Learning User Preferences in Robot Motion Planning through Interaction

Nils Wilde Dana Kulić Stephen L. Smith

***Abstract*— In this paper we develop an approach for learning user preferences for complex task specifications through human-robot interaction. We consider the problem of planning robot motion in a known environment, but where a user has specified additional spatial and temporal constraints on allowable robot motions. To illustrate the impact of the user's constraints on performance, we iteratively present users with alternative solutions on an interface. The user provides a ranking of alternate paths, and from this we learn about the importance of different constraints. This allows for an accessible method for specifying complex robot tasks. We present an algorithm that iteratively builds a set of constraints on the relative importance of each user constraint, and prove that with sufficient interaction, the algorithm determines a user-optimal path. We demonstrate the practical performance by simulating realistic material transport scenarios in industrial facilities.**

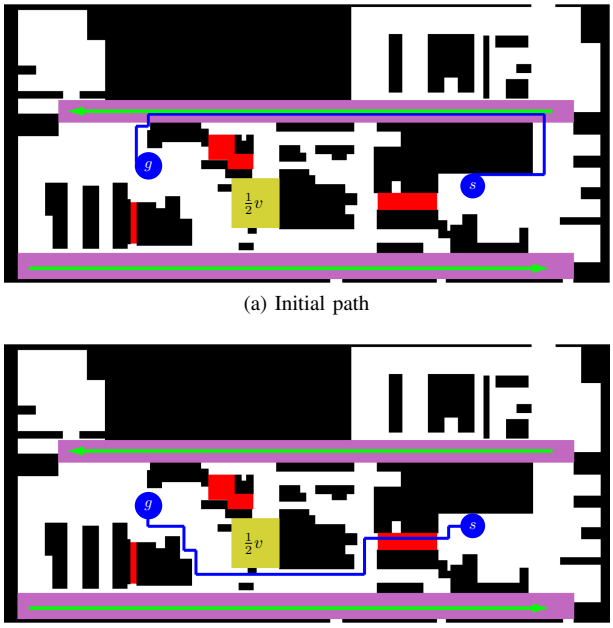
I. Introduction

While autonomous robots are finding increasingly widespread application, specifying robot tasks usually requires a high level of expertise. To enable a broader range of users to direct autonomous robots, human robot interfaces that allow non-expert users to set up complex task specifications [1], [2], [3] are required.

In many applications of autonomous mobile robots, it is desirable to restrict or specify the robot's behaviour due to safety reasons or to adjust to existing work flows [4]. For instance, in a warehouse or manufacturing environment a facility operator might define regions a robot should avoid or road rules. This can be accomplished by tagging the environment on a graphical user interface in order to define constraints like areas of avoidance, one way roads or speed limits. However, it might not be obvious to the user what impact each constraint has on the performance of the robot task. They might be willing to accept the robot violating less important constraints that are not mandatory for safety reasons if it is sufficiently beneficial. For instance, Figure 1 shows an industrial environment with several user defined constraints. In (a) the robot path respects all constraints, while in (b) the user agreed to the robot traversing (i.e., violating) one constraint as this enables significant reduction in the time to travel from start to goal. Instead of asking the user to specify costs for each constraint, we propose a procedure where we iteratively present the user with alternative solutions in order to learn their preferences.

This research is partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and Otto Motors.

The authors are with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo ON, N2L 3G1 Canada (dana.kulic@uwaterloo.ca; nwilde@uwaterloo.ca; stephen.smith@uwaterloo.ca)



(a) Initial path

(b) Path after learning user weights

Fig. 1. Example environment (white) with obstacles (black) and user defined constraints. There are two one way roads, drawn in purple with an arrow indicating the direction, a speed limit zone drawn in yellow where only half the maximum speed is allowed and several areas of avoidance, drawn in red. Blue indicates the shortest path from s to g. We compare (a) the initial path respecting all user constraints with a completion time of 62s and (b) the optimal path after learning user preferences that traverses an area of avoidance with a completion time of 46s.

In this work we consider a known environment where the robot must traverse from start to goal locations. We propose a user-on-the-loop algorithm: Starting with an initial solution, we iteratively generate alternative paths and request user feedback. Thereby, the user is "on-the-loop", i.e., does not need to constantly provide feedback as the best solution so far can already be executed. We assume that the user has some hidden cost for each constraint which expresses for what time benefit they will accept a violation. For known user costs the shortest path can easily be found using algorithms like Dijkstra's or A* [5]. By presenting alternative solutions to the user and requesting feedback, we iteratively learn in what range the hidden user costs lie. Eventually, we approach a unique solution for the shortest path problem and have then sufficiently learned the user preferences.

Related work: Learning user preferences has been studied in the area of route selection in navigation systems [6]. Although a user does not explicitly define constraints, they choose paths based on route-related attributes other than just shortest travel time. These preferences are learned by

observing the user's choices of alternative route options.

Our problem can also be understood as planning in partially known environments [5], [7]. While we have complete knowledge about the physical environment, the user costs are unknown. In our case, the costs are updated based on user feedback instead of observations.

Constraint revision for temporal logic specification is discussed in [8] and [9]. Thereby, the user provides a task specification in linear temporal logic (LTL), which is then subject to revision. Strictly following the specification might be infeasible [9], or result in a low probability of success [8]. The problem is to find a minimal revision, i.e., satisfy the original specification as far as possible. This is closely related to our problem: The initial specification does not violate any user constraints, but may result in long execution times. However, as the costs are unknown, we need to learn what revisions (i.e., violations) are acceptable to the user via human-robot-interaction.

In [10] interactive task learning is introduced as a new area of research, emphasizing the increasing demand for specifying robot tasks in an intuitive, interactive manner. Closely related to our work, [11] focuses on a multi-objective shortest path problem. A graphical user interface is designed to capture human intent and then explore trade-offs between different objectives. In our work we trade-off task performance and violation of the user specification. Nonetheless, instead of prompting the user with an attributes palette we investigate how user preferences can be learned from the feedback to alternative solutions.

Contributions: The main contributions of this paper are as follows: First, we introduce a problem formulation that combines the shortest path problem with user defined constraints. We define the cost of the constraints to be the minimal time benefit for which a user would accept a violation. Based on this we introduce a linear, deterministic user model that describes how the user would choose between alternative paths. Using this model, we propose a complete algorithm to learn user preferences through user interaction. Finally, we demonstrate that the algorithm has a good practical performance in a simulated material transport scenario.

II. Problem Formulation

A. Preliminaries

Following [12], 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 $\Psi : E \to \{(v, w) \in V \times V : v \neq w\}$ associates each edge with an ordered pair of vertices. Two edges are called parallel if they connect the same ordered pair of vertices.

In a weighted graph a real value function associates weights to each edge of the graph: $c : E(G) \to \mathbb{R}$. 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 $v_1, e_1, v_1, e_2, \ldots, e_k, v_{k+1}$ where $e_1, e_2, \ldots e_k$ are distinct. A path $P_{v_1, v_{k+1}}$ 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 $v_1, e_1, v_1, e_2, \ldots, e_k, v_{k+1}$ 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 $c_1(P) = \sum c_1(e),\ e \in P,\ c_2(P) = \sum c_2(e),\ e \in P$.

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 a matrix ($\boldsymbol{A}$).

B. Problem statement

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 has 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:

$$w(e) = \sum_{\gamma_i \in \Gamma | e \in E_i} w_i \tag{1}$$

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:

$$\min_P \sum_{e \in P} w(e) + t(e) \tag{2}$$

We state this objective without a scaling factor as w is in units of time. That is, w captures the increase in completion time that the user is willing to allow in order to satisfy the constraint. Even though the values for w_i are unknown, we can pick an estimate for all w_i. Then, the graph collapses to

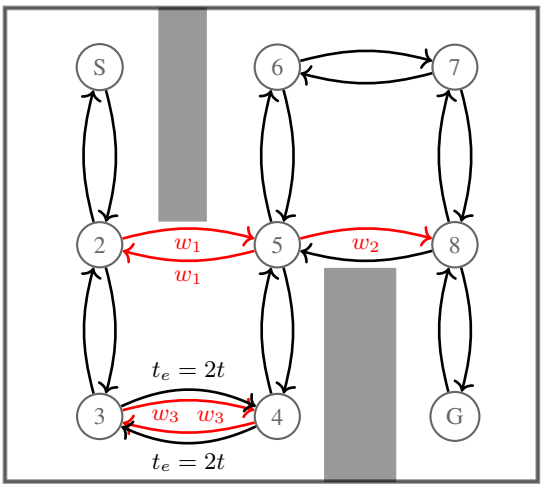


Fig. 2. Example scenario. The traversal times t_e between all edges are t, with exception of the outer pair between vertex 3 and 4. w_1, w_2, w_3 indicate the cost associated with the constrained edges, shown in red.

a singly-weighted directed graph, and the shortest path can be efficiently computed with Dijkstra's algorithm or A*. The goal is to find estimates for all w_i such that the corresponding path is optimal according to objective (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 (1). Figure 2 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 (2). To do so, we need to gain information about the hidden user preferences w_i.

III. 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.

A. User constraints and interaction

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 $\boldsymbol{w} = \begin{bmatrix} w_1 & w_2 & \cdots & w_n \end{bmatrix}^T$. Our estimate of the weight vector is denoted by $\hat{\boldsymbol{w}}$.

Definition 1 (Constraint violation)**.** Consider a path P on G and a user constraint γ_i. We say that P violates γ_i if and only if there exists an edge in P that is in E_i.

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 (2) for some estimate $\hat{\boldsymbol{w}}^j$, 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 estimated cost of a path for a weight $\hat{\boldsymbol{w}}^j$ then is

$$\hat{C}(P^j) = \boldsymbol{\phi}(P^j)\hat{\boldsymbol{w}}^j + t(P^j) = \boldsymbol{\phi}^j\hat{\boldsymbol{w}}^j + t^j \tag{3}$$

Now, we 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.

Finally, we define the true user cost C. Let P^j be some arbitrary path. If the true user weights $\boldsymbol{w}$ were known, we could directly compute the true user cost of P^j as follows:

$$C(P^j) = \boldsymbol{\phi}^j\boldsymbol{w} + t^j. \tag{4}$$

B. Equivalence region

Shortest paths are sensitive to the edge weights on the graph [13]. As different values for $\hat{\boldsymbol{w}}$ do not automatically lead to distinct solutions, we introduce *equivalence regions*.

Definition 2 (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 $\boldsymbol{w}^i$: $\Omega(\boldsymbol{w}^i) = \{\boldsymbol{w}^j \in \mathbb{R}^n_{\geq 0} | \boldsymbol{w}^j \text{ is } \textit{equivalent} \text{ to } \boldsymbol{w}^i\}$.

Note that equivalence is also applicable to estimated weights. Let $\hat{\boldsymbol{w}}^i$ and $\hat{\boldsymbol{w}}^j$ be equivalent. Then $C(P^i) = C(P^j)$ and $\hat{C}(P^i) = \hat{C}(P^j)$, however, the paths themselves are not necessarily the same. Moreover, a weight $\hat{\boldsymbol{w}}$ lies in multiple equivalence regions when the shortest path given $\hat{\boldsymbol{w}}$ is not unique. Equivalence regions are closely related to sensitivity of the shortest path to changes in the weights of the graph. According to [14], 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.

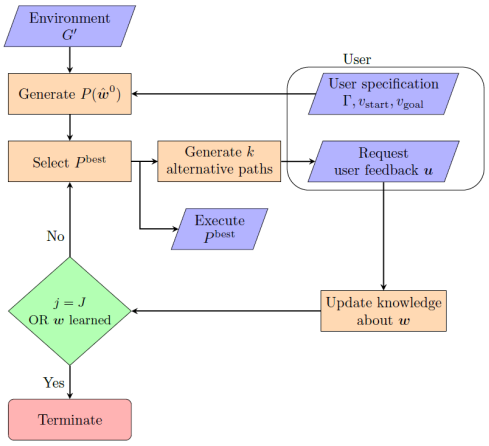
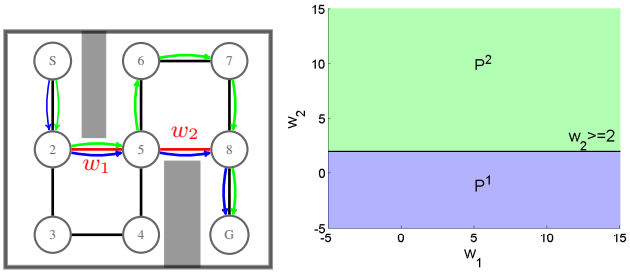


Fig. 3. Flowchart of the learning process. j denotes the single iteration, J is the maximum number of user interactions.



(a) Graph with different paths (b) Feasible space.

Fig. 4. (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.

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

$$\sum_{e \in P^i} w(e) \leq \sum_{e \in P^j} w(e), \ \forall i \neq j, \ i, j = 1, 2, \dots, l. \tag{5}$$

All constraints in (5) define a half-space in $\mathbb{R}^m_{\geq 0}$. By definition, the intersection of halfspaces form a convex set [13]. *Equivalence regions* describe a special case where only some edge costs can vary, thus are also convex. □

C. Learning user preferences

We now present the procedure for learning user preferences, illustrated in Figure 3. Assuming that there exists a path from start to goal that does not violate any user constraints, 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 $\boldsymbol{w}$ is sufficiently learned. 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 $\hat{\boldsymbol{w}}^i$ and $\hat{\boldsymbol{w}}^j$. If $C(P^i) > C(P^j)$ the user always provides a feedback where $u_i > u_j$. In this case, we assume that the user prefers P^j by at least some small $\epsilon > 0$. If $C(P^i) = C(P^j)$ the user feedback is $u_i = u_j$. Using the definition of C in Equation (4) we can derive the following constraints on the true user weights:

$$\begin{aligned} &\text{if } u_j < u_i \text{ then } \left(\boldsymbol{\phi}^j - \boldsymbol{\phi}^i\right) \boldsymbol{w} \leq t^i - t^j - \epsilon, \\ &\text{if } u_j = u_i \text{ then } \left(\boldsymbol{\phi}^j - \boldsymbol{\phi}^i\right) \boldsymbol{w} \leq t^i - t^j, \text{ and} \\ &\qquad\qquad \left(\boldsymbol{\phi}^i - \boldsymbol{\phi}^j\right) \boldsymbol{w} \leq t^j - t^i. \end{aligned} \tag{6}$$

From each pair (u_i, u_j) we learn that the user weight lies either on one side of the the hyperplane described by equation (6) if $u_i \neq u_j$, or on the hyperplane if $u_i = u_j$.

Definition 3 (Feasible space)**.** Given a user feedback $\boldsymbol{u}$ of length k, we can derive $k-1$ non-redundant constraints on the user weights $\boldsymbol{w}$ from expression (6). The feasible space 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 $\boldsymbol{A}\boldsymbol{w} \leq \boldsymbol{b}$ [13].

The initial feasible space 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 $\boldsymbol{A}\boldsymbol{w} \leq \boldsymbol{b}$, which define the feasible space, given the prior is linear.

Example 2 (Learning constraints of the feasible space)**.** 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. 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 space then is defined by $\begin{bmatrix} 0 & -1 \end{bmatrix} \begin{bmatrix} w_1 & w_2 \end{bmatrix}^T \leq \begin{bmatrix} -2 \end{bmatrix}$.

D. Learning Algorithm

Notice that the user ranks paths of equivalent weights equally. Thus, our goal is to find a weight $\hat{\boldsymbol{w}}^{\text{best}}$ that lies in the equivalence region of the true user preference $\boldsymbol{w}$. Based on the flowchart in Figure 3, we introduce Algorithm 1.

In line 1 of the algorithm we initialize the feasible space with some finite upper bounds $\boldsymbol{w}^{\max}$. We initialize $\hat{\boldsymbol{w}}^{\text{best}} = \boldsymbol{w}^{\max}$ and find the corresponding path P^{best}, which does not

Algorithm 1: Learning user weights

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

1 Initialize $\boldsymbol{A}$, $\boldsymbol{b}$, $\hat{\boldsymbol{w}}^{\text{best}}$, P^{best} and $\mathcal{W} = \emptyset$
2 **for** $j = 1$ *to* J **do**
3 $\mathcal{W}_{\text{new}} \leftarrow \pi(\boldsymbol{A}, \boldsymbol{b}, k, \mathcal{W}, \boldsymbol{w}^{\text{best}})$
4 **if** $\mathcal{W}_{\text{new}} = \emptyset$ **then**
5 **return** $\hat{\boldsymbol{w}}^{\text{best}}$
6 Compute paths $P^1, \dots, P^k$ for all $\hat{\boldsymbol{w}} \in \mathcal{W}_{\text{new}}$
7 Get user feedback $\boldsymbol{u}^j$ for paths $P^{\text{best}}, P^1, \dots, P^k$
8 Update $\boldsymbol{A}$ and $\boldsymbol{b}$ based on $\boldsymbol{u}^j$ (Definition 3)
9 Choose $\hat{\boldsymbol{w}}^{\text{best}}$ as the element from $\hat{\boldsymbol{w}}^{\text{best}} \cup \mathcal{W}_{\text{new}}$ with the best user feedback, $P^{\text{best}} = P(\hat{\boldsymbol{w}}^{\text{best}})$
10 $\mathcal{W} = \mathcal{W} \cup \mathcal{W}_{\text{new}}$
11 **return** $\hat{\boldsymbol{w}}^{\text{best}}$

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 $P(\boldsymbol{w}^{\infty})$ where $w_i^{\infty} = \infty$ for all $i = 1, \dots, n$. Then choose $w^{\max} = t(P(\boldsymbol{w}^{\infty}))$ and $\boldsymbol{w}^{\max} = w^{\max}\mathbf{1}^T$.

Proof. Let P' be a path that violates at least one constraint. According to equation (3), we have $\hat{C}(P') \geq \boldsymbol{w}^{\max} + t(P')$. As $t > 0$ for all edges, P' has a higher cost than $P^{\max}$. □

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

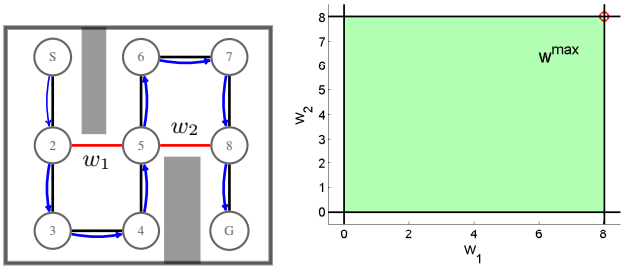
Definition 4 (Admissible policy.)**.** A policy is a function π that maps from $(\boldsymbol{A}, \boldsymbol{b}, k, \mathcal{W}, \hat{\boldsymbol{w}}^{\text{best}})$ to a set of k weights $\mathcal{W}^{\text{new}}$. A policy is *admissible* if each $\hat{\boldsymbol{w}} \in \mathcal{W}^{\text{new}}$ satisfies:

(i) $\boldsymbol{A}\hat{\boldsymbol{w}} \leq \boldsymbol{b}$, and
(ii) $\hat{\boldsymbol{w}}$ is not equivalent to any previous weight in $\mathcal{W}$ or any of the other new weights in $\mathcal{W}_{\text{new}}$.

We discuss two alternative policies in Sections III-E and III-F. 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 space 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 $\hat{\boldsymbol{w}}^{\text{best}}$ is *equivalent* to $\boldsymbol{w}$ and the optimal solution is obtained.

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



(a) Graph with initial path P^0. (b) Feasible space

Fig. 5. (a) shows the initial Path P^0 on the graph. In (b) we see the initial feasible space with $w^{\max}$.

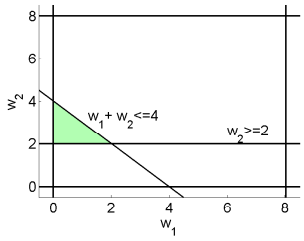


Fig. 6. Feasible space after three comparison. The solution is now uniquely determined.

Naively checking if all vertices of the feasible space are equivalent might be 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. We initially find the upper limit $w^{\max} = 8$ and obtain a path P^0 with $t(P^0) = 8$, shown in Figure 5. We then pick a new point of the feasible space, say $\hat{\boldsymbol{w}}^1 = \begin{bmatrix}0 & 0\end{bmatrix}^T$. 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 (6) 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 6. After three iterations all vertices of the feasible space are *equivalent* and result in path P^2.

Using *equivalence regions* we establish our main result:

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

Proof. Let $\hat{\boldsymbol{w}}^i$ and $\hat{\boldsymbol{w}}^j$ be non-equivalent weights. From the user model and inequalities (6) we derive the following relation when the user prefers P^i over P^j:

$$u_i < u_j \implies \boldsymbol{w} \notin \Omega(\hat{\boldsymbol{w}}_j). \tag{7}$$

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

$$u_i = u_j \implies \boldsymbol{w} \in \operatorname{span}\left(\Omega(\hat{\boldsymbol{w}}_i) \cap \Omega(\hat{\boldsymbol{w}}_j)\right). \tag{8}$$

We notice that the interior of $\Omega(\hat{\boldsymbol{w}}_i)$ and the interior $\Omega(\hat{\boldsymbol{w}}_j)$ 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 space.

As both, the number of equivalence regions and the number of user constraints, i.e., dimensions, are finite, the feasible space eventually either consists of only the *equivalence region* $\Omega(\boldsymbol{w})$, or $\boldsymbol{w}$ is uniquely determined. 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 IV we show that in practice the number of iterations is linear in the size of the input.

E. Vertex Search

We propose the policy $\pi_{\text{vertexSearch}}$ to find new vertices, shown in Algorithm 2. The *feasible space* 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 space, starting at the previously best solution $\hat{\boldsymbol{w}}^{\text{best}}$. 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 [13]. 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 [15], where the set of search directions consists of the vectors from the current weight to all vertices of the feasible space. Notice that for $i_{\max} \to \infty$ the policy is admissible. Furthermore, DFS explores a given graph in linear time.

Algorithm 2: $\pi_{\text{vertexSearch}}$, find new weights using DFS

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

1 Initialize set $\mathcal{W}_{\text{new}} = \emptyset$, `openList` $= \{\hat{\boldsymbol{w}}^{\text{best}}\}$ and maximum iterations $i_{\max}$
2 **for** $i = 0$ *to* $i_{\max}$ **do**
3 **if** $|\mathcal{W}_{\text{new}}| = k$ *or* `openList` *is empty* **then**
4 **return** $\mathcal{W}_{\text{new}}$
5 $\tilde{\boldsymbol{w}} =$ `openList`.pop()
6 **if** $\tilde{\boldsymbol{w}}$ is not equivalent to any $\hat{\boldsymbol{w}} \in \mathcal{W}_{\text{new}} \cup \mathcal{W}$ **then**
7 Add $\tilde{\boldsymbol{w}}$ to $\mathcal{W}_{\text{new}}$
8 **if** $\tilde{\boldsymbol{w}}$ is not labelled as discovered **then**
9 Label $\tilde{\boldsymbol{w}}$ as discovered
10 **for** *all* $\boldsymbol{w}' \in$ getAdjacentVertices($\tilde{\boldsymbol{w}}$) **do**
11 **if** $\boldsymbol{w}' \notin$ `openList` and $\tilde{\boldsymbol{w}}$ is not labelled as discovered **then**
12 `openList`.insert($\boldsymbol{w}'$)
13 **return** $\mathcal{W}_{\text{new}}$

F. Minimal Vertex

As $\pi_{\text{vertexSearch}}$ 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 weight space 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 $\pi_{\text{minSearch}}$ that always tries to minimize the weights within the current feasible space, 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, $\pi_{\text{vertexSearch}}$ is called (line 18).

IV. Evaluation

In the experiments we used layouts of real industrial facilities. We generate graphs G' by uniform grid-based sampling. For instance, in the environment from Figure 1, G' has 3646 vertices and 12456 edges. User feedback in each iteration is simulated using the true user cost C from equation (4). The quality of a path is evaluated using the relative error $err_{\text{rel}}(P)$, which is defined as follows:

$$err_{\text{rel}}(P) = \frac{C(P) - C(P^{\text{user}})}{C(P^{\text{user}})} \tag{9}$$

If $\hat{\boldsymbol{w}}$ is *equivalent* to $\boldsymbol{w}$, then $err_{\text{rel}} = 0$. Algorithm 1 does not have access to $C(\cdot)$. Therefore, alternative paths are 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 $\pi_{\text{vertexSearch}}$. 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

Algorithm 3: $\pi_{\text{minSearch}}$, find minimizing new weights

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

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

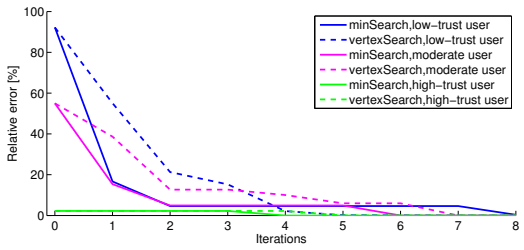
TABLE I
DIFFERENT TYPES OF USER FOR EXPERIMENT 1.

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

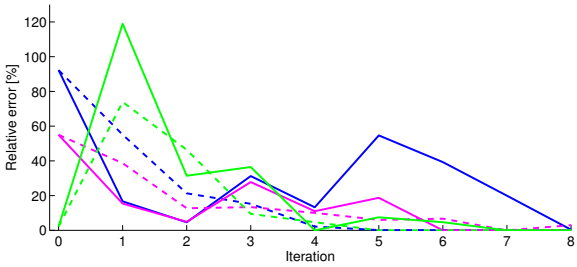
Table I. Problem instances are generated from a set of start-goal configurations. In Figure 7 we compare $\pi_{\text{minSearch}}$ and $\pi_{\text{vertexSearch}}$ for learning the preferences of all three users. We show the relative error of the best path so far P^{best} in subfigure (a) as well as the relative error of the new path P^j in subfigure (b), at each iteration j averaged over all trials.

In Figure 7 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 $\pi_{\text{minSearch}}$ finds relatively good solutions more quickly than $\pi_{\text{vertexSearch}}$. 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.

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



(a) Relative error the current best path P^{best}



(b) Relative error of each presented path P^j

Fig. 7. Comparison of the policies $\pi_{\text{minSearch}}$ and $\pi_{\text{vertexSearch}}$ for all three users, described in Table I

TABLE II
COMPARISON OF $\pi_{\text{vertexSearch}}$ AND $\pi_{\text{minSearch}}$.

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

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

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 (6) reduce the feasible space 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 II compares the best solutions each policy finds after j iterations. Even though in many cases both policies perform equally well, $\pi_{\text{minSearch}}$ still outperforms $\pi_{\text{vertexSearch}}$ between iteration two and eight.

Consequently, Table III shows the results of using $\pi_{\text{minSearch}}$. 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

TABLE III

RESULTS OF THE SECOND EXPERIMENT USING $\pi_{\text{minSearch}}$.

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	56.0	22.5

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.

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% 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.

V. DISCUSSION AND CONCLUSION

In this paper 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 classes 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 current work 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. A more detailed study of the equivalence regions is needed to characterize the problem; however, we believe that finding equivalence regions in general is computationally intractable. 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. To handle such cases, an extension to a probabilistic user model should be investigated. Third, in the evaluation we used either manually designed or randomly generated data that do not necessarily reflect real users. Future work should include user studies to generate more realistic data. When the user input is not simulated but comes from a human operator, additional performance measures can be captured. Furthermore, different heuristics could be studied in order to find a policy that is not only computationally efficient but also proposes alternative plans with consideration of the user.

REFERENCES

[1] T. B. Sheridan, "Humanrobot interaction," *Human Factors*, vol. 58, no. 4, pp. 525–532, 2016.

[2] L. A. León, A. C. Tenorio, and E. F. Morales, "Human interaction for effective reinforcement learning," in *European Conf. Mach. Learning and Principles and Practice of Knowledge Discovery in Databases*, 2013.

[3] B. Akgun, K. Subramanian, and A. L. Thomaz, "Novel interaction strategies for learning from teleoperation." in *AAAI Fall Symposium: Robots Learning Interactively from Human Teachers*, vol. 12, 2012, p. 07.

[4] M. Gombolay, A. Bair, C. Huang, and J. Shah, "Computational design of mixed-initiative humanrobot teaming that considers human factors: situational awareness, workload, and workflow preferences," *IJRR*, vol. 36, no. 5-7, pp. 597–617, 2017.

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

[6] K. Park, M. Bell, I. Kaparias, and K. Bogenberger, "Learning user preferences of route choice behaviour for adaptive route guidance," *IET Intelligent Transport Systems*, vol. 1, no. 2, pp. 159–166, June 2007.

[7] S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 354–363, June 2005.

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

[9] C. I. Vasile, J. Tumova, S. Karaman, C. Belta, and D. Rus, "Minimum-violation scltl motion planning for mobility-on-demand," in *IEEE ICRA*, May 2017, pp. 1481–1488.

[10] J. E. Laird, K. Gluck, J. Anderson, K. D. Forbus, O. C. Jenkins, C. Lebiere, D. Salvucci, M. Scheutz, A. Thomaz, G. Trafton, R. E. Wray, S. Mohan, and J. R. Kirk, "Interactive task learning," *IEEE Intelligent Systems*, vol. 32, no. 4, pp. 6–21, 2017.

[11] M. T. Shaikh and M. 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*. New York, NY, USA: ACM, pp. 389–397.

[12] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*, 4th ed. Springer Publishing Company, Inc., 2007.

[13] D. Bertsimas and J. Tsitsiklis, *Introduction to Linear Optimization*, 1st ed. Athena Scientific, 1997.

[14] R. Ramaswamy, J. B. Orlin, and N. Chakravarti, "Sensitivity analysis for shortest path problems and maximum capacity path problems in undirected graphs," *Mathematical Programming*, vol. 102, no. 2, pp. 355–369, Mar 2005.

[15] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed. New York: Springer, 2006.