_i). \tag{8.11}$$

8.2.2 Estimation of the Loss Function

We now use the user model to find a solution to Problem 6, given a set of user demonstrations $\mathcal{D}$. We consider two approaches. The first is taking the conditional expectation over equation (8.8), given $\mathcal{D}$:

$$\begin{aligned}(\bar{\mathcal{E}},\bar{\boldsymbol{w}}) = \arg\min_{\mathcal{E}',\boldsymbol{w}'} \mathbb{E}_{\boldsymbol{w}}\left[\max_{j\in V}\ \frac{u_j(\mathcal{E}',\boldsymbol{w}'|\boldsymbol{w})}{c_j^*(\boldsymbol{w})}\middle|\mathcal{D}\right]\\ s.t.\ |\mathcal{E}'|\le k,\ \boldsymbol{w}'\in[0,1]^n.\end{aligned} \tag{8.12}$$

The second approach is to use the expectation $\hat{\boldsymbol{w}} = \mathbb{E}[\boldsymbol{w}|\mathcal{D}]$ to compute $(\bar{\mathcal{E}},\bar{\boldsymbol{w}})$, also known as the plug-in estimator [105]:

$$\begin{aligned}(\bar{\mathcal{E}},\bar{\boldsymbol{w}}) = \arg\min_{\mathcal{E}',\boldsymbol{w}'}\ \max_{j\in V}\ \frac{u_j(\mathcal{E}',\boldsymbol{w}'|\hat{\boldsymbol{w}})}{c_j^*(\hat{\boldsymbol{w}})}\\ s.t.\ |\mathcal{E}'|\le k,\ \boldsymbol{w}'\in[0,1]^n.\end{aligned} \tag{8.13}$$

While it is an approximation, (8.13) approaches the desired Problem 6 as $|\mathcal{D}|\to\infty$. Thus, this work focuses on solving (8.13).

8.2.3 Main Results

In this section, we present the main theorem of this chapter that proposes a solution to the minimization problem in (8.13). The high-level idea is: given demonstrations $\mathcal{D}$, the expected user weight $\hat{\boldsymbol{w}}$ is computed. This user weight is used to calculate trajectories that minimize the user cost for all connections in $\mathcal{B}$. Finally, a set $\mathcal{E}\subseteq\mathcal{B}$ of size $\le k$ is found to produce a control set $(\mathcal{E},\hat{w})$. In order to illustrate how the connection set $\mathcal{E}$ is computed, we present a variant of the MTSCSP from [83, 15] that is used in the main results of this chapter.

Problem 7 (Minimum k-spanning Connection set Problem (MKSCSP))**.** Given a tuple $(V, \mathcal{B}, c)$, and an integer $k > 0$, compute a set $\mathcal{E}$ such that

$$\mathcal{E} = \arg\min_{\substack{\mathcal{E}' \subseteq \mathcal{B} \\ |\mathcal{E}| \leq k}} \tau(\mathcal{E}'), \tag{8.14}$$

where $\tau(\mathcal{E})$ is defined in (3.1) with $c(j) = c_j^{\mathcal{B}}$ for all $j \in V$. If $\mathcal{E}$ solves (8.14), and $\tau(\mathcal{E}) < \infty$ we say that $\mathcal{E}$ is a *minimal k-spanning connection set* (MKSCS).

We make an observation about the cost function u, that motivates our main results.

Observation 1 (Weight Choice)**.** For any set of connections $\mathcal{E}$, any vertex $j \in V$, and any pair of weights $\boldsymbol{w}, \boldsymbol{w}' \in [0,1]^n$, it must hold that

$$u_j(\mathcal{E}, \boldsymbol{w}|\boldsymbol{w}) \leq u_j(\mathcal{E}, \boldsymbol{w}'|\boldsymbol{w}). \tag{8.15}$$

Indeed, from the definition of paths $P_j^{\mathcal{B},\boldsymbol{w}}$ and user costs in (8.5), and the definition of user costs evaluated by other weights in (8.7), we observe that

$$u_j(\mathcal{E}, \boldsymbol{w}|\boldsymbol{w}) = u(P_j^{\mathcal{E},\boldsymbol{w}}, \boldsymbol{w}) \leq u(P_j^{\mathcal{E},\boldsymbol{w}'}, \boldsymbol{w}) = u_j(\mathcal{E}, \boldsymbol{w}'|\boldsymbol{w}). \tag{8.16}$$

Theorem 1 (Problem Solution)**.** If the tuple $(\bar{\mathcal{E}}, \bar{\boldsymbol{w}})$ is a solution to minimization problem (8.13), then $\bar{\boldsymbol{w}} = \hat{\boldsymbol{w}}$, and $\bar{\mathcal{E}}$ is a MKSCS of the lattice whose costs are computed using weights $\hat{\boldsymbol{w}}$ (i.e., the lattice $G^{\mathcal{B},\hat{\boldsymbol{w}}}$).

Proof. Let $\hat{\mathcal{E}}$ denote a MKSCS of the lattice $G^{\mathcal{B},\hat{\boldsymbol{w}}}$. Note that Observation 1 implies that the optimal value for $\bar{\boldsymbol{w}}$ in equation (8.13) is given by $\hat{\boldsymbol{w}}$ for any set of connections $\mathcal{E}'$. Indeed, for any weights $\boldsymbol{w}' \in [0,1]^n$, equation (8.15) implies that $u_j(\mathcal{E}', \boldsymbol{w}'|\hat{\boldsymbol{w}}) \geq u_j(\mathcal{E}', \hat{\boldsymbol{w}}|\hat{\boldsymbol{w}})$. Thus, for any $\mathcal{E}'$, the pair $(\mathcal{E}', \boldsymbol{w}')$ must result in paths whose cost is at least $(\mathcal{E}', \hat{\boldsymbol{w}})$. This allows us to simplify equation (8.13) to

$$\begin{aligned} \bar{\mathcal{E}} = \arg\min_{\mathcal{E}'} \ \max_{j \in V} \ & \frac{u_j(\mathcal{E}', \hat{\boldsymbol{w}}|\hat{\boldsymbol{w}})}{c_j^*(\hat{\boldsymbol{w}})}. \\ s.t. \ & |\mathcal{E}'| \leq k. \end{aligned} \tag{8.17}$$

We will now show that $\hat{\mathcal{E}}$ solves (8.17). Observe that the t-error for any set $\mathcal{E}'$ given the lattice $G^{\mathcal{B},\hat{\boldsymbol{w}}}$ is exactly $u_j(\mathcal{E}', \hat{\boldsymbol{w}}|\hat{\boldsymbol{w}}) / c_j^*(\hat{\boldsymbol{w}})$ which appears in the minimization in (8.17). Therefore, it must hold that if $\hat{t} = \tau(\hat{\mathcal{E}})$ given the lattice $G^{\mathcal{B},\hat{\boldsymbol{w}}}$, then for any other set of primitives $\bar{\mathcal{E}} \subseteq \mathcal{B}$ such that $|\bar{\mathcal{E}}| \leq k$ we have

$$\tau(\hat{\mathcal{E}}) \leq \tau(\bar{\mathcal{E}}). \tag{8.18}$$

This follows from the assumption that $\hat{\mathcal{E}}$ is a MKSCS for the lattice $G^{\mathcal{B},\hat{\boldsymbol{w}}}$. Thus $\bar{\mathcal{E}}$ solves the minimization problem (8.17). Finally, let $\bar{\mathcal{E}}$ be any solution to (8.17). Then, it must hold that $\tau(\bar{\mathcal{E}}) = \hat{t}$, implying $\bar{\mathcal{E}}$ is also a MKSCS of the lattice $G^{\mathcal{B},\hat{\boldsymbol{w}}}$. Therefore, we have that $\boldsymbol{w}'$ is a weight that solves (8.13) if and only if $\boldsymbol{w}' = \hat{\boldsymbol{w}}$. Given that $\boldsymbol{w}' = \hat{\boldsymbol{w}}$, we may reduce (8.13) to (8.17) which is solved by a connection set $\bar{\mathcal{E}}$, if and only if $\bar{\mathcal{E}}$ is a MKSCS of the lattice $G^{\mathcal{B},\hat{\boldsymbol{w}}}$. □

Corollary 1.1. If the control set $(\mathcal{E}, \boldsymbol{w})$ is a solution to equation (8.8), then $\boldsymbol{w} = \boldsymbol{w}^*$, and $\mathcal{E}$ is a MKSCS on the lattice $G^{\mathcal{B},\boldsymbol{w}^*}$.

The proof of Corollary (1.1) follows closely the proof of Theorem 1, and is therefore omitted. Letting $\hat{\mathcal{E}}$ denote a MKSCS of the lattice $G^{\mathcal{B},\hat{\boldsymbol{w}}}$, Theorem 1 shows that $(\hat{\mathcal{E}}, \hat{\boldsymbol{w}})$ solves (8.13). The next theorem extends this result to a lattice whose trajectory costs are given by their expected costs, given $\mathcal{D}$, if the features $f_i, i \in \{1, \dots, n\}$ are piece-wise continuous functions of $\boldsymbol{w}$. We say that a multivariate function $f_i(\boldsymbol{w})$ is piece-wise continuous on the compact set $[0,1]^n$ if for any two weight vectors $\boldsymbol{w}_1, \boldsymbol{w}_2 \in [0,1]^n$, the function f_i is piece-wise continuous on the line-segment connecting $\boldsymbol{w}_1, \boldsymbol{w}_2$.

Theorem 2 (Solution for the Expected Cost Lattice)**.** Let $(\hat{\mathcal{E}}, \hat{\boldsymbol{w}})$ be a solution to problem (8.13). If the features f_i are piece-wise continuous functions of $\boldsymbol{w}$ for all $i \in \{1, \dots, n\}$ then it holds that

$$(\hat{\mathcal{E}}, \hat{\boldsymbol{w}}) = \arg\min_{\mathcal{E}', \boldsymbol{w}'} \max_{j \in V} \frac{\mathbb{E}_{\boldsymbol{w}}[u_j(\mathcal{E}', \boldsymbol{w}'|\boldsymbol{w})|\mathcal{D}]}{\mathbb{E}_{\boldsymbol{w}}[c_j^*(\boldsymbol{w})|\mathcal{D}]}. \tag{8.19}$$

That is, the control set $(\hat{\mathcal{E}}, \hat{\boldsymbol{w}})$ minimizes the t-error given a lattice whose trajectory costs are defined by their expected costs given $\mathcal{D}$.

Proof. We first show that for any fixed $\mathcal{E}' \subseteq \mathcal{B}$ and $j \in V$, the function $u_j(\mathcal{E}', \boldsymbol{w}|\boldsymbol{w})$ is a piece-wise linear function of $\boldsymbol{w}$. If we fix the first argument as $\boldsymbol{w} = \boldsymbol{w}'$, the cost function $u_j(\mathcal{E}', \boldsymbol{w}'|\boldsymbol{w})$ becomes linear in $\boldsymbol{w}$. Indeed, from (8.4) and (8.7), we have $u_j(\mathcal{E}', \boldsymbol{w}'|\boldsymbol{w}) = \boldsymbol{w} \cdot \boldsymbol{f}(P_j^{\mathcal{E}', \boldsymbol{w}'})$, where $\boldsymbol{f}(P_j^{\mathcal{E}', \boldsymbol{w}'})$ is constant for a fixed control set $(\mathcal{E}', \boldsymbol{w}')$. Because the features $\boldsymbol{f}$ are assumed to be piece-wise continuous, we can conclude from (8.7) that $u(\mathcal{E}', \boldsymbol{w}|\boldsymbol{w})$ is also piece-wise continuous. Let $\epsilon > 0, \boldsymbol{w}_1, \boldsymbol{w}_2$ be such that $u(\mathcal{E}', \boldsymbol{w}|\boldsymbol{w})$ is continuous on a line-segment

$$L = \{\lambda_1 \boldsymbol{w}_1 + \lambda_2 \boldsymbol{w}_2 \ : \ \lambda_1 \in [-\epsilon, 1+\epsilon], \ \lambda_2 = (1-\lambda_1)\}. \tag{8.20}$$

That is, $u(\mathcal{E}', \boldsymbol{w}|\boldsymbol{w})$ is continuous on a line segment containing and extending a small distance depending on ϵ past $\boldsymbol{w}_1, \boldsymbol{w}_2$. Observe that the linearity of $u(\mathcal{E}', \boldsymbol{w}'|\boldsymbol{w})$ for any

fixed weights $\boldsymbol{w}'$, together with Observation 1 imply that

$$\begin{aligned} u_j(\mathcal{E}', &\lambda_1\boldsymbol{w}_1 + \lambda_2\boldsymbol{w}_2|\lambda_1\boldsymbol{w}_1 + \lambda_2\boldsymbol{w}_2) \\ &= \lambda_1 u_j(\mathcal{E}', \lambda_1\boldsymbol{w}_1 + \lambda_2\boldsymbol{w}_2|\boldsymbol{w}_1) + \lambda_2 u_j(\mathcal{E}', \lambda_1\boldsymbol{w}_1 + \lambda_2\boldsymbol{w}_2|\boldsymbol{w}_2) \\ &\geq \lambda_1 u_j(\mathcal{E}', \boldsymbol{w}_1|\boldsymbol{w}_1) + \lambda_2 u_j(\mathcal{E}', \boldsymbol{w}_2|\boldsymbol{w}_2). \end{aligned} \tag{8.21}$$

Thus, the function u_j lies above the line passing through $u_j(\mathcal{E}', \boldsymbol{w}_1|\boldsymbol{w}_1)$, and $u_j(\mathcal{E}', \boldsymbol{w}_2|\boldsymbol{w}_2)$. Since this holds for all $\boldsymbol{w}_1', \boldsymbol{w}_2' \in L$, and $u_j(\mathcal{E}', \boldsymbol{w}|\boldsymbol{w})$ is continuous on the line connecting $\boldsymbol{w}_1', \boldsymbol{w}_2'$, it must be linear there. Thus, if $\boldsymbol{f}(\boldsymbol{w})$ is continuous on any line segment, then $u_j(\mathcal{E}', \boldsymbol{w}|\boldsymbol{w})$ is linear there. Were the values of λ_1, λ_2 in (8.21) restricted to $[0, 1]$, it would only imply concavity, but (8.21) holds for all $\lambda_1, \lambda_2 \in \mathbb{R}_{\geq 0}$, implying piece-wise linearity. By the linearity of the cost function, $u_j(\mathcal{E}', \boldsymbol{w}'|\boldsymbol{w})$ in $\boldsymbol{w}$ for fixed $\boldsymbol{w}'$, we have $\mathbb{E}_{\boldsymbol{w}}[u_j(\mathcal{E}', \boldsymbol{w}'|\boldsymbol{w})|\mathcal{D}] = u_j(\mathcal{E}', \boldsymbol{w}'|\hat{\boldsymbol{w}})$. By the linearity of $u_j(\mathcal{E}', \boldsymbol{w}|\boldsymbol{w})$ in $\boldsymbol{w}$ for any $\mathcal{E}' \subseteq \mathcal{B}$, and (8.6) we have $\mathbb{E}_{\boldsymbol{w}}[c_j^*(w)|\mathcal{D}] = c_j^*(\hat{\boldsymbol{w}})$. Thus the minimization problem (8.19) reduces to (8.13), the solution to which is given by $(\hat{\mathcal{E}}, \boldsymbol{w})$ in Theorem 1. □

8.2.4 Computational Complexity

In this section we prove the NP-completeness of the decision version of the minimization (8.8). We begin by stating the decision version of Problem 7.

Problem 8 (Decision version of MKSCSP)**.** Given tuple $(V, \mathcal{B}, c)$, integer $k > 0$, and a real number $t \geq 1$, does there exist a set $\mathcal{E} \subseteq \mathcal{B}$ with $|\mathcal{E}| \leq k$ and $\tau(\mathcal{E}) \leq t$?

Theorem 3 (NP-completeness)**.** Problem 8 is NP-complete.

Proof. The MTSCSP in the preliminary section was shown to be NP-complete in [15]. Its decition version is: *Given a lattice $G^{\mathcal{B},\boldsymbol{w}}$, a real number $t \geq 1$ and an integer $k > 0$, does there exist a set $\mathcal{E} \subseteq \mathcal{B}$ where $\tau(\mathcal{E}) \leq t$ and $|\mathcal{E}| \leq k$?* We reduce the MTSCSP decision problem to Problem 8. Given a lattice $G^{\mathcal{B},\boldsymbol{w}}$, and $t \geq 1, k > 0$, we construct an instance of the MKSCSP as follows: let $V = \{j : (s, j) \in \mathcal{B}\}$, and let $c(j) = c_j^*(\boldsymbol{w})$ for all $j \in V$. Then we observe that Problem 8 on this instance is identical to the decision version of the MTSCSP. Thus, Problem 8 is NP-hard because the MTSCSP decision problem is. Observe that a potential solution $\mathcal{E} \subseteq \mathcal{B}$ to Problem 8 can be verified in time polynomial $|V|$ by iterating over all vertices $j \in V$ to check $\tau(\mathcal{E}) \leq t$, and additionally checking that $|\mathcal{E}| \leq k$. Hence, Problem 8 is in NP and thus NP-complete. □

8.2.5 Computing an Optimal Control Set

We now briefly summarize the proposed solution to problem (8.13). By Theorem 1, the solution is a tuple $(\hat{\mathcal{E}}, \hat{\boldsymbol{w}})$, where $\hat{\boldsymbol{w}}$ can be approximated using (8.11), and the MKSCS, $\hat{\mathcal{E}}$ is computed via a variant of the MTSCSP mixed-integer linear program from [15]. This is done by adding the constraint that $|\hat{\mathcal{E}}| \leq k$, and changing the objective of the problem from minimizing $|\hat{\mathcal{E}}|$ to minimizing $\tau(\hat{\mathcal{E}})$.

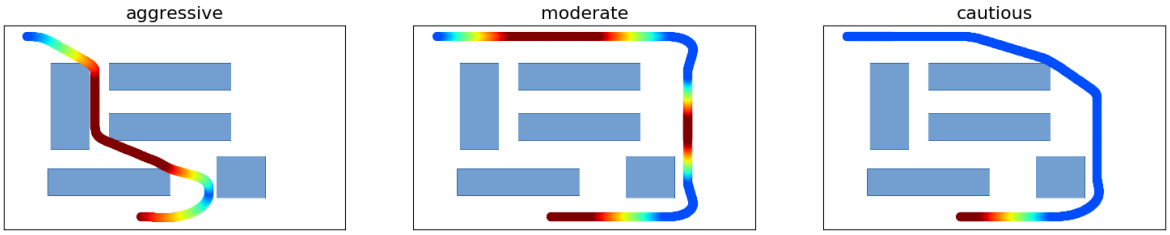
8.3 Evaluation

We now demonstrate the performance of the proposed approach in simulations. We consider a known, static environment, discretized into four-dimensional states. Each state consists of a (x, y) position, an orientation θ and the current speed v. We considered 3 discrete values of speed, and 8 discrete headings (cardinal and ordinal). All used *environments* consisted of 15×15 grid of (x, y) positions and hence of $15 \times 15 \times 8 \times 3 = 5400$ states (vertices). The *lattice* had pre-computed trajectories for x-values between 0 and 4, y-values between -3 and 3, all 8 headings and all 3 speed values. Trajectories were computed using clothoids, a subset of the used lattice is illustrated in Figure 8.1. Each vertex has to be reached from a start for every discrete speed value and from a cardinal and ordinal orientation, yielding 6 different starts $(0, 0, j, v_i), j \in \{0, \pi/4\}, i \in \{1, 2, 3\}$. Thus, the connection set $\mathcal{B}$ for each start contains up to 672 connections.

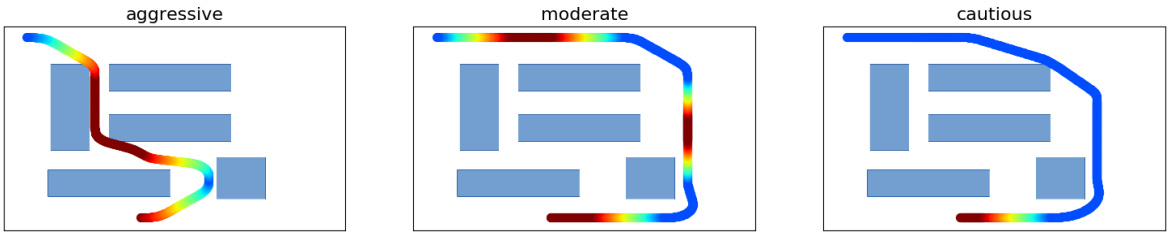
We model users who evaluate trajectories based on three features: travel time, longitudinal acceleration and lateral acceleration. User demonstration were simulated by sampling features from the distribution in (8.9). We consider three preferences corresponding to different driving styles: An aggressive style which prefers short travel times, a cautious style that favours low accelerations, and a moderate style that balances the features more equally. For all user types we set the covariance in (8.9) to 0.1. We illustrate an example demonstration of each user in Figure 8.2a. Finally, the experiment comprises one training environment, shown in Figure 8.2 and two test environments.

8.3.1 Training Error

For each of the three driving style, we obtain three demonstrations for different start-goal pairs. We show that we can effectively learn user preferences despite this small number of demonstrations. We estimate $\hat{\boldsymbol{w}}$ for each user by computing the expectation in equation



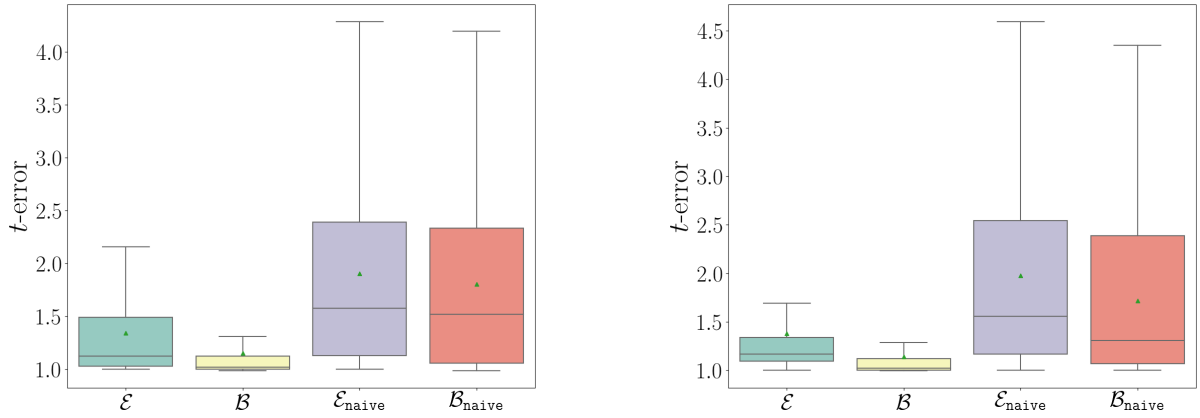
(a) Demonstrated behaviour.



(b) Learned behaviour.

Figure 8.2: Demonstrated and learned behaviour in the training environment. The color of the path indicates speed, where blue corresponds to slow and red corresponds to fast.

(8.11) using $N = 10$ samples and assuming a covariance of 0.05, i.e., we overestimate how accurately the user demonstration match their optimal trajectories. Finally, we run the experiment for minimal k-spanning connection sets of size 25, 50 and 100 over 40 trials each. Figure 8.2 shows one of the training demonstrations starting at the bottom at high speed and going to the top left corner. We compare the optimal paths for the three different users, together with the paths that were planned using the learned control set $(\hat{\mathcal{E}}, \hat{\boldsymbol{w}})$. While the shape of the paths is slightly different from the demonstrations, key characteristics of the user preferences are replicated. The aggressive user only breaks for sharp turns and immediately accelerates again. The moderate user avoids breaking too abruptly and thus cannot take the shortcut. Finally, the cautious user minimizes lateral and longitudinal acceleration and thus drives with minimal speed whenever possible. Figure 8.3a shows the t-error over all users and k values for the control set $(\hat{\mathcal{E}}, \hat{\boldsymbol{w}})$ together with the error of the control set using all connections $(\mathcal{B}, \hat{\boldsymbol{w}})$ in the lattice. We include a naive approach as a reference, where $\boldsymbol{w}_{\texttt{naive}}$ is sampled randomly and independent to the demonstrations. This serves as a baseline to illustrate the advantage of estimating driver behavior, i.e., the sensitivity of the cost function to user preferences $\boldsymbol{w}$. For the naive approach we show the error of a MKSCS $(\mathcal{E}_{\texttt{naive}}, \boldsymbol{w}_{\texttt{naive}})$ and of $(\mathcal{B}_{\texttt{naive}}, \boldsymbol{w}_{\texttt{naive}})$. We observe that $(\mathcal{B}, \hat{\boldsymbol{w}})$ achieves an average error of 1.16, with a median of 1.02. The MKSCS solution has a mean and



(a) Training Error. (b) Test error.

Figure 8.3: The t-error for all connections $\mathcal{B}$ and the MKSCS connections $\mathcal{E}$ compared to a naive approach with $\mathcal{B}_{\texttt{naive}}$ and $\mathcal{E}_{\texttt{naive}}$, respectively.

median error of 1.34 and 1.13, respectively. Thus, the limited size of the control set leads to a mean cost that is 34% higher than the optimum, but for half of all cases the cost is at most 13% higher. In comparison, we observe that the naive approach yields a mean error of 1.81 with a median of 1.52 when using all connections, and a slightly higher error for a MKSCS.

In Figure 8.4 we compare how different values for k influence the t-error and the planning time speedup when using a MKSCS. Generally, the t-error decreases as k increases, though we observe large differences between user types. For the cautious user the error does not exceed 1.1 on average, independent of k. While the aggressive user type achieves comparable values for $k = 100$, the error increases drastically for smaller k, reaching an average of ≈ 2.2 for $k = 25$ with a large deviation from the mean. The moderate user shows a more balanced result with the average not surpassing 1.5 for small k. Yet, the planning time speedup is less affected by the user type. For $k = 25$ we observe the highest speedup of ≈ 35 on the median for all three user types. For higher values of k the speedup decreases, but even using a connection set with $|\mathcal{E}| = 100$, the median path planning is at least 10 times faster than when using all connections $\mathcal{B}$. We conclude that using a MKSCS drastically decreases planning time though the cost increases 34% compared to the optimum (twice the error of the best estimate). However, in half of all training examples the increase in cost is less than 10%. Between users, the performance benefit is similar, while the path quality differs.

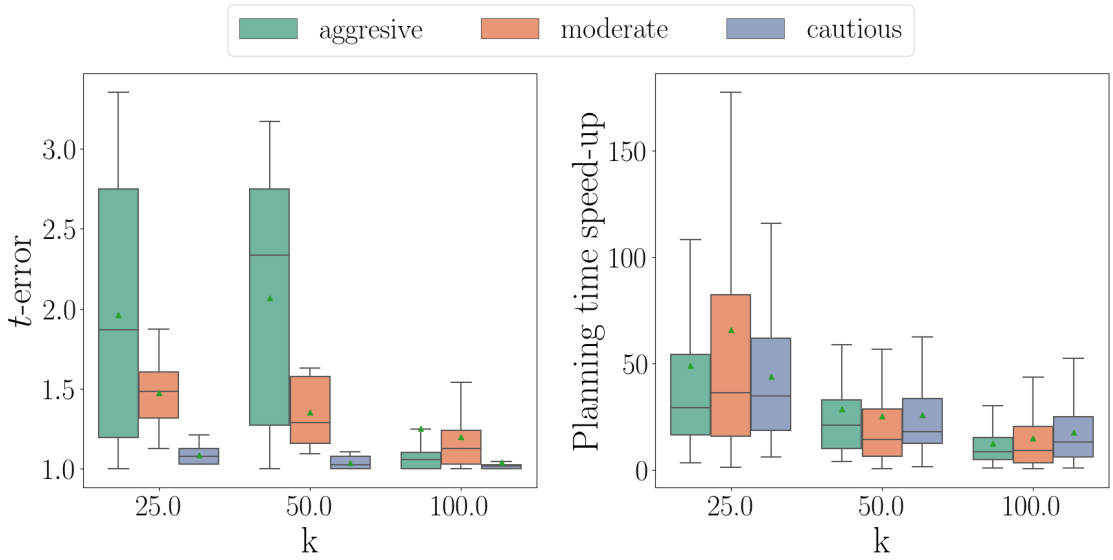


Figure 8.4: Training data: The t-error and planning time speedup for different sizes of the MKSCS and the different user types.

8.3.2 Test Error

The test error is evaluated on three different start–goal pairs for which no demonstrations were obtained in two different environments. In Figure 8.3b we show a similar analysis as we did for the training error. The error of the learned control set $(\hat{\mathcal{E}}, \hat{\boldsymbol{w}})$ is slightly higher than in the training with a mean of 1.38 and a median of 1.17 while the deviation is smaller. The naive approach performs slightly better than in the training, with mean and median values for $(\mathcal{B}_{\texttt{naive}}, \boldsymbol{w}_{\texttt{naive}})$ of 1.72 and 1.31, respectively. This indicates that the user behaviour might be less sensitive in the test scenarios. Figure 8.5 shows a comparable relationship between k and the t-error (left) and planning time speedup (right). The overall t-error is comparable in mean to the training, but increases with smaller k for the cautious user type. The aggressive user shows the highest error, with a mean of 1.8 for $k = 50$. The planning speedup shows a similar trend as the training data. However, the deviations differ more between user types. All three users yield median speedup factors of ≈ 30 for $k = 25$, but are still above 10 for $k = 100$.

We conclude that the learned control set $(\hat{\mathcal{E}}, \hat{\boldsymbol{w}})$ achieves good t-errors in both training and test environments while allowing for a substantial planning time speedup. Increasing k, the t-error tends to decrease, approaching the error of the estimation. Even though the performance also decreases with growing k, we still obtain an improvement of more than a

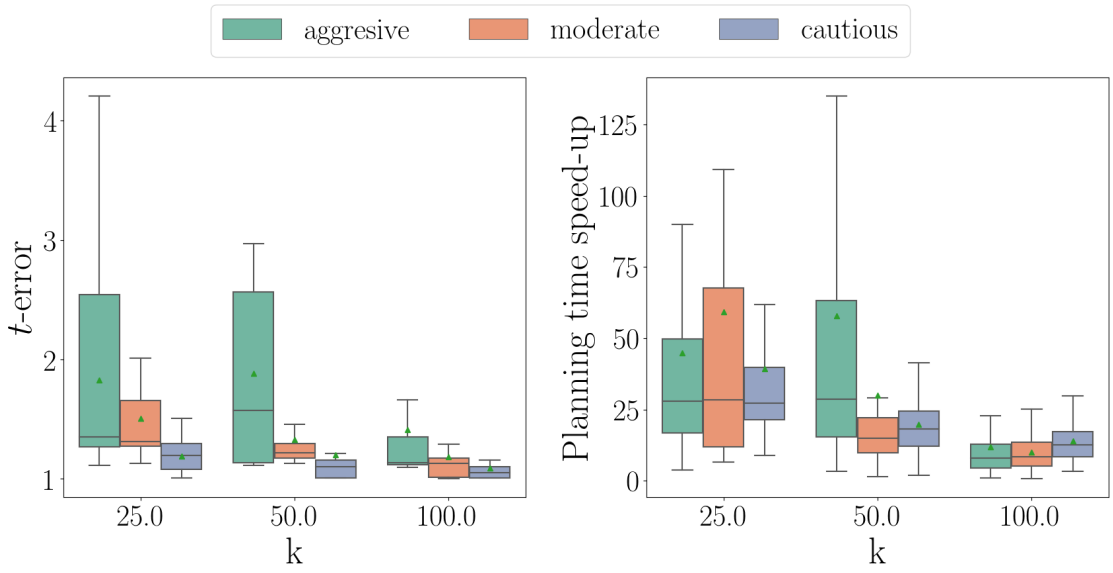


Figure 8.5: Test data: The t-error and planning time speedup for different sizes of the MKSCS and the different user types.

factor of 10 for $k = 100$ compared to paths planned using the complete connection set $\mathcal{B}$.

8.4 Discussion

We studied a novel approach for computing a control set that captures user preferences. First we find an estimate $\hat{\boldsymbol{w}}$ of the user weights, given data. Based on the estimate weights trajectories for all pairs $(s, j) \in \mathcal{B}$ are computed. These trajectories are calculated to minimize a user cost function that is a linear combination of weights $\hat{\boldsymbol{w}}$ and trajectory features. Next, a set of connections $\hat{\mathcal{E}} \subseteq \mathcal{B}$ and its associated trajectories is determined. This control set minimizes the relative error of trajectories generated. We illustrate how both $\hat{\boldsymbol{w}}$ and $\hat{\mathcal{E}}$ are computed, and validate our findings using a clothoid trajectory planner. The results illustrate that the control set $(\hat{\mathcal{E}}, \hat{\boldsymbol{w}})$ is able to capture a user's preferences while greatly reducing online computation time relative to using the full connection set.

In future, this approach should be tested with real-world data, e.g., recorded driving behaviour. The use of other trajectory planners such as polynomial splines or Bezier curves could be used to investigate generalizability. Finally, these results should be extended to account for situational user weights.

Chapter 9

Learning User Preferences from Corrections on State Lattices

A version of this chapter was published in the proceedings of the IEEE International Conference on Robotic and Automation (ICRA) 2020 [111].

In this chapter we return to the specification revision problem that we studied throughout Chapter 4 to 6. We address one of the main problem of the employed active preference learning framework: *What if the user does not like either of the presented paths?* Forcing the user to choose among two equally bad choices can be frustrating for the users, since they might feel impotent to tell the robot what they want – which was reported by some participants in our user study in Chapter 5. can lead to inefficient learning since it does not learn much about the user preferences, and users might be more inaccurate in their choice.

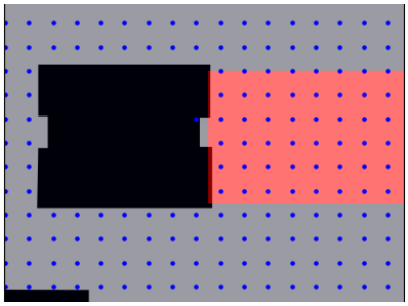
To address this issue we study a richer form of user feedback: Corrections. Given a potentially suboptimal candidate solution a user changes parts of the trajectory such that it better fits their preferences. This approach is a compromise between learning from demonstrations and active preference learning: Users are not required to show the entire trajectory to the robot since this be challenging in complex environments. On the other hand, the robot presents the user with their current best solution and gives the user the opportunity to correct the behaviour, which might allow for much quicker convergence than if the user would have to go through many iterations of active preference learning.

Similar to Problem 3, we consider user preferences for how a robot should accomplish its task that can be described by constraints on the robot's task and action space. A set of such constraints defines rewards and penalties for corresponding parts of the task space,

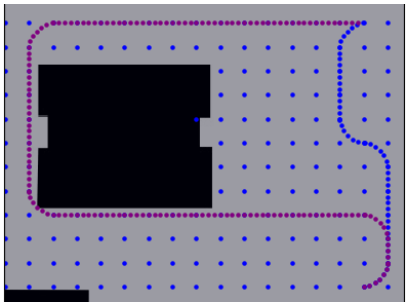
usually expressed by weights, which can then be considered by the robot's motion planner. In our previous framework (Chapter 4) we ask users to provide an initial specification, i.e., define constraints, and then revise the specification through active preference learning. We now modify the problem setup: User do not provide a specification, but we still assume that their preferences for robot motions could be expressed by a set of constraints. Following an interactive learning approach, we iteratively suggest a solution for a task, i.e., a path from a start to a goal location, and then ask the user to provide a correction if that solution does not fit their preferences.

The corrections can be provided on an interface showing a map of the environment together with the presented path. The user can choose to correct the parts of the presented path they dislike by either setting via-points or by drawing an alternative sub-path. From the difference between the presented solution and its correction we learn about the user's preferences for robot behavior in different parts of the environment. We illustrate this in Figure 9.1. In (a) the environment is shown together with a penalty area, i.e., the user prefers that a robot does not traverse the red-shaded area on the right. We do not ask the user to specify this region, but rather show them the current optimal solution computed by the motion planner. The user then provides a correction, as shown in (b), from which the planner learns that there must be some user preference making the presented path inferior.

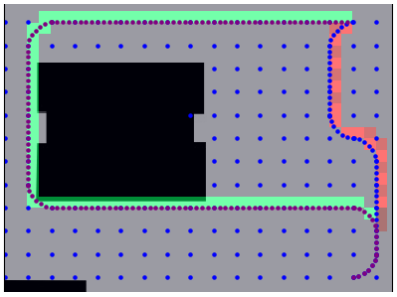
Given a set of tasks, users are queried for corrections over multiple iterations. From the user feedback, we learn about user preferences about the robot's state relative to the *environment*, e.g., areas where robot traffic is encouraged or discouraged, and preferences about the robot's *motion* describing control actions that should be avoided such as sharp turn maneuvers. It is neither necessary nor realistic to expect that the learned preferences exactly equal the hidden ones. Rather, the objective is that the learned preferences result in a similar behaviour, i.e., the planner finds paths that are similar to the ones the planner would find after a user had precisely defined constraints. Further, the learning should generalize the information obtained from the corrections, as illustrated in Figure 9.1 (c) and (d). When updating preferences without any generalization, the planner might only avoid the exact part of the environment traversed by the presented path, or prefer to use the part of environment used by the correction, shown in sub-figure (c). As a result, in the next iteration it might propose a path that is only slightly different than the one presented previously. The goal of generalizing the learned information is to infer the user's intent when they provide a correction. This can be done by updating the weights for the area around the presented path and the correction, as illustrated in sub-figure (d). This potentially allows for faster learning and might improve the performance on tasks for which no corrections were obtained, i.e., that were not available during the interaction with the user.



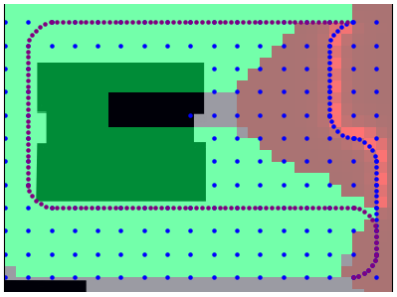
(a) Hidden user preferences on the environment.



(b) Suggested path (blue) and user correction (purple).



(c) Sparse preferences learned from one correction.



(d) Generalizing preferences learned from one correction.

Figure 9.1: Example of the learning from correction framework, with an environment with obstacles (black), avoidance areas (red) and areas where robot traffic is encouraged (green).

Contributions We study a novel framework for learning user preferences from corrections for a state lattice motion planner. In our previous work on specification revision (Chapters 4 to 6), our linear cost function considered only *environment* features characterizing robot behaviour based on its current location in the environment (i.e., local features). Using a lattice planner allows us to include *motion* features that allow us to model preferences about control actions across the entire state lattice (i.e., global features). From a user's correction to a presented path we derive inequalities about weights for both types of features. Given a sequence of presented paths and their corrections, we update the weights to satisfy the user's preferences while generalizing the information learned about weights to improve the performance. We prove completeness of our algorithm and demonstrate its performance in simulations. Thereby, we show the weights learned over 20 iterations result in paths similar to the optimal paths for tasks used during learning as well as for a set of test tasks.

9.1 Problem Formulation

In the problem we consider the following input:

- A directed graph $G = (V, E, t)$ induced by a state lattice [81] with a control set, i.e., motion primitives, B. The graph G encodes a robotic road-map of the environment. The travel times t_i are non-negative for all $e_i \in E$.
- A set of training tasks, $\mathcal{Z}^{\texttt{train}}$ and a set of *hidden* test tasks $\mathcal{Z}^{\texttt{test}}$. Both contain ordered pairs $\{(s_1, g_1), (s_2, g_2), \dots\}$ where s_j and g_j are vertices on G and constitute the start and goal, for which we want to find the shortest path that satisfies all user preferences.
- A grid-based cost map M [81] of the environment. Each cell $\mu_i \in M$ has a *hidden* user weight w_i^* where $w_i^* \in [0, w^{\max}]$. Weights $w_i > 1$ express that it is undesired for robots to visit cell μ_i while $w_i < 1$ correspond to a reward. We collect all weights in a vector $\boldsymbol{w}^* \in \mathbb{R}_{\geq 0}^{|M|}$.
- A *hidden* vector $\boldsymbol{u}^* \in \mathbb{R}_{\geq 1}^{|B|}$ where each u_l^* describes the user's preference for the robot using the motion primitive $b_l \in B$.
- Vectors $\boldsymbol{w}^0$ and $\boldsymbol{u}^0$ with prior weights.
- A user providing feedback: Given a path P for a task (s_i, g_i) the user provides a corrected path Q.

The graph G can be combined with weights $\boldsymbol{w}$ and $\boldsymbol{u}$ to obtain the graph $G^{\boldsymbol{w},\boldsymbol{u}} = (V, E, c)$. Given an edge e and its primitive path, let $\phi_j(e)$ be the length of the section of its primitive path in the cell μ_j. Further, let $\eta_l(e)$ be a binary function indicating if the edge e is an instance of the primitive b_l. The cost c_i of an edge e_i is then defined as

$$c_i = \sum_{\mu_j \in M} \phi_j(e_i) \cdot w_j + \sum_{b_l \in B} t_l \cdot \eta_l(e_i) \cdot u_l. \tag{9.1}$$

Thus, each w_j expresses a relative reward or penalty for an edge passing through the cell μ_j. The weight u_l describes a preference for using the corresponding motion primitive. As $w_j \geq 0$ for all μ_j and $u_l \geq 1$ for all b_l, we ensure that $c_i \geq 0$ always holds. As a consequence, the graph cannot contain negative cycles. As $\boldsymbol{w}^*$ and $\boldsymbol{u}^*$ are hidden we can only obtain the graph G^0, composed of G, $\boldsymbol{w}^0$ and $\boldsymbol{u}^0$. The objective is to learn about all

weights $\boldsymbol{w}$ and $\boldsymbol{u}$ such that the behaviour of a robot planning with the learned weights is equivalent to the behaviour described by $\boldsymbol{w}^*$ and $\boldsymbol{u}^*$, i.e., that the optimal solution for all tasks is the same.

9.1.1 User Cost Function

We model the user to have an internal linear cost function for paths, extending our prior work in [107] to incorporate both environment and motion features. Given a graph $G^{\boldsymbol{w},\boldsymbol{u}}$ combined from G with some $\boldsymbol{w}$ and $\boldsymbol{u}$, the cost of a path is

$$c(P, \boldsymbol{w}, \boldsymbol{u}) = \sum_{e_i \in P} c_i. \tag{9.2}$$

Let $\boldsymbol{\phi}(P)$ be the vector summarizing the path length for each cell $\mu \in M$, defined as

$$\boldsymbol{\phi}(P) = \left[\sum_{e \in P} \phi_1(e) \ \sum_{e \in P} \phi_2(e) \ \dots \ \sum_{e \in P} \phi_{|M|}(e) \right]. \tag{9.3}$$

Similarly, $\boldsymbol{\eta}(P)$ counts how often each primitive $b \in B$ is used on the path P and multiplies it with the duration of b:

$$\boldsymbol{\eta}(P) = \left[\sum_{e \in P} t_1 \eta_1(e) \ \sum_{e \in P} t_2 \eta_2(e) \ \dots \ \sum_{e \in P} t_{|B|} \eta_{|B|}(e) \right]. \tag{9.4}$$

This allows us to rewrite the cost function as

$$c(P, \boldsymbol{w}, \boldsymbol{u}) = \boldsymbol{\phi}(P)\boldsymbol{w} \ + \boldsymbol{\eta}(P)\boldsymbol{u}. \tag{9.5}$$

Similar to reward functions in reinforcement learning, $\boldsymbol{\phi}(P)$ and $\boldsymbol{\eta}(P)$ describe features of a path, which are then weighted by $\boldsymbol{w}$ and $\boldsymbol{u}$, respectively. We call $\boldsymbol{\phi}(P)$ *environment* features as they describe a preference for the robot's behaviour in particular regions of the environment. The vector $\boldsymbol{\eta}(P)$ expresses *motion* features for the robot; they describe preferences over the motion primitives in the lattice and apply globally. For instance a user might want the robot to avoid sharp curves to reduce risk, or left turns as they can affect performance in dense traffic.

9.1.2 Problem Statement

We consider arbitrary weights $\boldsymbol{w}$ and $\boldsymbol{u}$ defining a graph $G^{\boldsymbol{w},\boldsymbol{u}}$, as well as the hidden optimal weights $\boldsymbol{w}^*$ and $\boldsymbol{u}^*$ defining a graph G^*. The graphs $G^{\boldsymbol{w},\boldsymbol{u}}$ and G^* have the same vertices and edges, but different costs c on the edges. For any task (s_j, g_j) we can find a shortest path on $G^{\boldsymbol{w},\boldsymbol{u}}$, denoted by P_j and a shortest path on G^*, denoted by P_j^*. Finally, let $c(P_j, \boldsymbol{w}^*, \boldsymbol{u}^*)$ denote the cost of a path P_j on G^*. This allows us to define the loss function for a set of tasks $\mathcal{Z}$ as the summed relative error in cost:

$$\texttt{Err}(\mathcal{Z}, \boldsymbol{w}, \boldsymbol{u}) = \sum_{(s_j,g_j)\in\mathcal{Z}} \frac{c(P_j, \boldsymbol{w}^*, \boldsymbol{u}^*)}{c(P_j^*, \boldsymbol{w}^*, \boldsymbol{u}^*)} - 1. \tag{9.6}$$

This error is a special case of our general error function (3.4), we now explicitly distinguish between local and global features and thus weights $\boldsymbol{w}$ and $\boldsymbol{u}$. We notice that this error cannot be calculated as $\boldsymbol{w}^*$ and $\boldsymbol{u}^*$ are hidden. Nonetheless, the objective is to find weights $\boldsymbol{w}$ and $\boldsymbol{u}$ that minimize the error for all tasks.

Problem 9 (Learning user preference)**.** Given G and $\mathcal{Z}^{\texttt{train}}$ and a budget of K user interactions, find weights $\begin{bmatrix}\boldsymbol{w}^K & \boldsymbol{u}^K\end{bmatrix}$ where

$$\begin{aligned}\begin{bmatrix}\boldsymbol{w}^K & \boldsymbol{u}^K\end{bmatrix} = \underset{[\boldsymbol{w}'\ \boldsymbol{u}']}{\arg\min}\ & \texttt{Err}(\mathcal{Z}^{\texttt{train}} \cup \mathcal{Z}^{\texttt{test}}, \boldsymbol{w}', \boldsymbol{u}') \\ \text{s.t. } & 0 \le w'_j,\ j = 1, 2, \ldots, |M| \\ & 1 \le u'_l,\ l = 1, 2, \ldots, |B|.\end{aligned} \tag{9.7}$$

9.2 Approach

We present our framework for solving Problem 9 in Algorithm 7. The approach is similar to active preference learning [87, 109], but while an active preference learning algorithm proposes at least two new paths in each iteration, LfC presents only one path and asks the user for a correction. In contrast to our previous work [109, 107], the user is not required to provide any initial input such as specifying constraints. In each iteration k we maintain an estimate of the weights $\begin{bmatrix}\boldsymbol{w}^k & \boldsymbol{u}^k\end{bmatrix}$, based on the user corrections obtained so far. A planner can compute shortest paths for all tasks based on $\begin{bmatrix}\boldsymbol{w}^k & \boldsymbol{u}^k\end{bmatrix}$, from which we randomly select one path that is shown to the user (line 5). After the user provides a correction (line 6), the weights are updated (line 11).

In the next sections we describe our user model for how they provide corrections and show how weights are updated. Finally, we prove completeness of the algorithm.

Algorithm 7: Learning from Corrections.

Input: G, $\mathcal{Z}^{\texttt{train}}$, $[\boldsymbol{w}^0\ \boldsymbol{u}^0]$, K
Output: $\left[\boldsymbol{w}^k\ \boldsymbol{u}^k\right]$

1 Initialize $\Psi = \emptyset$, $\mathcal{Z} = \mathcal{Z}^{\texttt{train}}$
2 **for** $k = 1$ *to* K **do**
3 **if** $\mathcal{Z} = \emptyset$ **then**
4 **return** $\left[\boldsymbol{w}^k\ \boldsymbol{u}^k\right]$
5 $P^k,\ (s_j, g_j) \leftarrow \texttt{Sample\_new\_path}(\mathcal{Z})$
6 $Q^k \leftarrow \texttt{Get\_user\_correction}(P^k)$
7 **if** $Q^k = \emptyset$ **then**
8 $\mathcal{Z} = \mathcal{Z} \setminus (s_j, g_j)$
9 **continue**
10 $\Psi \leftarrow \Psi \cup \{P^k, Q^k\}$
11 $\left[\boldsymbol{w}^k\ \boldsymbol{u}^k\right] \leftarrow \texttt{Update}(\boldsymbol{w}^0, \boldsymbol{u}^0, \boldsymbol{w}^{k-1}\boldsymbol{u}^{k-1}, \Psi)$
12 **return** $\left[\boldsymbol{w}^k\ \boldsymbol{u}^k\right]$

9.2.1 User Model

In our framework users are presented with a path for a task (s_j, g_j) and provide feedback in the form of corrections: Given a path P they return a corrected path Q. We assume that the correction fits the user preferences better than P, i.e., has a lower cost with respect to the hidden weights, as summarized in the following Assumption:

Assumption 2 (User feedback)**.** Given a path P with $c(P, \boldsymbol{w}^*, \boldsymbol{u}^*) > c(P^*, \boldsymbol{w}^*, \boldsymbol{u}^*)$ the user returns a corrected path Q such that

$$c(Q, \boldsymbol{w}^*, \boldsymbol{u}^*) < c(P, \boldsymbol{w}^*, \boldsymbol{u}^*). \tag{9.8}$$

If $c(P, \boldsymbol{w}^*, \boldsymbol{u}^*) = c(P^*, \boldsymbol{w}^*, \boldsymbol{u}^*)$ an empty set is returned.

In case the presented path is optimal and thus an empty set was returned the algorithm cannot learn about the weights any more from that task and discards it (line 8).

9.2.2 Learning from Corrections

In order to update the weights in line 11 of Algorithm 7 we show how information about weights is derived from user corrections. Using the user cost function from equation (9.5)

allows us to rewrite inequality (9.8) as

$$(\boldsymbol{\phi}(Q) - \boldsymbol{\phi}(P))\, \boldsymbol{w}^* + (\boldsymbol{\eta}(Q) - \boldsymbol{\eta}(P))\, \boldsymbol{u}^* < 0. \tag{9.9}$$

To avoid a strict inequality we use some small ϵ:

$$(\boldsymbol{\phi}(Q) - \boldsymbol{\phi}(P))\, \boldsymbol{w}^* + (\boldsymbol{\eta}(Q) - \boldsymbol{\eta}(P))\, \boldsymbol{u}^* \leq -\epsilon. \tag{9.10}$$

This then defines a half-space for feasible weights for each (P^k, Q^k). Hence, any weight $[\boldsymbol{w}\ \boldsymbol{u}]$ within that half-space guarantees that Q^k has a lower cost than P^k. Further, as we can only observe corrections, all weights within the half-space are indistinguishable from $[\boldsymbol{w}^*\ \boldsymbol{u}^*]$ with respect to the user cost function, and thus we can pick any such $[\boldsymbol{w}\ \boldsymbol{u}]$.

9.2.3 Updating Weights

Feasible weights Our primary objective is to find weights $\boldsymbol{w}^k$ and $\boldsymbol{u}^k$ that minimize the error $\texttt{Err}(\boldsymbol{w}^k, \boldsymbol{u}^k)$. However, as $\boldsymbol{w}^*$ and $\boldsymbol{u}^*$ are hidden $\texttt{Err}(\boldsymbol{w}^k, \boldsymbol{u}^k)$ cannot be computed. Nonetheless, we say weights are feasible if they are consistent with the user feedback, i.e., satisfy the inequalities based on Assumption 2. Given a sequence of paths and their user corrections $\Psi = \left(P^1, Q^1, P^2, Q^2, \ldots, P^k, Q^k\right)$, we can intersect the half-spaces described in equation (9.10) to define a convex set of feasible weights. This allows us to write the objective as a constraint in the optimization problem: To minimize $\texttt{Err}(\boldsymbol{w}^k, \boldsymbol{u}^k)$ we have to pick weights $\boldsymbol{w}^k$ and $\boldsymbol{u}^k$ such that equation (9.10) holds for all $k = 1, 2, \ldots, K$.

Generalization Among all weights that satisfy the inequalities from Assumption 2, we want to choose $\boldsymbol{w}^k$ and $\boldsymbol{u}^k$ that *generalize* the information we obtain for each (P, Q). As we express the objective from equation (9.7) as constraints, we can choose a new objective function $g(\boldsymbol{w}, \boldsymbol{u})$, leading to the optimization problem

$$\begin{aligned}
\begin{bmatrix}\boldsymbol{w}^K & \boldsymbol{u}^K\end{bmatrix} = \ & \underset{[\boldsymbol{w}'\ \boldsymbol{u}']}{\arg\min}\ g(\boldsymbol{w}', \boldsymbol{u}') \\
\text{s.t.} \quad & \left(\boldsymbol{\phi}(Q^k) - \boldsymbol{\phi}(P^k)\right) \boldsymbol{w}' \\
& + \left(\boldsymbol{\eta}(Q^k) - \boldsymbol{\eta}(P^k)\right) \boldsymbol{u}' \\
& \leq -\epsilon, \ \text{for } k = 1, 2, \ldots, K \\
& 0 \leq w'_j, \ \text{for } j = 1, 2, \ldots, |M| \\
& 1 \leq u'_l, \ \ \text{for } l = 1, 2, \ldots, |B|.
\end{aligned} \tag{9.11}$$

The idea of generalization is that the learned weights do not only affect edges that belong to the presented path or the obtained correction, but to rather obtain more homogeneous *environment* weights, i.e., encourage neighbouring cells to have the same weight. This is motivated by observations from previous user studies, where users typically have preferences for areas in the environment instead of single locations [107]. Let $N(\mu_j)$ be the set of cells in M that are adjacent to μ_j. We want to minimize the mean square difference in weights between neighbours. On the other hand, it is desirable to avoid weights $w_j \neq 1$. This can be captured by an l_2-norm regularization as used in machine learning [73]. Combining the generalization with the regularization, we obtain

$$g(\boldsymbol{w}) = \lambda \sum_{\mu_i \in M} \sum_{\mu_j \in N(\mu_i)} (w_i - w_j)^2 + (1-\lambda) \sum_{\mu_i \in M} (w_i - 1)^2. \tag{9.12}$$

Thereby, λ takes values in $[0, 1)$ and balances between generalization and regularization. We do not allow $\lambda = 1$ as this would result in all weights being close to zero. For the *motion* features we only use a regularization $g(\boldsymbol{u}) = \sum_{b_l \in B}(u_l - 1)^2$. Finally, we define the objective in equation (9.7) as $g(\boldsymbol{w}, \boldsymbol{u}) = g(\boldsymbol{w}) + g(\boldsymbol{u})$. We notice that $g(\boldsymbol{w}, \boldsymbol{u})$ is a convex quadratic function implying that the optimization problem in equation (9.11) can be solved in polynomial time.

9.2.4 Completeness

Based on the user model and the proposed weight update we now show that Algorithm 7 is complete.

Proposition 5 (Completeness)**.** For each problem instance there exists a finite K such that Algorithm 7 finds an optimal solution, i.e., returns weights $\begin{bmatrix}\boldsymbol{w}^K \ \boldsymbol{u}^K\end{bmatrix}$ such that the error $\texttt{Err}(\boldsymbol{w}^K, \boldsymbol{u}^K) = 0$.

Proof. Consider the case that the path P^k presented to the user is optimal. Then the corresponding task (s_j, g_j) gets removed from $\mathcal{Z}$ and ${}^{c(P_j^k, \boldsymbol{w}^*, \boldsymbol{u}^*)}/_{c(P_j^*, \boldsymbol{w}^*, \boldsymbol{u}^*)} - 1 = 0$ (line 8 in Algorithm 7).
If P^k is not optimal, a correction Q^k satisfying Assumption 2 is obtained from which we derive an inequality as in equation (9.10). Then all weights $\begin{bmatrix}\boldsymbol{w}^l \ \boldsymbol{u}^l\end{bmatrix}$ where $l > k$ satisfy this inequality, implying that for any $l > k$ the path P^k is never optimal for any feasible weight vector $\begin{bmatrix}\boldsymbol{w}^l \ \boldsymbol{u}^l\end{bmatrix}$ and thus cannot be presented again. That guarantees that each update removes at least one path from the set of all paths that could be shown for the

same task in a later iteration. For every pair (s_j, g_j) there exists only a finite number of paths; hence, a path P^l where $c(P_j^l, \boldsymbol{w}^*, \boldsymbol{u}^*) = c(P_j^*, \boldsymbol{w}^*, \boldsymbol{u}^*)$ must be presented to the user after a finite number of iterations, and ${}^{c(P_j^l, \boldsymbol{w}^*, \boldsymbol{u}^*)}/_{c(P_j^*, \boldsymbol{w}^*, \boldsymbol{u}^*)} - 1 = 0$ for every task (s_j, g_j) is achieved. □

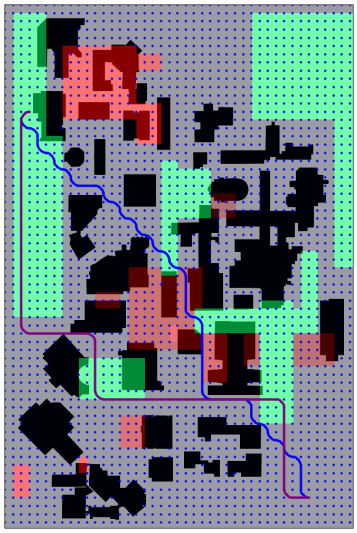
Proposition 5 ensures that the presented framework learns weights for robot motion planning that result in the same behaviour as described by the hidden optimal weights $\boldsymbol{w}^*$. In the next section we evaluate the performance in a realistic transportation scenario.

9.3 Evaluation

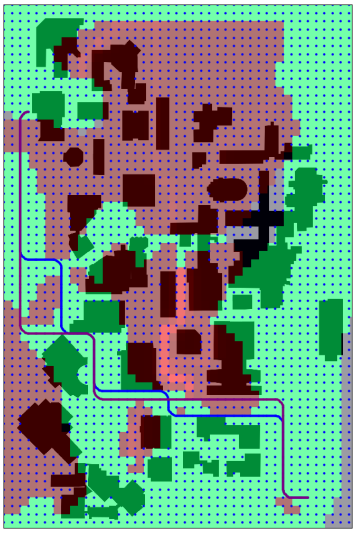
In the evaluation we use a state lattice with 11,008 vertices, based on the layout of the campus of the University of Waterloo, and a cost map with 2,752 quadratic cells. The user corrections are simulated. Each simulated user has a hidden cost map with weights $\boldsymbol{w}^*$ such that the map is structured into neighbourhoods of multiple cells that have equal weights. This allows to define areas in the environment that are rewarded or penalized, i.e., have a weight different than one, which we call constraints. In each trial the user weights are generated by randomly drawing from a subset of 60 constraints. These constraints were predefined such that narrow gaps on the environment might be blocked by penalty constraints while wide open areas are usually rewarded. An example specification with 25 constraints is illustrated in Figure 9.2 (a), Further, in the experiments we use the control set $B = \{(0, 1, 0), (1, 1, \pi/2), (-1, 1^{-\pi/2}), (2, 1, \pi/2), (-2, 1, {}^{-\pi/2}), (1, 2, \pi/2), (-1, 2, {}^{-\pi/2})\}$, always with *motion* user preferences that heavily penalize left turns and slightly penalize right turns, as shown in sub-figure (c).

We consider two types of user behaviour: *Optimal users* that always provide a correction that is an optimal solution for the task, and *uniform users* who randomly generate a correction satisfying Assumption 2. As the optimal weights $\boldsymbol{w}^*$ and $\boldsymbol{u}^*$ are known to the simulated user, optimal paths can be computed using shortest path search. For uniform users, the correction is the shortest path Q for some weights $\bar{\boldsymbol{w}}$ and $\bar{\boldsymbol{u}}$. We generate $\bar{\boldsymbol{w}}$ using a random walk: Let $\boldsymbol{w}$ be the weight for which the presented path P is optimal. Then, the random walk to find $\bar{\boldsymbol{w}}$ is constrained by the polyhedron $\{\boldsymbol{w}' \in [0, w^{\max}] | \min\{w_i, w_i^*\} \leq w_i \leq \max\{w_i, w_i^*\}, \forall \mu_i \in M\}$. This ensures that Q has a lower cost than P.

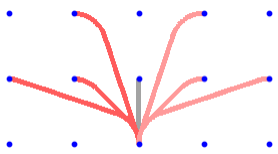
All experiments were repeated for 24 trials where each trial runs Algorithm 7 with $K = 20$ and a uniform prior for $\boldsymbol{w}$ and $\boldsymbol{u}$. In every trial the set of training and test tasks are randomly generated pairs of vertices. The run time for the quadratic program with 2752 variables is $\approx 9s$; finding a new query takes $\approx 11s$ for 5 training tasks and up to $20s$



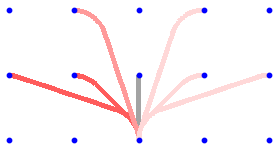
(a) Hidden user weights $\boldsymbol{w}^*$ with initially presented path and optimal correction.



(b) Learned weights $\boldsymbol{w}^{20}$ using $\lambda = 0.5$ with learned path and optimal correction.



(c) Hidden environment user weights $\boldsymbol{u}^*$.



(d) Learned motion user weights $\boldsymbol{u}^{20}$.

Figure 9.2: Environment with learned environment and motion weights. Red cells indicate a weight of $w_j > 1$, i.e., a penalty. Green cells correspond to weights $w_j < 1$ belonging to a reward while grey cells have a weight of $w_j = 1$. The blue and purple lines show the learned path and the optimal path for an example task. In (c) and (d) red indicates weights $u_l > 1$. Even though the learned cost map in (b) has different weights on many cells than the optimal cost map in (a), they lead to similar optimal paths on the training and test task set.

for 10 training tasks. However, the query generation can potentially be sped up using a parallel implementation.

Experiment 1 - Generalization First, we investigate how different values for λ affect the algorithm. We use an optimal user, 25 constraints, 5 training and 20 test tasks. We illustrate the relative errors $\texttt{Err}(\mathcal{Z}^{\texttt{test}}, \boldsymbol{w}^K, \boldsymbol{u}^K)$ and $\texttt{Err}(\mathcal{Z}^{\texttt{train}}, \boldsymbol{w}^K, \boldsymbol{u}^K)$ before and after learning in Figure 9.3.

We observe no conclusive difference in the test error between $\lambda = 0.0$ and $\lambda = 0.5$. However, the final median training error is 0.09 for $\lambda = 0.0$, i.e., paths generated with the learned weights have 9% higher cost than those with the hidden user weight. The final median training error for $\lambda = 0.5$ is 0.01. Thus, using a generalization allows for more efficient learning on the training set, as it avoids presenting paths that are too similar to

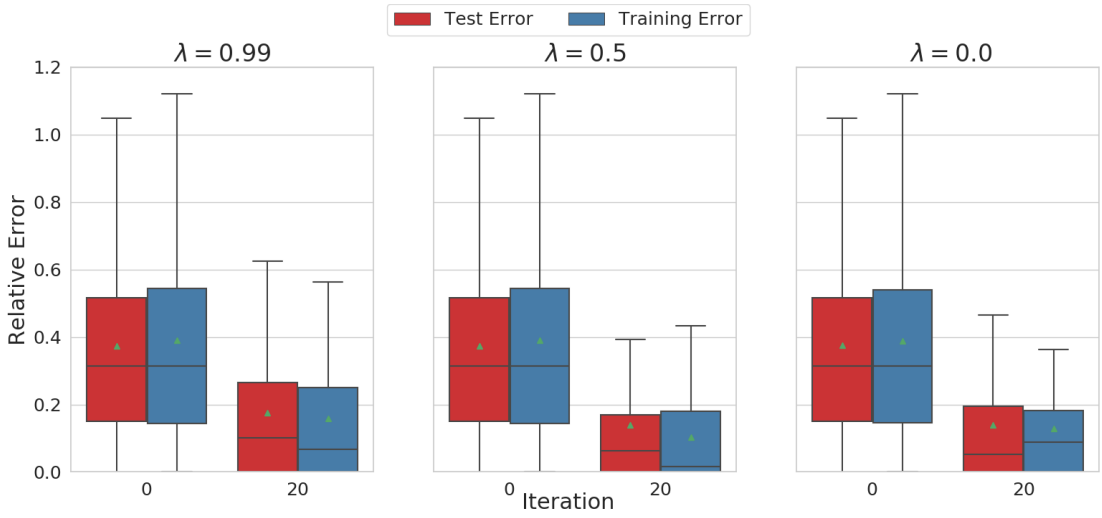


Figure 9.3: Learning from corrections with different values for λ.

those previously presented. On the other hand, the training and test error is worse for $\lambda = 0.99$ than for $\lambda = 0.5$. A large generalization overestimates the size of user constraints and leads to poor behaviour. In the subsequent experiments we use $\lambda = 0.5$.

Experiment 2 - Naive algorithm In the second experiment we compare the presented algorithm to a naive approach. Given a pair (P, Q) the naive approach updates the weights on the cost map by directly penalizing cells that are traversed by the presented path but not in the correction and rewarding cells which are traversed by only the correction. Hence,

$$w_j^{k+1} = \begin{cases} 2\, w_j^k & \text{if } \phi_j(P) > 0 \text{ and } \phi_j(Q) = 0, \\ 1/2\, w_j^k & \text{if } \phi_j(P) = 0 \text{ and } \phi_j(Q) > 0, \\ w_j^k & \text{otherwise.} \end{cases} \tag{9.13}$$

We use the same experimental setup as before, but with 10 training tasks to reduce the risk of over-fitting for the naive approach. The results are illustrated in Figure 9.4.

The naive approach shows moderate results for the training data with a final median error of 0.19. However, the progress on the test error is marginal with a final median of 0.41. In contrast, our algorithm improves on both sets of tasks, achieving a final median error of 0.06 on the training and 0.07 on the test set. We conclude by noting that the optimization problem in equation (9.11) generalizes about environment weights and achieves good performance on training instances. Moreover, the algorithm successfully generalizes

the information obtained from the corrections and thus achieves good results on the test tasks.

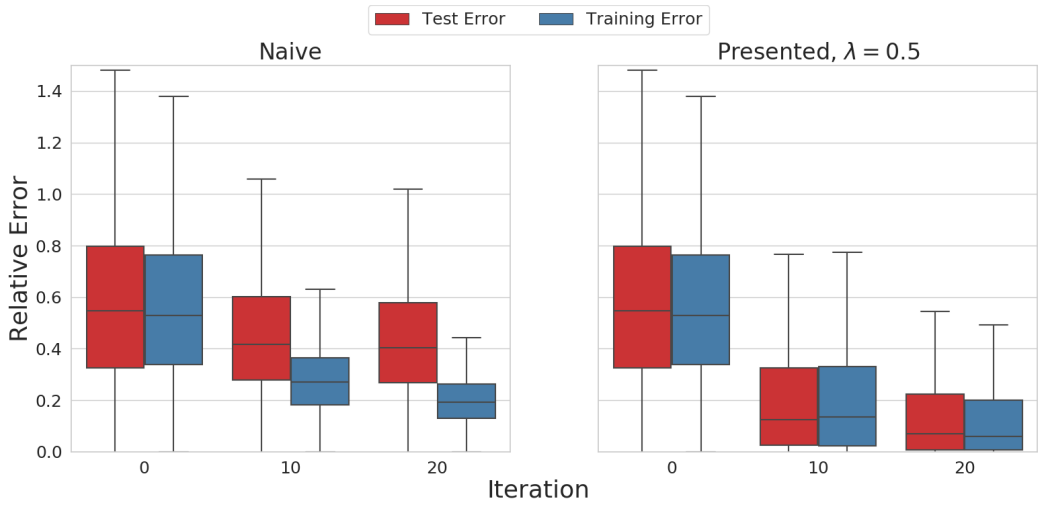


Figure 9.4: Comparison between the naive and the proposed approach, with $\lambda = 0.5$.

Experiment 3 - User types In this experiment we investigate how our algorithm behaves for the two different types of users, using the same setup as for the previous experiment. The results are summarized in Figure 9.5. Overall we observe that the learning behaves similarly for both user types. The optimal user shows a better result on the training data where 42% of trials achieve a final error of 0, compared to 31% for the uniform user. However, the final median values are 0.02 and 0.01 for the test and training error of the optimal user and 0.03 and 0.02 for the uniform user, respectively. Hence, the optimal user leads to only marginally better results. In conclusion, the third experiment shows that the presented algorithm learns about the user's preferences efficiently even when the user is not always providing optimal corrections.

9.4 Discussion

Summary We propose an LfC framework for a state lattice planner that learns about environment and motion user preferences. We update weights using a quadratic program, which is constrained by the assumption that users provide corrections that fit their preferences better than the presented path, while the objective function combines generalization

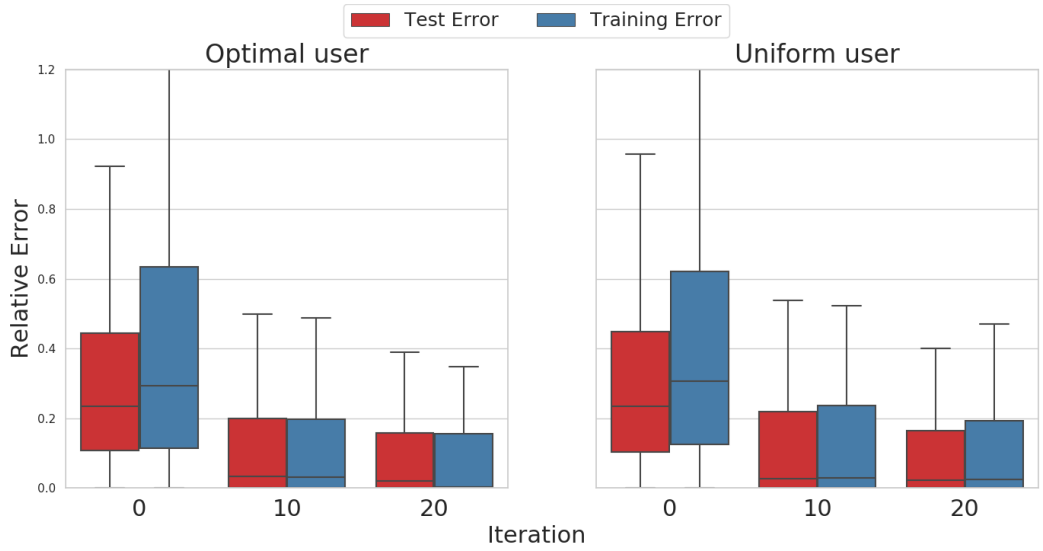


Figure 9.5: Relative error when using $\lambda = 0.5$ for optimal and uniform user corrections.

with regularization. In simulations we show that the proposed approach generates paths that closely approximate user preferences for both training and test tasks after only a few corrections, with either optimal or non-optimal user corrections.

Chapter 10

Discussion and future directions

A major challenge in human robot interaction is enabling inexperienced users to efficiently deploy autonomous robots. This thesis has studied interactive learning frameworks for instructing robots on *how* they can accomplish their tasks such that their behaviour meets user preferences. Common models assume that users evaluate robot behaviour based on a fixed set of features; learning the relative importance of such features allows a robot's motion planner to compute a corresponding optimal motion. We characterized an underlying structural property of such learning problems: The *sensitivity of motion planners*, i.e., changes in a cost function do not linearly relate to changes in the corresponding optimal trajectory. Thus, a robot does not need to recover the user's cost function exactly in order to find a user optimal trajectory. Therefore, we investigate learning techniques that hypothesize over different trajectories instead of different parameters in the cost function. We demonstrated the performance of our algorithms in several simulations, including comparisons to other state-of-the-art techniques. Moreover, a user study has shown the practicality of our work and highlighted that the proposed methods can help inexperienced users to deploy robots more efficiently.

10.1 Summary

In Chapters 4 to 6, we studied how active preference learning can be used to revise user specifications, i.e., learn the weights of user defined soft constraints on robot motions. After providing the initial specification, the user is iteratively presented with alternative robot paths and chooses the preferred one. We first considered a deterministic user model, where the user's choice always follows a linear cost function. This allows the robot to

iteratively collect inequality constraints on the user weights and thus shrink the set of candidate solutions. We extended the framework to a multi-task scenario and investigated its practicality in a user study in Chapter 5. Moreover, in Chapter 6 we introduced a probabilistic learning framework to relax the assumptions of the user model, and derived an efficient learning heuristic based on equivalence regions.

In Chapter 7 we extended our concept of learning about trajectories instead of weights of cost functions to general active preference learning problems. We study differences between an error metric based on cost function weights and our metric based on trajectory cost and propose a new heuristic for learning that always presents paths with the largest mutual relative error, discounted by the learned belief. We show that this approach outperforms another state-of-the-art method in simulations, especially for high dimensional problems, and that the generated queries are also easy to answer for the user.

In Chapter 8 we studied a more extensive framework for robots to adapt to user preferences: Instead of only learning a cost function and then planning optimal trajectories, we compute an optimal control set for the robot, i.e., subset of the robot's action space. Given a maximum number of controls a robot can use, we show how to optimally choose these controls, given an estimate of the cost function for the user preferences. In simulations we show that this method is suitable to design user specific motion planners, and that the limited size of the control set leads to substantial planning time benefits.

In chapter 9 we considered users *correcting* robot trajectories instead of choosing among presented alternatives. Expanding on our previous problem formulations, we distinguished between local and global user preferences and proposed an algorithm that learns about both simultaneously. Further, our approach generalizes the learned information, yielding good results in both training and test cases.

10.2 Unified Framework

An interesting future work direction for the specification learning problem is combining different methods of user feedback. Presented with a pair of paths (P^A, P^B) the user could: (1) choose P^A or P^B, or (2) provide a corrected path Q if both P^A and P^B are undesired, or (3) modify the specification, i.e., remove or add constraints.

The new forms of user feedback, correction and modification, are motivated from comments by participants of our user study in Chapter 5. Corrections are particularly useful when both presented paths are poor options for the user, or when one of the paths is *mostly* a good solution, but the user dislikes a relatively short part of it. Modifications on the

other hand give the user the option to change their specification when the user realises that the set of constraints they initially specified is incomplete or incorrect, or the robot interprets their constraints much differently than they expected.

Choice feedback When the user chooses option 1, we can learn from a user choosing path P^A over P^B (or vice versa) following our probabilistic models in Chapter 6 or 7. That is, we assume that path P^A has lower cost than path P^B, with some probability p^{AB}.

Correction feedback In the second form of user feedback, they design a correction Q of path P^A or P^B themselves. The path Q might share edges with P^A or P^B and thus be a correction of one of the presented alternatives instead of an entirely new path. If a correction is provided, we assume that it has a lower cost than both P^A and P^B. Thus, from a correction feedback, we learn *two* inequalities: Q is a better path than P^A and Q is a better path than P^B. Similarly to the choice feedback, we take a probabilistic perspective, i.e., assume that the cost of Q being lower than the cost of P^A and P^B holds with some probability p^{QA} and p^{QB}, respectively.

However, correction feedback in the context of specification revision brings up a new challenge: a correction might not be explainable with the current user specification, i.e., no weights exist such that Q has a lower cost than P^A and P^B. In that case the robot needs to propose a constraint that should be added to the specification. In Chapter 9 we showed how a cost map can be updated to incorporate corrections feedback. This approach could be adapted to propose new constraints. To accommodate for the correction it is desired that the cells whose weights are updated form a connected component and that these cells then all have the same weight. This would allow for grouping the updated cells into one new constraint.

Specification modification In the third form of user feedback, a user can also modify their specification. This form of user feedback is motivated from the study in Chapter 5 where some users stated that they would have liked to change their specification once they were presented with the resulting robot behaviour. Changes in the specification can be summarized by removing a subset of constraints or adding a set of new constraints. When the specification gets modified, the feedback obtained before is still assumed to be valid. Each path P that was presented so far can be re-evaluated with respect to the new specification. This allows us to derive inequalities on the weights and thus update the belief space.

In the future, the outlined unified framework needs to be formalized and validated. The methodology of constraint proposal after corrections requires a more detailed analysis to ensure the practicality of the approach: For instance if the proposed constraints drastically over- or under-fit the user preferences, users might reject them and may decide to modify the specification themselves. In this case the framework may be unappealing for the user. Finally, the unified framework would ideally be validated in practice by conducting a second user study.

10.3 Sample-free Active Preference Learning

Throughout this thesis we have studied various approaches for active preference learning: Vertex search for deterministic user feedback (Chapter 4), posterior minimization over equivalence regions (Chapter 6), regret minimization (Chapter 7) as well as the volume and entropy minimization proposed in [87] and [20]. Except for the first one (which only works for deterministic user models), all these approaches have in common that they require a finite set of paths as an input. That is, the set of all paths needs to be pre-sampled. This does not only hold for continuous state and action spaces, but also for the discrete case, since the number of all paths for a discrete planner can be exponential in the size of the number of discrete states, as shown in Chapter 6. The necessity of pre-sampling the solution space is a major hindrance for applying these techniques to new scenarios since pre-sampling can require significant computation time.

A potential method for avoiding pre-sampling is to approximate the query selection. Thereby we trade-off two objectives: (1) Fast computation of new queries to minimize the user wait time between iterations, and (2) ensuring a high quality of the approximation to still learn efficiently. The maximum regret approach proposed in Section 7 is particularly promising for an approximate solution since it already has a strong computational advantage compared to the equivalence region, volume or entropy approaches. Given a candidate query (P, Q), it does not require the computation of the tentative posterior distributions for the possible outcomes, i.e., the case the user prefers P over Q or vice versa.

Search space reduction Independent of the measure that is used to find a new pair of paths, we have seen two different frameworks for query generation: Selecting any two new paths (P, Q) as in [87, 20] or Chapter 7, or fixing one path of a query to be the one that the user preferred in the previous iteration, i.e., *keeping the current path* as in Chapter 4 to 6. In the latter case, the query selection consists of finding *one* new path Q such that some

measure for learning is optimized, given the current path P. We observe that, independent of the measure (volume, entropy, regret, etc.), keeping the current path can lead at most to the same greedy improvement of the measure as selecting two new paths, since the current path is available to the optimization for finding a new pair (P, Q). Further, when P is fixed to the previously preferred path, the query selection cannot be a submodular function, even when minimizing volume or entropy: The domain, i.e., the set of paths that can be presented to the user, changes in each iteration; thus, the maximum is chosen over a different set in each iteration. Nonetheless, fixing P to the previously preferred path also comes with the desirable property of simultaneous *exploration and exploitation*: By keeping P in the query more information about $\boldsymbol{w}^P$ will be gained; at the same time Q can be chosen freely and thus enables further exploration of the weight space. The regret based query selection can then be simplified to finding some path Q, given the current path P:

$$Q(P) = \arg\max_{\boldsymbol{w}^Q \in \Omega} \mathcal{R}^k(\boldsymbol{w}^P, \boldsymbol{w}^Q | U^k). \tag{10.1}$$

Incremental sampling algorithm A potential approximation that avoids pre-sampling paths could be incremental sampling. The idea is to iteratively build a set of sampled paths instead of computing a large set prior to the interactive learning. In each iteration the motion planning problem is solved for l sampled weights. Thus the set of paths over which (10.1) optimizes grows in each iteration. At the same time the runtime of each query generation is limited by the number of new samples in each iteration l.

Future work should further study the outlined approach to propose an algorithm for finding queries (P, Q) without pre-sampling. A key component of such an algorithm is drawing the new samples in each iteration. Instead of naively sampling from a uniform distribution, a heuristic for informative samples should be investigated, which could have great impact on the performance.

10.4 Other Future Work Directions

In addition to the unified framework and the sample free learning method, a major future direction is to further investigate the *regret* based learning method. This includes studying special cases of the presented problems such as discrete motion planners to potentially propose an approximation algorithm for Problem 2. Moreover, the min-max regret approach could be extended to more general problems such as the one of finding an optimal decision tree [22, 35].

Another extension to the presented work is allowing for richer user feedback such as an *equal preference* option in active preference learning. The work of [20] showed that giving users the option to not choose one of the presented paths can improve learning while [9] demonstrated that asking users for the reason behind their choice can increase the learning efficiency and the user trust. It would be interesting to investigate if such forms of richer user feedback are also beneficial for min-max regret queries.

In Chapter 8 we proposed a method for computing user optimal control sets for lattice planners based on an estimate of the user cost function. Future work should further validate the practicality of the proposed framework; in the scenario of autonomous driving this could be done with recorded real world driving data or using a driving simulator. Additionally, the approach should be validated for other scenarios such as motion planning for a manipulator robot. Moreover, the concept of learning a user specific motion planner (i.e., control set) instead of only learning the user cost function could also be applied to other interactive learning frameworks. In active preference learning the proposed trajectories could convey different user preferences more clearly when they are computed with respective control sets.

References

[1] Pieter Abbeel, Dmitri Dolgov, Andrew Y Ng, and Sebastian Thrun. Apprenticeship learning for motion planning with application to parking lot navigation. In *2008 IEEE/RSJ IROS*, pages 1083–1090. IEEE, 2008.

[2] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004.

[3] Baris Akgun, Maya Cakmak, Jae Wook Yoo, and Andrea Lockerd Thomaz. Trajectories and keyframes for kinesthetic teaching: A human-robot interaction perspective. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, pages 391–398. ACM, 2012.

[4] Riad Akrour, Marc Schoenauer, and Michèle Sebag. April: Active preference learning-based reinforcement learning. *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 116–131, 2012.

[5] G. T. Anderson and G. Yang. A proposed measure of environmental complexity for robotic applications. In *2007 IEEE International Conference on Systems, Man and Cybernetics*, pages 2461–2466, October 2007.

[6] Galen Andrew and Jianfeng Gao. Scalable training of l 1-regularized log-linear models. In *Proceedings of the 24th international conference on Machine learning*, pages 33–40, 2007.

[7] Brenna D. Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469 – 483, 2009.

[8] Aaron Bangor, Philip T. Kortum, and James T. Miller. An empirical evaluation of the system usability scale. *International Journal of Human-Computer Interaction*, 24(6):574–594, 2008.

[9] Chandrayee Basu, Mukesh Singhal, and Anca D. Dragan. Learning from richer human guidance: Augmenting comparison-based learning with feature queries. In *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, HRI '18, pages 132–140, New York, NY, USA, 2018. ACM.

[10] Dimitris Bertsimas and John Tsitsiklis. *Introduction to Linear Optimization*. Athena Scientific, 1st edition, 1997.

[11] Amit Bhatia, Lydia E Kavraki, and Moshe Y Vardi. Sampling-based motion planning with temporal goals. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2689–2696. IEEE, 2010.

[12] Aude G Billard, Sylvain Calinon, and Rüdiger Dillmann. *Learning from humans*, pages 1995–2014. Springer, 2016.

[13] A. Blidaru, S. L. Smith, and D. Kulić. Assessing user specifications for robot task planning. In *2018 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, pages 72–79, Aug 2018.

[14] Andreea Bobu, Dexter RR Scobee, Jaime F Fisac, S Shankar Sastry, and Anca D Dragan. Less is more: Rethinking probabilistic models of human behavior. In *Proceedings of the 2020 ACM/IEEE International Conference on Human-Robot Interaction*, pages 429–437, 2020.

[15] A. Botros and S. L. Smith. Computing a minimal set of t-spanning motion primitives for lattice planners. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2328–2335, 2019.

[16] Alexander Botros, Nils Wilde, and Stephen L. Smith. Learning control sets for lattice planners from user preferences. In *The 14th International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, volume : To appear, 2020.

[17] Darius Braziunas. Computational approaches to preference elicitation. *Department of Computer Science, University of Toronto, Tech. Rep*, page 62, 2006.

[18] Daniel S Brown, Yuchen Cui, and Scott Niekum. Risk-aware active inverse reinforcement learning. In *Conference on Robot Learning*, pages 362–372, 2018.

[19] Daniel S. Brown, Wonjoon Goo, and Scott Niekum. Better-than-demonstrator imitation learning via automatically-ranked demonstrations, 2019.

[20] Erdem Bıyık, Malayandi Palan, Nicholas C. Landolfi, Dylan P. Losey, and Dorsa Sadigh. Asking easy questions: A user-friendly approach to active reward learning, 2019.

[21] Maya Cakmak and Andrea L Thomaz. Designing robot learners that ask good questions. In *2012 7th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 17–24. IEEE, 2012.

[22] Venkatesan T Chakaravarthy, Vinayaka Pandit, Sambuddha Roy, Pranjal Awasthi, and Mukesh Mohania. Decision trees for entity identification: Approximation algorithms and hardness results. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 53–62. ACM, 2007.

[23] Ernest Cheung, Aniket Bera, Emily Kubin, Kurt Gray, and Dinesh Manocha. Identifying driver behaviors using trajectory features for vehicle navigation. In *2018 IEEE/RSJ IROS*, pages 3445–3452. IEEE, 2018.

[24] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in Neural Information Processing Systems*, pages 4299–4307, 2017.

[25] John J Craig. *Introduction to robotics: mechanics and control, 3/E*. Pearson Education India, 2009.

[26] Nadia Creignou and Miki Hermann. On P completeness of some counting problems. Research Report RR-2144, INRIA, 1993.

[27] Y. Cui and S. Niekum. Active reward learning from critiques. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6907–6914, May 2018.

[28] Christian Daniel, Malte Viering, Jan Metz, Oliver Kroemer, and Jan Peters. Active Reward Learning. *Robotics: Science and Systems (RSS)*, 10(July), 2014.

[29] Ryan De Iaco, Stephen L Smith, and Krzysztof Czarnecki. Learning a lattice planner control set for autonomous vehicles. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 549–556. IEEE, 2019.

[30] Staffan Ekvall and Danica Kragic. Robot learning from demonstration: a task-level planning approach. *International Journal of Advanced Robotic Systems*, 5(3):33, 2008.

[31] G. E. Fainekos. Revising temporal logic specifications for motion planning. In *2011 IEEE International Conference on Robotics and Automation*, pages 40–45, May 2011.

[32] Cameron Finucane, Gangyuan Jing, and Hadas Kress-Gazit. Ltlmop: Experimenting with language, temporal logic and robot control. In *IROS 2010*, pages 1988–1993. IEEE, 2010.

[33] Thierry Fraichard and Alexis Scheuer. From reeds and shepp's to continuous-curvature paths. *IEEE Transactions on Robotics*, 20(6):1025–1035, 2004.

[34] Daniel Golovin and Andreas Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research*, 42:427–486, 2011.

[35] Daniel Golovin, Andreas Krause, and Debajyoti Ray. Near-optimal bayesian active learning with noisy observations. In *Advances in Neural Information Processing Systems*, pages 766–774, 2010.

[36] Matthew Gombolay, Anna Bair, Cindy Huang, and Julie Shah. Computational design of mixed-initiative human–robot teaming that considers human factors: situational awareness, workload, and workflow preferences. *IJRR*, 36(5-7):597–617, 2017.

[37] David Sierra González, Ozgur Erkent, Víctor Romero-Cano, Jilles Dibangoye, and Christian Laugier. Modeling driver behavior from demonstrations in dynamic environments using spatiotemporal lattices. In *2018 IEEE ICRA*, pages 1–7. IEEE, 2018.

[38] Daniel H Grollman and Aude Billard. Donut as i do: Learning from failed demonstrations. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3804–3809. IEEE, 2011.

[39] Tianyu Gu, Jason Atwood, Chiyu Dong, John M Dolan, and Jin-Woo Lee. Tunable and stable real-time trajectory planning for urban autonomous driving. In *2015 IEEE/RSJ IROS*, pages 250–256. IEEE, 2015.

[40] Shengbo Guo and Scott Sanner. Real-time multiattribute bayesian preference elicitation with pairwise comparison queries. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 289–296, 2010.

[41] Reymundo A Gutierrez, Vivian Chu, Andrea L Thomaz, and Scott Niekum. Incremental task modification via corrective demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1126–1133. IEEE, 2018.

[42] Dylan Hadfield-Menell, Smitha Milli, Pieter Abbeel, Stuart J Russell, and Anca Dragan. Inverse reward design. In *Advances in neural information processing systems*, pages 6765–6774, 2017.

[43] K Hauser. The Minimum Constraint Removal Problem with Three Robotics Applications. *The International Journal of Robotics Research*, 33(1):5–17, 2014.

[44] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

[45] Rachel Holladay, Shervin Javdani, Anca Dragan, and Siddhartha Srinivasa. Active comparison based learning incorporating user uncertainty and noise. In *RSS Workshop on Model Learning for Human-Robot Communication*, 2016.

[46] Ashesh Jain, Shikhar Sharma, Thorsten Joachims, and Ashutosh Saxena. Learning preferences for manipulation tasks from online coactive feedback. *International Journal of Robotics Research*, 34(10):1296–1313, 2015.

[47] Ashesh Jain, Shikhar Sharma, Thorsten Joachims, and Ashutosh Saxena. Learning preferences for manipulation tasks from online coactive feedback. *The International Journal of Robotics Research*, 34(10):1296–1313, 2015.

[48] Ashesh Jain, Brian Wojcik, Thorsten Joachims, and Ashutosh Saxena. Learning trajectory preferences for manipulators via iterative improvement. In *Advances in neural information processing systems*, pages 575–583, 2013.

[49] Kevin G Jamieson and Robert Nowak. Active ranking using pairwise comparisons. In *Advances in Neural Information Processing Systems*, pages 2240–2248, 2011.

[50] Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *IJRR*, 34(7):883–921, 2015.

[51] Hong Jun Jeon, Smitha Milli, and Anca D. Dragan. Reward-rational (implicit) choice: A unifying formalism for reward learning, 2020.

[52] Rudolf Emil Kalman. When is a linear control system optimal? In *Joint Automatic Control Conference*, pages 1–15, 1963.

[53] J. Karlsson, C. Vasile, J. Tumova, S. Karaman, and D. Rus. Multi-vehicle motion planning for social optimal mobility-on-demand. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7298–7305, 2018.

[54] Adam Kasperski and Pawel Zielinski. An approximation algorithm for interval data minmax regret combinatorial optimization problems. *Inf. Process. Lett.*, 97(5):177–180, 2006.

[55] W Bradley Knox, Peter Stone, and Cynthia Breazeal. Training a robot via human feedback: A case study. In *International Conference on Social Robotics*, pages 460–470. Springer, 2013.

[56] Jens Kober, J. Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics. *The International Journal of Robotics Research*, 32(11):1238–1274, 2015.

[57] Marina Kollmitz, Kaijen Hsiao, Johannes Gaa, and Wolfram Burgard. Time dependent planning on a layered social cost map for human-aware robot navigation. In *Proc. of the IEEE Eur. Conf. on Mobile Robotics (ECMR)*, 2015.

[58] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*. Springer Publishing Company, Inc., 4th edition, 2007.

[59] Morteza Lahijanian and M Kwiatkowska. Specification revision for markov decision processes with optimal trade-off. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 7411–7418, 2016.

[60] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006.

[61] L Adrián León, Ana C Tenorio, and Eduardo F Morales. Human interaction for effective reinforcement learning. In *European Conf. Mach. Learning and Principles and Practice of Knowledge Discovery in Databases (ECMLPKDD 2013)*, volume 3, 2013.

[62] Marek Libura. Sensitivity analysis for minimum hamiltonian path and traveling salesman problems. *Discrete Applied Mathematics*, 30(2):197 – 211, 1991.

[63] Maxim Likhachev and Dave Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *IJRR*, 28(8):933–945, 2009.

[64] Yun Lin, Shaogang Ren, Matthew Clevenger, and Yu Sun. Learning grasping force from demonstration. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1526–1531. IEEE, 2012.

[65] Manuel Lopes, Francisco Melo, and Luis Montesano. Active learning for reward estimation in inverse reinforcement learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 31–46. Springer, 2009.

[66] Dylan P Losey and Marcia K O'Malley. Including uncertainty when learning from human corrections. In *Conference on Robot Learning*, pages 123–132, 2018.

[67] Björn Lötjens, Michael Everett, and Jonathan P How. Safe reinforcement learning with model uncertainty estimates. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8662–8668. IEEE, 2019.

[68] Silvano Martello. Knapsack problems: algorithms and computer implementations. *Wiley-Interscience series in discrete mathematics and optimization*, 1990.

[69] Matthew McNaughton, Chris Urmson, John M Dolan, and Jin-Woo Lee. Motion planning for autonomous driving with a conformal spatiotemporal lattice. In *2011 IEEE International Conference on Robotics and Automation*, pages 4889–4895. IEEE, 2011.

[70] Matthew McNaughton, Chris Urmson, John M Dolan, and Jin-Woo Lee. Motion planning for autonomous driving with a conformal spatiotemporal lattice. In *IEEE International Conference on Robotics and Automation*, pages 4889–4895, 2011.

[71] Bernard Michini, Thomas J Walsh, Ali-Akbar Agha-Mohammadi, and Jonathan P How. Bayesian nonparametric reward learning from demonstration. *IEEE Transactions on Robotics*, 31(2):369–386, 2015.

[72] R. Montemanni and L.M. Gambardella. An exact algorithm for the robust shortest path problem with interval data. *Computers & Operations Research*, 31(10):1667 – 1680, 2004.

[73] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[74] Scott Niekum, Sachin Chitta, Andrew G Barto, Bhaskara Marthi, and Sarah Osentoski. Incremental semantically grounded learning from demonstration. In *Robotics: Science and Systems*, volume 9. Berlin, Germany, 2013.

[75] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, New York, 2nd edition, 2006.

[76] OTTO Motors. Agv vs. sdv: A comparison of automated material transport white paper, November 2016. https://www.ottomotors.com/agv-vs-sdv [Online; posted 07-November-2016].

[77] Malayandi Palan, Nicholas C Landolfi, Gleb Shevchuk, and Dorsa Sadigh. Learning reward functions by integrating human demonstrations and preferences. *arXiv preprint arXiv:1906.08928*, 2019.

[78] Alexandros Paraschos, Christian Daniel, Jan Peters, and Gerhard Neumann. Using probabilistic movement primitives in robotics. *Autonomous Robots*, 42(3):529–551, 2018.

[79] K. Park, M. Bell, I. Kaparias, and K. Bogenberger. Learning user preferences of route choice behaviour for adaptive route guidance. *IET Intelligent Transport Systems*, 1(2):159–166, June 2007.

[80] Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2817–2826. JMLR. org, 2017.

[81] Mihail Pivtoraiko and Alonzo Kelly. Generating near minimal spanning control sets for constrained motion planning in discrete state spaces. In *2005 IEEE/RSJ IROS*, pages 3231–3237. IEEE, 2005.

[82] Mihail Pivtoraiko and Alonzo Kelly. Kinodynamic motion planning with state lattice motion primitives. In *2011 IEEE/RSJ IROS*, pages 2172–2179. IEEE, 2011.

[83] 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, 2009.

[84] Mattia Racca and Ville Kyrki. Active robot learning for temporal task models. In *Proceedings of the 2018 ACM/IEEE International Conference on Human-Robot Interaction*, pages 123–131, 2018.

[85] Mattia Racca, Antti Oulasvirta, and Ville Kyrki. Teacher-aware active robot learning. In *2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 335–343. IEEE, 2019.

[86] Ramkumar Ramaswamy, James B. Orlin, and Nilopal Chakravarti. Sensitivity analysis for shortest path problems and maximum capacity path problems in undirected graphs. *Mathematical Programming*, 102(2):355–369, Mar 2005.

[87] Dorsa Sadigh, Anca D Dragan, Shankar Sastry, and Sanjit A Seshia. Active preference-based learning of reward functions. In *Robotics: Science and Systems (RSS)*, 2017.

[88] Ankit Shah, Pritish Kamath, Julie A Shah, and Shen Li. Bayesian inference of temporal task specifications from demonstrations. In *Advances in Neural Information Processing Systems*, pages 3804–3813, 2018.

[89] Meher T. Shaikh and Michael A. Goodrich. Design and evaluation of adverb palette: A gui for selecting tradeoffs in multi-objective optimization problems. In *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, pages 389–397, New York, NY, USA, 2017. ACM.

[90] Thomas B. Sheridan. Human–robot interaction. *Human Factors*, 58(4):525–532, 2016.

[91] W. D. Smart and L. Pack Kaelbling. Effective reinforcement learning for mobile robots. In *Proceedings 2002 IEEE International Conference on Robotics and Automation*, volume 4, pages 3404–3410 vol.4, May 2002.

[92] Thane Somers and Geoffrey A. Hollinger. Human–robot planning and learning for marine data collection. *Autonomous Robots*, 40(7):1123–1137, Oct 2016.

[93] Shashank Srinivas, Ramtin Kermani, Kangjin Kim, Yoshihiro Kobayashi, and Georgios Fainekos. A graphical language for ltl motion and mission planning. In *Robotics and Biomimetics (ROBIO), 2013 IEEE International Conference on*, pages 704–709. IEEE, 2013.

[94] A. Steinfeld. Interface lessons for fully and semi-autonomous mobile robots. In *2004 IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04*, volume 3, pages 2752–2757 Vol.3, April 2004.

[95] Peter Stone, Richard S Sutton, and Gregory Kuhlmann. Reinforcement learning for robocup soccer keepaway. *Adaptive Behavior*, 13(3):165–188, 2005.

[96] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[97] Robert Endre Tarjan. Sensitivity analysis of minimum spanning trees and shortest path trees. *Information Processing Letters*, 14(1):30 – 33, 1982.

[98] Simon Thompson, Takehiro Horiuchi, and Satoshi Kagami. A probabilistic model of human motion and navigation intent for mobile robot path planning. In *Autonomous Robots and Agents, 2009. ICARA 2009. 4th International Conference on*, pages 663–668. IEEE, 2009.

[99] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008.

[100] Leslie G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189 – 201, 1979.

[101] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[102] C. I. Vasile, J. Tumova, S. Karaman, C. Belta, and D. Rus. Minimum-violation scltl motion planning for mobility-on-demand. In *IEEE ICRA*, pages 1481–1488, May 2017.

[103] Valeria Villani, Fabio Pini, Francesco Leali, and Cristian Secchi. Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications. *Mechatronics*, 55:248 – 266, 2018.

[104] Nick Walker, Yuqian Jiang, Maya Cakmak, and Peter Stone. Desiderata for planning systems in general-purpose service robots. *arXiv preprint arXiv:1907.02300*, 2019.

[105] Larry Wasserman. *All of statistics: a concise course in statistical inference*. Springer Science & Business Media, 2013.

[106] C. Weinrich, M. Volkhardt, E. Einhorn, and H. M. Gross. Prediction of human collision avoidance behavior by lifelong learning for socially compliant robot navigation. In *2013 IEEE International Conference on Robotics and Automation*, pages 376–381, May 2013.

[107] Nils Wilde, Alexandru Blidaru, Stephen L Smith, and Dana Kulić. Improving user specifications for robot behavior through active preference learning: Framework and evaluation. *The International Journal of Robotics Research*, 39(6):651–667, 2020.

[108] Nils Wilde, Dana Kulić, and Stephen L. Smith. Learning user preferences in robot motion planning through interaction. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 619–626, May 2018.

[109] Nils Wilde, Dana Kulić, and Stephen L. Smith. Bayesian active learning for collaborative task specification using equivalence regions. *IEEE Robotics and Automation Letters*, 4(2):1691–1698, April 2019.

[110] Nils Wilde, Dana Kulić, and Stephen L. Smith. Active preference learning using maximum regret. In *2020 IEEE International Conference on Intelligent Robots and Systems (IROS)*, volume : To appear, October 2020.

[111] Nils Wilde, Dana Kulić, and Stephen L. Smith. Learning user preferences from corrections on state lattices. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, volume : To appear, May 2020.

[112] Aaron Wilson, Alan Fern, and Prasad Tadepalli. A bayesian approach for policy learning from trajectory preference queries. In *Advances in neural information processing systems*, pages 1133–1141, 2012.

[113] Holly A. Yanco, Adam Norton, Willard Ober, David Shane, Anna Skinner, and Jack Vice. Analysis of Human-robot Interaction at the DARPA Robotics Challenge Trials. *Journal of Field Robotics*, 32(3):420–444, May 2015.

[114] Gang Yu and Jian Yang. On the robust shortest path problem. *Computers & Operations Research*, 25(6):457 – 468, 1998.

[115] Jason Y Zhang and Anca D Dragan. Learning from extrapolated corrections. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7034–7040. IEEE, 2019.

[116] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

APPENDICES

Appendix A

Sensitivity of Shortest Path Problems

In this section we study the sensitivity of the shortest path problem on graphs. Broadly speaking, given is a weighted graph G, a fixed start and goal vertex on G and some path P between the start and goal. We call the set of all edge weights for which P is a shortest path between the start and goal the *sensitivity region* of P. This is closely related to the notion of *equivalence regions* that we introduced in Chapter 4 and further studied in Chapter 6. In fact, in Proposition 4 and its proof, we considered specifications that put a single constraint on every edge of the graph, effectively rendering equivalence regions to sensitivity regions. Thus, equivalence regions are a generalization of sensitivity regions. Proposition 4 stated that finding all equivalence regions is computationally intractable since the problem of finding all paths between a fixed start and goal on a graph can be reduced to it. We now show that finding a description of just one single equivalence region is also #P-hard.

Overview

We consider a weighted graph $G = (V, E, w)$ with n vertices and m edges and positive edge weights. The edge weights w do not necessarily satisfy the triangle inequality. We denote the set of all paths from a start vertex s to a goal vertex g in V with $\mathcal{P}^{sg}$.

Definition 8 (Sensitive region)**.** Let $G = (V, E, w)$ be a graph, s and g vertices in V and P an element of $\mathcal{P}^{sg}$ Given a set of allowable weights for all edges$Q \subseteq \mathbb{R}^m$, the *sensitive region* of P is then the subset of Q for which P is a shortest path from start to goal. In

other words, any other path $P^j \in \mathcal{P}^{sg}$ has an equal or higher cost than P for any weight $\boldsymbol{w}$ in the sensitive region $S(P)$. More formally we have:

$$S(P) = \{\boldsymbol{w} \in Q | \boldsymbol{\phi}\boldsymbol{w} \leq \boldsymbol{\phi}^j\boldsymbol{w},\ \forall j\} \tag{A.1}$$

Related Work The sensitivity of shortest path problems has been study for the case that only one edge weight changes at a time in[97, 86]. Similarly, sensitivity has been studied for the traveling saleman problem (TSP)[62]. Changes or uncertainty in edge weights are also considered in *robust shortest path problems*[114, 72].

To characterize the hardness of describing a sensitive region, we consider the complexity class of *counting problems*[100, 101, 26]. Generally, #P-hard counting problems are at least as hard as NP-hard decision problems. Typical counting problems include finding the number of solutions of optimization problems, such as the TSP tours or SAT solutions, but also finding the number of shortest paths is #P-hard, even though finding *a* shortest path can be done in polynomial time.

Characteristics of Sensitivity Regions

Generally, we assume that all weights are positive and finite; hence, $Q = [0, 1]^m$. We now further characterize sensitive regions and derive their representation. From the definition of paths follows that $\mathcal{P}^{sg}$ is of finite size. Initially, we introduce two lemmas.

Lemma 4 (Zero weights)**.** For each $P \in \mathcal{P}^{sg}$, the point $\boldsymbol{w} = \mathbf{0}$ lies in $S(P)$, as $c(P, \mathbf{0}) = 0$.

Lemma 5 (Convexity)**.** Sensitive regions are convex.

Proof. The set of weights for which a path P^i is optimal satisfies

$$\boldsymbol{\phi}^i\boldsymbol{w} \leq \boldsymbol{\phi}^j\boldsymbol{w}, \forall j \neq i,\ i, j = 1, 2, \ldots, |\mathcal{P}^{sg}|. \tag{A.2}$$

As $|\mathcal{P}^{sg}|$ is finite, there exists only a finite number of paths P^j. Each constraint in (A.2) defines a half-space in $\mathbb{R}^m_{\geq 0}$. By definition, the intersection of halfspaces form a convex set. □

For a more sparse notation we write $S(P^i)$ as S^i.The hyperplane separating two sensitive regions S^i and S^j is given by the set H^{ij} where P^i and P^j have equal cost, i.e., $H^{ij} = \{\boldsymbol{w} \in Q : (\boldsymbol{\phi}^i - \boldsymbol{\phi}^j)\boldsymbol{w} = 0\}$. We notice that $H^{ij} = S^i \cap S^j$. All weights $\boldsymbol{w} \in S^i$ then satisfy

$\boldsymbol{a}^{ij}\boldsymbol{w} \leq 0$ for all i, j where $\boldsymbol{a}^{ij} = \boldsymbol{\phi}^i - \boldsymbol{\phi}^j$. By definition, $\boldsymbol{a}^{ij}$ takes values in $\{-1, 0, 1\}^m$. Moreover, H^{ij} is a vector hyperplane, i.e., it passes through the origin.
Given any graph $G = (V, E, w)$ and a shortest path P, the sensitive region $S(P)$ can be described by a set of hyperplanes $\boldsymbol{a}^{ij}\boldsymbol{w} \leq 0$. It can be written as a polyhedron $\boldsymbol{A}\boldsymbol{w} \leq 0$. To further characterize the hyperplanes $\boldsymbol{a}^{ij}$ we use some basic properties of paths and introduce the following lemma.

Lemma 6 (Exclusive edges)**.** Consider a Graph $G = (V, E, w)$, a start and a goal vertex s and g and any two paths P^i and P^j from $\mathcal{P}^{sg}$. Then there exists at least one edge in P^i that is not in P^j.

Proof. Assume that all edges that are in P^i are also traversed by P^j. Then either $E^i = E^j$ and the paths are identical or P^j visits at least one vertex twice, which contradicts the definition of a path. Hence, the statement holds. □

From Lemma 6, we conclude that $\boldsymbol{a}^{ij}$ has at least one entry equal to 1 and one entry equal to -1. This property is crucial to the proof of Theorem 4. As we consider a closed space Q, the set S is a convex polytope and can be written in either a vertex or hyperplane representation.

Proposition 6 (Representation with hyperplanes)**.** Sensitive regions can be represented with at most as many hyperplanes as vertices.

Proof. We prove the statement by showing that number of vertices of S is always an exponential function of m while the number of hyperplanes is at most of the same size. Given a graph G and a path P^i, let $Q = [0, 1]^m$. For any hyperplane given by $\boldsymbol{a}^{ij}$, let r^{ij} be the number of 1 entries in $\boldsymbol{a}^{ij}$. Then $0 < r^{ij} \leq |E(P^i)|$ as the path P^j cannot traverse all edges that P^i traverses by the definition of a path. Then, the number of vertices of Q that satisfy $\boldsymbol{a}^{ij}\boldsymbol{w} \leq 0$ is at least $2^{m-r^{ij}}$ as we can pick either 0 or 1 for each w_k where $\boldsymbol{a}^{ij}_k \neq 1$. Given a set of hyperplanes $\{\boldsymbol{a}^{i0}, \boldsymbol{a}^{i1} \dots\}$ at least $2^{m-|E(P^i)|}$ vertices satisfy all inequality constraints.
The number of vertices is exponential in the size of m. Let $E'(P^i)$ be the set of edges P^i does not traverse implying $|E'(P^i)| = m - |E(P^i)|$. Each path P^j has to use at least one edge in $E'(P^i)$. For each subset of $E'(P^i)$ there exists at most one P^j that traverses it and no other edges in $E'(P^i)$. This implies that there exists at most $2^{|E'(P^i)|} = 2^{m-|E(P^i)|}$ paths P^j and therewith hyperplanes H^{ij}. Hence, the number of hyperplanes is at most the number of vertices. On the other hand, we can easily construct examples, where the number of hyperplanes is linear while the number vertices is exponential. We conclude that the hyperplane representation of S^i is always at least as efficient as the vertex representation.

□

Even though sensitive regions S can be represented by a set of $|\mathcal{P}^{sg}| - 1$ inequality constraints, there might exist graphs where fewer are necessary to uniquely define S, i.e., some rows of $\boldsymbol{A}$ are redundant with respect to others. Therefore, we introduce the notion of domination.

Definition 9 (Domination)**.** Let $\boldsymbol{A}$ be a $k \times m$-matrix and $\boldsymbol{b}$ be a vector in $\mathbb{R}^m$ where each element takes values in $\{-1, 0, 1\}$ and all rows of $\boldsymbol{A}$ as well as $\boldsymbol{b}$ have at least one entry equal to 1 and one entry equal to -1. We say that $\boldsymbol{A}$ dominates $\boldsymbol{b}$ if for all $\boldsymbol{w} \geq 0$ where $\boldsymbol{A}\boldsymbol{w} \leq 0$ is satisfied, $\boldsymbol{b}\boldsymbol{w} \leq 0$ also holds.

Definition 10 (Neighbours)**.** On a graph $G = (V, E, w)$ we call two sensitive regions S^i and S^j neighbours if their intersection forms a non-zero dimensional subspace, i.e., their intersection does not only consist of the origin. Then, the vector $\boldsymbol{a}^{ij}$ is not dominated by any collection of other vectors defining S^i, or S^j respectively.

Using the notion of domination we can now establish one of the main result of this work: Specifying how many hyperplanes are required to describe a sensitive regions.

Theorem 4 (Number of hyperplanes)**.** There exist at least one class of graphs where the number of hyperplanes that are necessary to define a sensitive region S is equal to the number of paths between the start and the goal vertex.

Proof. In Lemma 5 we have shown that S can be defined by the hyperplanes obtained from comparing a path to all other paths from s to g. Using the notion of domination, we construct an example where in fact all hyperplanes obtained from comparing a path to all other paths is necessary as no hyperplane is redundant with respect to the others.

Choose any graph G where the direct edge between s and g is in the edge set. We then fix P^0 as the path only using this edge. We observe that every other path P^j has a disjoint edge set. From Lemma 6 we conclude that any third path P^u then must traverse at least one edge that neither P^0 nor P^j traverse. We define $\boldsymbol{A}$ to be the matrix containing all comparisons of P^0 and P^j for all paths P^j on G, where $j > 0$. So each row is of the form $\boldsymbol{a}^{0j}$.

We pick some arbitrary r and denote the matrix $\boldsymbol{A}$ with the r-th row removed as $\boldsymbol{A}^r$. We now show that $\boldsymbol{A}^r$ does not dominate the removed row $\boldsymbol{a}^{0r}$ by constructing a $\bar{\boldsymbol{w}} \geq 0$

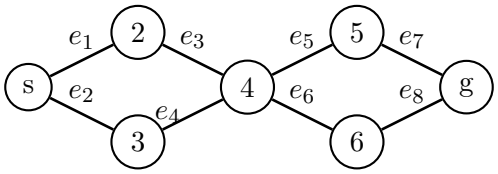


Figure A.1: Example of a graph where not all paths are required to characterize a sensitive region S

such that $\boldsymbol{A}^r\bar{\boldsymbol{w}} \leq 0$ but $\boldsymbol{a}^{0r}\bar{\boldsymbol{w}} > 0$. Let k be the smallest integer where $a_k^{0r} = 1$. Then

$$\bar{w}_i = \begin{cases} 1, \text{ if } i = k \\ 1, \text{ if } a_i^{0r} = 0, \text{and } e_i \notin P^0 \\ 0, \text{ otherwise.} \end{cases} \tag{A.3}$$

By construction, $\boldsymbol{a}^{0r}\bar{\boldsymbol{w}} > 0$ always holds. Moreover, for any $j \neq r$ there exists only one i where $a_i^{0j}\bar{w}_i$ can become 1, namely for $i = k$. On the other hand, each vector $\boldsymbol{a}^{0j}$ has at least one -1 entry for some i where $a_i^{0r} = 0$ and e_i is not in P^0. This is due to the fact, that every path P^j must traverse at least one edge that P^0 and P^r did not traverse. As we picked $\bar{w}_i$ to be 1 in that case, there exists at least one i such that $a_i^{0j}\bar{w}_i = -1$. Hence, the sum $\sum_i a_i^{0j}\bar{w}_i$ is guaranteed to be non-positive, so $\boldsymbol{A}^r\bar{\boldsymbol{w}} \leq 0$ holds. We conclude that $\boldsymbol{A}^r$ does not dominate $\boldsymbol{a}^{0r}$ and therefore all comparisons $\boldsymbol{a}^{\mathbf{0}\boldsymbol{j}}$ are necessary to describe S. □

From a different point of view Theorem 4 can be summarized as follows: In some graphs all paths from start to goal have to be computed in order to characterize a sensitive region. However, this does not apply to any graph in general.

Example 5 (Simple sensitive regions)**.** Let l be the number of paths from a start to a goal on a given graph. There exists at least one graph, where fewer than $l-1$ hyperplanes are necessary to characterize a sensitive region. We consider the example graph from Figure A.1. All possible $s-g$ paths then are $P^0 = \{s, 2, 4, 5, g\}$, $P^1 = \{s, 2, 4, 6, g\}$, $P^2 = \{s, 3, 4, 5, g\}$ and $P^3 = \{s, 3, 4, 6, g\}$. Then, given the comparisons $\boldsymbol{a}^{01}$ and $\boldsymbol{a}^{02}$, the third one $\boldsymbol{a}^{03}$ is redundant, which can be verified numerically.

Summary In summary, we have shown that 1) sensitivity regions can be described most efficiently with hyperplanes, and 2) in a worst case, the number of hyperplanes equals $|\mathcal{P}^{sg}| - 1$ – the number of paths from s to g, minus one for the given path P.

Complexity of finding Sensitive Regions

In order to characterize the complexity of finding a sensitive region of a shortest path we introduce a precise problem formulation and a counting version of it.

Problem 10 (Find sensitive region)**.** Given a graph $G = (V, E, w)$, a start and a goal vertex s and g and a shortest path P from s to g, find $S(P)$, i.e., find all weights $\boldsymbol{w}$ in Q where P still is a shortest path.

From the previous analysis of sensitive regions we know that S^i is characterized by the hyperplanes obtained from comparing P^i with other paths P^j and the bounds given by the hypercube $[0, 1]^m$. We formulate a counting problem for finding the number of hyperplanes we need in order to characterize $S(P)$.

Problem 11 (Find neighbours)**.** Given a graph $G = (V, E, w)$, a start and a goal vertex s and g and a shortest path P from s to g, find the number of hyperplanes that characterize $S(P)$.

Reduction from S-T-Paths

A related problem is S-T-Paths. Given a graph and a start and goal vertex, it returns the number of paths connecting a start and a goal vertex. The problem is known to be $\#P$ complete [101]. We now reduce Problem S-T-Paths to Problem 11 to show hardness of finding $S(P)$.

Theorem 5 (Hardness)**.** The problem of finding the number of hyperplanes that define a sensitive region is $\#P$-complete.

Proof. We consider an instance of S-T Paths with a graph $G = (V, E)$ and a start and goal vertex s and g. We generate a weighted graph $G' = (V', E', \boldsymbol{w}')$ where $V' = V$. Let e_{sg} be the direct edge between s and g. The set E' is then defined as $E' = E \cup e_{sg}$. We choose $\boldsymbol{w}'$ such that the corresponding shortest path P^0 only traverses e_{sg}.

We then solve Problem 11 given G', s, g and P^0. We denote the output with k, which is the number of vector hyperplanes that describe $S(P^0)$. From Theorem 4 we know that S is described by the comparisons of P^0 to all other paths P^j on G'. Hence, k equals the number of S-T-paths on G' minus one. If the edge e_{sg} is in the original edge set E, the solution to S-T Paths on G then is k and is $k + 1$ otherwise.

Furthermore, we observe that the problem of finding the number of hyperplanes that define a sensitive region lies in $\#P$. According to [100] "$\#P$ is the class of functions that can be computed by counting TMs [Turing Machines] of polynomial time complexity." We can pose a decision version of our problem of the form *does* $S(P)$ *have at least/most* k *neighbours?* This problem can be solved by a nondeterministic touring machine and therefore lies in NP. We conclude that the counting problem lies in $\#P$. □

As Problem 11 is the counting version of Problem 10, Problem 10 is also $\#P$ complete. Together with Proposition 6, stating that S is represented most efficiently with hyperplanes, we conclude that finding S is also $\#P$ complete.

Summary With Theorem 5 we have shown that finding a description of a sensitive region is computationally intractable. This implies that finding an equivalence region for the preference learning problem is also intractable, since equivalence regions are a generalisation of sensitivity regions.

Appendix B

Hardness of Maximum Regret Computation for Deterministic Motion Planners

In this section we provide further insight into the *Maximum Regret under Constraints Problem (MRuCP)*. We analyse the hardness of solving this problem when paths are computed with a discrete motion planner, i.e., the set of all possible paths is of finite size. In that case the motion planner can be represented with a graph and the planning problem becomes one of finding a minimum cost path on a graph. We consider a graph where paths are described by features that indicate which edges are visited by the path: $\phi_i^P = 1$ if $e_i \in P$ and 0 otherwise. We notice that this is a generalization of any discrete planning problem, where features are (1) a functions of sets of state-action pairs, and (2) theses feature functions are modular. This allows us to define the discrete version of MRuCP:

Problem 12 (Discrete Maximum Regret under Constraints Problem - DMRuCP)**.** Given a graph $G = (V, E, w)$, a start and goal vertex s and g, a set of feasible weights $\mathcal{F} = \{\boldsymbol{w} \in \mathbb{R}_{\geq 0}^{|E|} \mid l_i \leq w_i \leq u_i \text{ and } \boldsymbol{A}\boldsymbol{w} \leq \boldsymbol{b}\}$ and some weight $\boldsymbol{w}^P \in \mathcal{F}$, find the weight $\boldsymbol{w}^Q \in \mathcal{F}$ that maximizes the regret $r(\boldsymbol{w}^P, \boldsymbol{w}^Q)$.

First, we observe that if we remove the constraints, i.e., $\boldsymbol{A} = \boldsymbol{0}$ and $\boldsymbol{b} = \boldsymbol{0}$, then finding $\boldsymbol{w}^Q$ is trivial [54]: Let P be the optimal path for $\boldsymbol{w}^P$, then $w_i^Q = l_i$ if $e_i \in P$ and u_i otherwise. That is, the weights of all edges used by the path P are at the upper bound and all other weights are at the lower bound. We now characterize the computational complexity of DMRuCP.

Theorem 6 (Hardness of DMRuCP)**.** The Discrete Maximum Regret under Constraints Problem (DMRuCP) is NP-complete.

Proof. We prove hardness by reducing the *Subset-Sum Problem* (SSP) [68] to DMRuC. SSP is a special case of the *1-0 Knapsack problem*, where the value v_i of each item equals its weight w_i. Consider the following SSP instance:

$$\begin{aligned} &\max_{\boldsymbol{x}} \sum_{i}^{n} \bar{w}_i x_i \\ &s.t. \sum_{i}^{n} \bar{w}_i x_i \leq W, \\ &\quad x_i \in \{0,1\} \forall i = 1,2,\ldots,n. \end{aligned} \tag{B.1}$$

Thereby, all w_i and W are assumed to be integers [68]. We now construct a graph G as illustrated in Figure B.1 where we choose V_0 as the start and V_n as the goal vertex. In the graph, all edges going from V_{i-1} to V_i have a cost of zero. For edges e_i we define the lower bound of the weight w_i is $l_i = -\bar{w}_i$, and the respective upper bounds $u_i = 0$ for all $i = 1,2,\ldots,n$. The bounds for the direct edge e_{n+1} are $l_{n+1} = u_{n+1} = -1/2$. For the edges e' the weights w'_i are unbounded, but we define an equality constrain $w'_i = M(w_i + \bar{w}_i)$. We set $M = 2n \cdot \bar{w}^{\max}$ with $\bar{w}^{\max} = \max_i\{\bar{w}_i\}$. Moreover, we complete the DMRuC instance by adding a budget constraints on the weights: $-(w_1 + w_2 + \cdots + w_n) \leq W$. Finally, we choose $\boldsymbol{w}^P$ such that $w_i^P = 0$ for all $i = 1,2,\ldots,n$ and $w_{n+1}^P = -1/2$: thus, the path P uses the direct edge from V_1 to V_n.

Let $\boldsymbol{w}^Q$ be an optimal solution of the DMRuC instance and Q be the shortest path for $\boldsymbol{w}^Q$. We observe that Q must be distinct from P and thus traverse all vertices $V_0, V_1, V_2, \ldots V_{n-1}$. This implies that Q is the path with the minimal cost among all paths that (1) are optimal for some feasible weight and (2) traverse all vertices, i.e., are not equal to P. Thus, finding the maximum regret path Q simplifies to finding the path with minimal cost other than P.

Claim 1. If Q visits V'_i then $w_i^Q = -\bar{w}_i$.

Proof. First, let c'_i be the cost for going from V_{i-1} to V_i via V'_i. We observe that a shortest

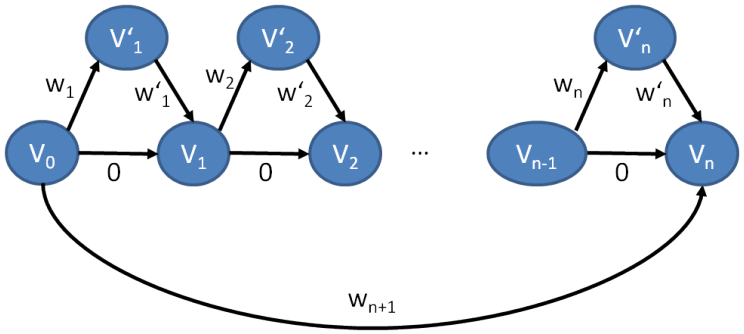


Figure B.1: Graph constructed from SSP instance. Visiting vertex V_i' corresponds to putting the i-th item into the knapsack. The input path P of the DMRuC instance takes the direct edge from V_0 to V_n.

path only traverses V_i' if $c_i' \leq 0$. Thus,

$$\begin{aligned} c_i' &\leq 0 \\ w_i^Q + M(w_i^Q + \bar{w}_i) &\leq 0 \\ w_i^Q &\leq -\bar{w}_i \frac{M}{1+M}. \end{aligned} \tag{B.2}$$

We call $-\bar{w}_i {}^{M}/_{1+M}$ the threshold τ_i. We observe that the difference between $-\bar{w}_i$ and the threshold τ_i is always less than 1, as $M = 2n \cdot \bar{w}^{\max}$. Further,

$$\sum_{i=1}^{n} -\bar{w}_i - \tau_i < \frac{M}{1+M}. \tag{B.3}$$

Notice that $\frac{M}{1+M} \leq -\min_i \tau_i$ as the smallest weight $\bar{w}_i = 1$. Thus, the sum of all differences between $\bar{w}_i$ and the thresholds τ_i is less then the minimal threshold.

For the solution $\boldsymbol{w}^Q$ let I be the set of indices $\{i | -\bar{w}_i \geq w_i^Q \geq \tau_i\}$, i.e., for all $i \in I$ the path Q visits V_i' as the cost is no more than the threshold, as described in equation (B.2). Following equation (B.3), there does not exist a set $I' \subset I$ where $|I'| = |I| - 1$ such that $\sum_{i \in I'}(-\bar{w}_i - \tau_i) \leq \tau_j$ where j is in I but not in I'. That is, setting any collection of w_i^Q to τ_i instead of $w_i^Q = -\bar{w}_i$ does not allow for Q to visit an additional vertex V_j'. Thus, when choosing some $w_i^Q > -\bar{w}_i$, the solution has a slack of $w_i^Q + \bar{w}_i$. As the slack cannot be used to visit and additional V_j', and $w_i^Q = -\bar{w}_i$ leads to a smaller cost of Q than $w_i^Q > -\bar{w}_i$, an optimal solution always has $w_i^Q = -\bar{w}_i$ if $w_i^Q \leq \tau_i$, i.e., if Q visits V_i'. □

We can transform the DMRCuP solution to a solution of the SSP: The decision variable $x_i = 1$ if Q visits V_i' and 0 otherwise. Thereby, Claim 1 ensures that $c(Q) = -\sum_i \bar{w}_i x_i$, concluding the reduction. Finally, we show membership in NP. The decision version of DRMuCP takes the same input, but additionally some number $\tau \in \mathbb{R}_{\geq 0}$. The problem then is deciding if there exists some $\boldsymbol{w}^Q \in \mathcal{F}$ such that $r(\boldsymbol{w}^P, \boldsymbol{w}^Q) >= \tau$. Given some candidate solution $\boldsymbol{w}^Q$ the regret $r(\boldsymbol{w}^P, \boldsymbol{w}^Q)$ can be computed in polynomial time by once computing the shortest paths P and Q; hence the solution can be verified in polynomial time establishing membership in NP and NP-completeness of DRMuCP.

□