2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

Revision of Specification Automata under Quantitative Preferences

Kangjin Kim and Georgios Fainekos

***Abstract*— We study the problem of revising specifications with preferences for automata based control synthesis problems. In this class of revision problems, the user provides a numerical ranking of the desirability of the subgoals in their specifications. When the specification cannot be satisfied on the system, then our algorithms automatically revise the specification so that the least desirable user goals are removed from the specification. We propose two different versions of the revision problem with preferences. In the first version, the algorithm returns an exact solution while in the second version the algorithm is an approximation algorithm with non-constant approximation ratio. Finally, we demonstrate the scalability of our algorithms and we experimentally study the approximation ratio of the approximation algorithm on random problem instances.**

I. Introduction

Linear Temporal Logic (LTL) has been widely adopted as a high-level specification language for robotic behaviors (see [1] for a recent overview). The wide spread adoption of LTL can be attributed to the tractable algorithms that can solve automation problems related to robotics (see [1]) and the connections to natural language [2] and other intuitive user interfaces [3]. In order for LTL-based control synthesis methods to move outside research labs and be widely adopted by the robotics community as a specification language of choice, specification debugging tools must be developed as well. In [4], [5], we studied the theoretical foundations of the specification automata revision problem and we proposed heuristic algorithms for its solution. In [6], we presented a version of the revision problem for weighted transition systems. In the last formulation, the debugging and revision problem becomes harder to solve since the specification could fail due to not satisfying certain cost constraints, such as, the battery capacity, certain time limit, etc.

Here, we revisit the problem posed in [4]. When automatically revising specifications, we are often faced with the challenge that not all goals have the same value for the user. In particular, we assume that the user has certain utility or preference value for each of the subgoals. Thus, an automatic specification revision should recommend removing the least desirable goals. In detail, we assume that the specification is provided as an ω-automaton, i.e., a finite automaton with Büchi acceptance conditions, and that each symbol labeling the transitions has a quantitative preference value (i.e., a positive number).

We formulate two different revision problems. The first problem concerns removing a set of symbols such that the synthesis problem has now a solution and the sum of the preference levels of the set of removed symbols is minimized. The second problem again seeks to remove a set of symbols such that the synthesis problem has now a solution; but now the largest preference level of the symbols in the removal set must be minimized.

Not surprisingly the former problem is intractable. However, interestingly, the latter problem can be solved in polynomial time. We show how the algorithm that we presented in [5] can be modified to provide an exact or approximate solution (depending on the cost function) to the revision problem with preferences in polynomial time. A practical implication of the results in this paper is that the user can now get an exact solution if the goal is to satisfy as many high preference goals as possible.

Contributions: We define two new versions of the problem of revision under quantitative preferences. We show that one version can be solved optimally in polynomial time while the other version of the problem is in general intractable. We provide an exact and an approximate, respectively, polynomial time algorithm based on Dijkstra's algorithm. Finally, we present some examples and we demonstrate the computational savings of our approximate algorithm over the Brute-Force Search Algorithm that solves the intractable version of the problem exactly.

Related Research: The problem of revising or resolving conflicting LTL specifications has received considerable attention recently. The closest work to ours is presented in [7]. The authors consider a number of high-level requirements in LTL which not all can be satisfied on the system. Each formula that is satisfied gains some reward. The goal of their algorithm is to maximize the rewards and, thus, maximize the number of requirements that can be satisfied on the system. Our problem definition is similar in spirit, but the problem goals are substantially different and the two approaches can be viewed as complementary. In [7], if a whole sub-specification cannot be realized, then it is aborted. In our case, we try to minimally revise the sub-specification so that it can be partially satisfied. Another substantial difference is that our proposed solutions can be incorporated directly within the control synthesis algorithm. Namely, as the algorithm searches for a satisfiable plan, it also creates the graph where the search for the revision will take place. In [7], the graph to be used for the revision must be constructed as a separate step.

The problem of LTL planning with qualitative preferences has been studied in [8], [9] (see also the references therein for more research in this direction). As opposed to revision problem, planning with preferences is based on the fact

This work has been partially supported by award NSF CNS 1116136.

K. Kim and G. Fainekos are with the School of Computing, Informatics and Decision Systems Engineering, Arizona State University, Tempe, AZ 85281, USA {Kangjin.Kim,fainekos}@asu.edu

that there are many satisfiable plans and, thus, the most preferable one should be selected. For LTL games, LTLMop [10] was developed to debug unrealizable LTL specifications in reactive planning for robotic applications. The problem of revising LTL specifications on-the-fly as the robot explores its environment is studied in [11].

In the context of general planners, the problem of finding good excuses on why the planning failed has been studied in [12]. Over-Subscription Planning (OSP) [13] and Partial Satisfaction Planning (PSP) [14] are also very related problems. The aforementioned approaches do not consider extended goals in LTL.

II. Preliminaries

In this paper, we work with discrete abstractions (Finite State Machines) of the continuous robotic control system [15]. Each state of the Finite State Machine (FSM) $\mathcal{T}$ is labeled by a number of symbols from a set $\Pi = \{\pi_0, \pi_1, \ldots, \pi_n\}$ that represent regions in the configuration space of the robot or, more generally, actions that can be performed by the robot.

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

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

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

In this work, we are interested in the specification automata that impose certain requirements on the traces of $\mathcal{T}$. In the following, $\mathcal{P}(\Pi)$ denotes the powerset of a set Π.

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

When $s' \in \delta_{\mathcal{B}_{\mathbf{s}}}(s, l)$, we also write $s \xrightarrow{l}_{\mathcal{B}_{\mathbf{s}}} s'$ or $(s, l, s') \in \rightarrow_{\mathcal{B}_{\mathbf{s}}}$. A *run* r of $\mathcal{B}_{\mathbf{s}}$ is a sequence of states $r : \mathbb{N} \rightarrow S_{\mathcal{B}_{\mathbf{s}}}$ that occurs under an input trace $\bar{p}$ taking values in $\mathcal{P}(\Pi)$. That is, for $i = 0$ we have $r(0) = s_0^{\mathcal{B}_{\mathbf{s}}}$ and for all $i \geq 0$ we have $r(i) \xrightarrow{\bar{p}(i)}_{\mathcal{B}_{\mathbf{s}}} r(i+1)$. Let $\lim(\cdot)$ be the function that returns the set of states that are encountered infinitely often in the run r of $\mathcal{B}_{\mathbf{s}}$. Then, a run r of an automaton $\mathcal{B}_{\mathbf{s}}$ over an infinite trace $\bar{p}$ is *accepting* if and only if $\lim(r) \cap F_{\mathcal{B}_{\mathbf{s}}} \neq \emptyset$. This is called a Büchi acceptance condition. Finally, we define the language $\mathcal{L}(\mathcal{B}_{\mathbf{s}})$ of $\mathcal{B}_{\mathbf{s}}$ to be the set of all traces $\bar{p}$ that have a run that is accepted by $\mathcal{B}_{\mathbf{s}}$.

In order to simplify the discussion in Section III, we will make the following assumption without loss of generality.

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

We will also be using the following notations.

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

In brief, our goal is to generate paths on $\mathcal{T}$ that satisfy the specification $\mathcal{B}_{\mathbf{s}}$ [15]. This can be achived by finding accepting runs on the product automaton $\mathcal{A} = \mathcal{T} \times \mathcal{B}_{\mathbf{s}}$.

Definition 3: The product automaton $\mathcal{A} = \mathcal{T} \times \mathcal{B}_{\mathbf{s}}$ is the automaton $\mathcal{A} = (S_{\mathcal{A}}, s_0^{\mathcal{A}}, \mathcal{P}(\Pi), \delta_{\mathcal{A}}, F_{\mathcal{A}})$ where:

- $\mathcal{S}_{\mathcal{A}} = Q \times S_{\mathcal{B}_{\mathbf{s}}}$,
- $s_0^{\mathcal{A}} = \{(q_0, s_0^{\mathcal{B}_{\mathbf{s}}}) \mid q_0 \in Q_0\}$,
- $\delta_{\mathcal{A}} : S_{\mathcal{A}} \times \mathcal{P}(\Pi) \rightarrow \mathcal{P}(S_{\mathcal{A}})$ s.t. $(q_j, s_j) \in \delta_{\mathcal{A}}((q_i, s_i), l)$ iff $q_i \rightarrow_{\mathcal{T}} q_j$ and $s_j \in \delta_{\mathcal{B}_{\mathbf{s}}}(s_i, l)$ with $l \subseteq h_{\mathcal{T}}(q_j)$,
- $F_{\mathcal{A}} = Q \times F$ is the set of accepting states.

We say that $\mathcal{B}_{\mathbf{s}}$ is *satisfiable* on $\mathcal{T}$ if $\mathcal{L}(\mathcal{A}) \neq \emptyset$. Moreover, finding a satisfying path on $\mathcal{T} \times \mathcal{B}_{\mathbf{s}}$ is an easy algorithmic problem [15]. Each accepting (infinite) run consists of two parts: **prefix:** a part that is executed only once (from an initial state to a final state) and, **lasso:** a part that is repeated infinitely (from a final state back to itself). Note that if the prefix or the lasso do not contain a final state, then the language $\mathcal{L}(\mathcal{A})$ is empty. Namely, the synthesis phase has failed and we cannot find a system behavior that satisfies the specification.

When a specification $\mathcal{B}$ is not satisfiable on a particular system $\mathcal{T}$, the current motion planning and control synthesis methods based on automata theoretic concepts [15]–[17] simply return that the specification is not satisfiable without any other user feedback. In such cases, our previous algorithms [4], [5] can provide as feedback to the user the closest revision under equal preference for all goals. Formally, a revision R is a subset of $\mathcal{P}(\Pi) \times E_{\mathcal{B}_{\mathbf{s}}}$. Each $(\pi, s, s') \in R$ indicates that π must be removed from $\lambda_{\mathcal{B}_{\mathbf{s}}}(s, s')$.

III. Revision Under Preferences

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

A revised specification is one that can be satisfied on the discrete abstraction of the workspace or the configuration space of the robot. In order to search for a minimal revision, we need first to define an ordering relation on automata as well as a distance function between automata. We do not want to consider the "space" of all possible automata, but rather the "space" of specification automata which are semantically close to the initial specification automaton $\mathcal{B}_{\mathbf{s}}$.

The later will imply that we remain close to the initial intention of the designer. We propose that this space consists of all the automata that can be derived from $\mathcal{B}_\mathbf{s}$ by removing symbols from the transitions. Our definition of the ordering relation between automata relies upon the previous assumption.

Definition 4 (Relaxation): Let $\mathcal{B}_1 = (S_{\mathcal{B}_1}, s_0^{\mathcal{B}_1}, \mathcal{P}(\Pi), \rightarrow_{\mathcal{B}_1}, F_{\mathcal{B}_1}, \theta_{\mathcal{B}_1})$ and $\mathcal{B}_2 = (S_{\mathcal{B}_2}, s_0^{\mathcal{B}_2}, \mathcal{P}(\Pi), \rightarrow_{\mathcal{B}_2}, F_{\mathcal{B}_2}, \theta_{\mathcal{B}_2})$ be two specification automata having the same preference levels for $\mathcal{P}(\Pi)$. Then, we say that $\mathcal{B}_2$ is a relaxation of $\mathcal{B}_1$ and we write $\mathcal{B}_1 \preceq \mathcal{B}_2$ if and only if $S_{\mathcal{B}_1} = S_{\mathcal{B}_2} = S$, $s_0^{\mathcal{B}_1} = s_0^{\mathcal{B}_2}$, $F_{\mathcal{B}_1} = F_{\mathcal{B}_2}$, $\theta_{\mathcal{B}_1} = \theta_{\mathcal{B}_2}$ and

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

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

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

Definition 5: Given a system $\mathcal{T}$ and and a specification automaton $\mathcal{B}_\mathbf{s}$, the set of *valid relaxations* of $\mathcal{B}_\mathbf{s}$ is defined as $\mathfrak{R}(\mathcal{B}_\mathbf{s}, \mathcal{T}) = \{\mathcal{B} \mid \mathcal{B}_\mathbf{s} \preceq \mathcal{B} \text{ and } \mathcal{L}(\mathcal{T} \times \mathcal{B}) \neq \emptyset\}$.

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

Definition 6 (Revision Set): Given a specification automaton $\mathcal{B}_\mathbf{s}$ and a $\mathcal{B} \in \mathfrak{R}(\mathcal{B}_\mathbf{s}, \mathcal{T})$, the revision set is defined as $R(\mathcal{B}_\mathbf{s}, \mathcal{B}) = \{(\pi, s, s') \mid \pi \in (\lambda_{\mathcal{B}_\mathbf{s}}(s, s') - \lambda_{\mathcal{B}}(s, s'))\}$.

We define two different revision problems.

Problem 1 (Min-Sum Revision): Given a system $\mathcal{T}$ and a specification automaton $\mathcal{B}_\mathbf{s}$, if the specification $\mathcal{B}_\mathbf{s}$ is not satisfiable on $\mathcal{T}$, then find a revision set R such that $\sum_{\rho \in R} \theta(\rho)$ is minimized.

Problem 2 (Min-Max Revision): Given a system $\mathcal{T}$ and a specification automaton $\mathcal{B}_\mathbf{s}$, if the specification $\mathcal{B}_\mathbf{s}$ is not satisfiable on $\mathcal{T}$, then find a revision set R such that $\max_{\rho \in R} \theta(\rho)$ is minimized.

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

Definition 7: Given a system $\mathcal{T}$ and a specification automaton $\mathcal{B}_\mathbf{s}$, we define the graph $G_\mathcal{A} = (V, E, v_s, V_f, \overline{\Pi}, \Lambda, p)$, which corresponds to the product $\mathcal{A} = \mathcal{T} \times \mathcal{B}_\mathbf{s}$ as follows

- $V = \mathcal{S}$ is the set of nodes
- $E = E_\mathcal{A} \cup E_D \subseteq \mathcal{S} \times \mathcal{S}$, where $E_\mathcal{A}$ is the set of edges that correspond to transitions on $\mathcal{A}$, i.e., $((q, s), (q', s')) \in E_\mathcal{A}$ iff $\exists l \in \mathcal{P}(\Pi) . (q, s) \xrightarrow{l}_\mathcal{A} (q', s')$; and E_D is the set of edges that correspond to disabled transitions, i.e., $((q, s), (q', s')) \in E_D$ iff $q \rightarrow_\mathcal{T} q'$ and $s \xrightarrow{l}_{\mathcal{B}_\mathbf{s}} s'$ with $l \cap (\Pi - h_\mathcal{T}(q')) \neq \emptyset$
- $v_s = s_0^\mathcal{A}$ is the source node
- $V_f = F_\mathcal{A}$ is the set of sinks
- $\overline{\Pi} = \{\langle \pi, (s, s') \rangle \mid \pi \in \Pi, (s, s') \in E_{\mathcal{B}_\mathbf{s}}\}$
- $\Lambda : E \rightarrow \mathcal{P}(\overline{\Pi})$ is the edge labeling function such that if $e = ((q, s), (q', s'))$, then
$$\Lambda(e) = \{\langle \pi, (s, s') \rangle \mid \pi \in (\lambda_{\mathcal{B}_\mathbf{s}}(s, s') - h_\mathcal{T}(q'))\}.$$
- $\theta : \overline{\Pi} \rightarrow \mathbb{R}_{\geq 0}$ is the preference function of $\mathcal{B}_\mathbf{s}$ restricted on $\overline{\Pi}$.

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

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

First, we study the computational complexity of the two problems by restricting the search problem only to paths from source (initial state) to sink (accept state). Let $Paths(G_\mathcal{A})$ denote all such paths on $G_\mathcal{A}$. We indicate that the graph search equivalent problem of Problem 2 is in P. Given a path $p = v_s v_1 v_2 \dots v_f$ on $G_\mathcal{A}$ with $v_f \in V_f$, we define the max-preference level of the path to be:

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

Note that this is the same as the original cost function in Problem 2 since clearly $\max_{(v_i, v_{i+1}) \in p} \theta(\Lambda(v_i, v_{i+1})) = \max_{\rho \in R} \theta(\rho)$ where $R = \cup_{(v_i, v_{i+1}) \in p} \Lambda(v_i, v_{i+1})$. Thus, Problem 2 is converted into the following optimization problem:

$$p^* = \arg \min_{p \in Paths(G_\mathcal{A})} \theta(p) \tag{1}$$

And, thus, the revision will be $R = \cup_{(v_i, v_{i+1}) \in p^*} \Lambda(v_i, v_{i+1})$. Now, we recall the weak optimality principle [18].

Definition 8 (Weak optimality principle): There is an optimal path formed by optimal subpaths.

Proposition 1: The graph search equivalent of Problem 2 satisfies the weak optimality principle.

The importance of the weak optimality principle being satisfied is that label correcting and label setting algorithms can be applied to such problems [18]. Dijkstra's algorithm is such an algorithm [19] and, thus, it can provide an exact solution to the problem.

Now, we proceed to the Min-Sum preference problem. Given a path $p = v_s v_1 v_2 \dots v_f$ on $G_\mathcal{A}$ with $v_f \in V_f$, we define the sum-preference level of the path to be:

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

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

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

Problem 3: Labeled Path under Additive Preferences (LPAP). INPUTS: A graph $G_{\mathcal{A}} = (V, E, v_s, V_f, \overline{\Pi}, \Lambda, \theta)$, and a preference bound $K \in \mathbb{N}$. OUTPUT: a set $R \subseteq \overline{\Pi}$ such that removing all elements in R from edges in E enables a path from v_s to some final vertex $v_f \in V_f$ and $\theta_+(R) \leq K$.

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

Theorem 1: Given an instance of the LPAP $(G_{\mathcal{A}}, K)$, the decision problem of whether there exists a path p such that $\theta_+(p) \leq K$ is NP-Complete.

IV. Algorithms for the Revision Problem with Preferences

In this section, we present Algorithms for the Revision Problem with Preferences (ARPP). It is based on the Approximation Algorithm of the Minimal Revision Problem (AAMRP) [5] which is in turn based on Dijkstra's shortest path algorithm [19]. The main difference from AAMRP is that instead of finding the minimum number of atomic propositions that must be removed from each edge on the paths of the graph $G_{\mathcal{A}}$, ARPP tracks paths having atomic propositions that minimize the preference level from each edge on the paths of the graph $G_{\mathcal{A}}$.

Here, we present the pseudocode for ARPP. ARPP is similar to AAMRP in [5]. The difference from [5] is that AARP uses PREF function instead of using cardinality of the set. For Min-Sum Revision, the function PREF: $\overline{\Pi} \to \mathbb{R}_{\geq 0}$ is defined as following: given a set of label $R \subseteq \overline{\Pi}$ and the preference function $\theta_+ : \overline{\Pi} \to \mathbb{R}_{\geq 0}$, PREF$(R) = \theta_+(R)$. The Min-Sum ARPP is denoted by $ARPP_+$. For Min-Max Revision, the function PREF: $\overline{\Pi} \to \mathbb{R}_{\geq 0}$ is defined as following: given a set of label $R \subseteq \overline{\Pi}$ and the preference function $\theta : \overline{\Pi} \to \mathbb{R}_{\geq 0}$, PREF$(R) = \max_{\rho \in R} \theta(\rho)$. The Min-Max ARPP is denoted by $ARPP_{max}$.

The main algorithm (Alg. 1) divides the problem into two tasks. First, in line 5, it finds an approximation to the minimum preference level of atomic propositions from $\overline{\Pi}$ that must be removed to have a prefix path to each reachable sink (see Section II). Then, in line 9, it repeats the process from each reachable final state to find an approximation to the minimum preference level of atomic propositions from $\overline{\Pi}$ that must be removed so that a lasso path is enabled. The combination of prefix/lasso that removes the least preferable atomic propositions is returned to the user. Due to space limitations, we omit Algorithm 2 FINDMINPATH and Algorithm 3 RELAX (for details, see [20]).

The analysis of the algorithm ARPP follows closely the analysis of AAMRP in [5]. The only difference in the time complexity is that ARPP uses PREF function in order to compute preference levels of all elements in $\overline{\Pi}$. Both Min-Sum Revision and Min-Max Revision take $O(\overline{\Pi})$ since at most they compute preference levels of all elements in $\overline{\Pi}$. Hence, the running time of FINDMINPATH is $O(E(\overline{\Pi}^2 \log \overline{\Pi} + \log V))$. Therefore, the running time of ARPP is $O(V_f(V\overline{\Pi} \log \overline{\Pi} + E(\overline{\Pi}^2 \log \overline{\Pi} + \log V))) = O(V_f E(\overline{\Pi}^2 \log \overline{\Pi} + \log V))$ which is polynomial in the size of the input graph.

Algorithm 1 ARPP

Inputs: a graph $G_{\mathcal{A}} = (V, E, v_s, V_f, \overline{\Pi}, \Lambda, \theta)$.
Outputs: the list L of symbols from $\overline{\Pi}$ that must be removed from $\mathcal{B}_{\mathbf{s}}$.

```
1:  procedure ARPP(G_A)
2:      L ← Π̄
3:      M[:,:] ← (Π̄, ∞)            ▷ Each row is set to (Π̄, ∞)
4:      M[v_s,:] ← (∅, 0)           ▷ Initialize the source node
5:      ⟨M, P, V⟩ ← FINDMINPATH(G_A, M, 0)
6:      if V ∩ V_f = ∅ then L ← ∅
7:      else
8:          for v_f ∈ V ∩ V_f do
9:              L_p ← GETAPFROMPATH(v_s, v_f, M, P)
10:             M'[:,:] ← (Π̄, ∞)
11:             M'[v_f,:] ← M[v_f,:]
12:             G'_A ← (V, E, v_f, {v_f}, Π̄, L)
13:             ⟨M', P', V'⟩ ← FINDMINPATH(G'_A, M', 1)
14:             if v_f ∈ V' then
15:                 L_l ← GETAPFROMPATH(v_f, v_f, M', P')
16:                 if PREF(L_p ∪ L_l) ≤ PREF(L) then
17:                     L ← L_p ∪ L_l
18:                 end if
19:             end if
20:         end for
21:     end if
22:     return L
23: end procedure
```

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

V. Example and Experiments

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

In the following example, we will be using LTL as a specification language. We remark that the results presented here can be easily extended to LTL formulas by renaming repeated occurrences of atomic propositions in the specification and adding them on the transition system (for details, see [21]).

The following example scenario was inspired by [16], [22], and we will be using LTL as a specification language.

Example 1 (Single Robot Data Gathering Task): In this example, we use a simplified road network having three gathering locations and two upload locations with four intersections of the road. In Fig. 1, the data gather locations, which are labeled g_1, g_2, and g_3, are dark gray, the data upload locations, which are labeled u_1 and u_2, are light gray, and the intersections are labeled i_1 through i_4. In order to gather data and upload the gather-data persistently, the following LTL formula may be considered: $\phi_A :=$ GF$(\varphi) \wedge$ GF(π), where $\varphi := g_1 \vee g_2 \vee g_3$ and $\pi := u_1 \vee u_2$. The following formula can make the robot move from gather locations to upload locations after gathering data: $\phi_G :=$

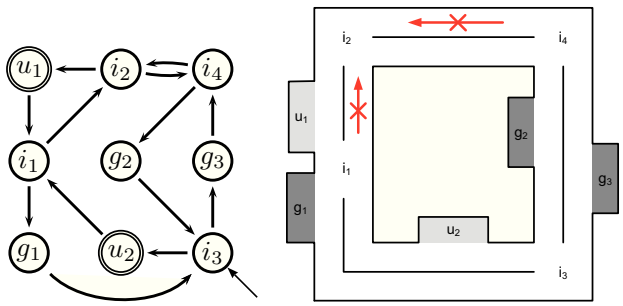

Fig. 1: Illustration of the simple road network environment of Example 1. The robot is required to drive right-side of the road.

$\mathrm{G}(\varphi \rightarrow \mathrm{X}(\neg\varphi \mathcal{U} \pi))$. In order for the robot to move to gather location after uploading, the following formula is needed: $\phi_U := \mathrm{G}(\pi \rightarrow \mathrm{X}(\neg\pi \mathcal{U} \varphi))$.

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

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

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

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

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

Now, we present some experimental results. The propotype implementation is written in Python. For the experiments, we utilized the ASU super computing center which consists of clusters of Dual 4-core processors, 16 GB Intel(R) Xeon(R) CPU X5355 @2.66 Ghz. The operating system is CentOS release 5.9. The clusters were used to run each test case on each single core in parallel.

In order to assess the experimental approximation ratio of the heuristic (Min-Sum Revision), we compared the solutions returned by the heuristic with Brute-force search algorithm. The Brute-force search is guaranteed to return a minimal solution to the Min-Sum Revision problem. We omit the explanation of the each test case and full experiment results (for details, see [20]).

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

Table II shows the results of ARPP for Min-Max Revision ($ARPP_{max}$) on test cases of different sizes (total number of nodes). This test results also used same test cases as the ones for Table I. We report minimum, average, and maximum computation times in sec. The "avg # nodes" shows the average number of nodes returned from $ARPP_{max}$.

Nodes	Brute-Force				Min-Sum Revision ($ARPP_+$)					RATIO		
	min	avg	max	succ	min	avg	max	succ	avg # nodes	min	avg	max
9	0.033	0.0921	0.945	200/200	0.019	0.183	0.874	200/200	1.305	1	1	1
100	0.065	0.3707	3.997	200/200	0.065	0.1598	2.66	200/200	1.95	1	1.003	1.619
196	0.278	303.55	11974	199/200	0.137	0.4927	12.057	200/200	2.305	1	1.0014	1.1475

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

Nodes	Min-Max Revision ($ARPP_{max}$)				
	min	avg	max	succ	avg # nodes
9	0.02	0.0508	0.66	200/200	1.785
100	0.061	0.1258	0.471	200/200	3.215
196	0.139	0.29824	0.74	200/200	3.84

TABLE II: Numerical Experiments: For each graph $G_{\mathcal{A}}$, Number of nodes versus the result of ARPP for Min-Max Revision ($ARPP_{max}$). Under the min, avg, max columns the numbers indicate computation times in sec.

VI. Conclusions

This paper discusses the problem of specification revision with user preferences. We have demonstrated that adding preference levels to the goals in the specification can render the revision problem easier to solve under the appropriate cost function. We view the automatic debugging and specification revision problems as foundational for formal methods to receive wider adoption in the robotics community and beyond. With the current paper and the predecessors [4]–[6], [23], we have studied the theoretical foundations of different versions of the problem. Our algorithms and tools can be used as add-ons to control synthesis methods developed by our and other groups [15]–[17], [24], [25]. Our goal for the future is to incorporate all the specification revision methods in a comprehensive user-friendly tool that can run on different platforms.

Acknowledgments

The authors would like to thank the anonymous reviewers for their detailed comments.

References

[1] H. Kress-Gazit, "Robot challenges: Toward development of verification and synthesis techniques [errata]," *IEEE Robotics Automation Magazine*, vol. 18, no. 4, pp. 108–109, Dec. 2011.

[2] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Translating structured english to robot controllers," *Advanced Robotics*, vol. 22, no. 12, pp. 1343–1359, 2008.

[3] S. Srinivas, R. Kermani, K. Kim, Y. Kobayashi, and G. Fainekos, "A graphical language for LTL motion and mission planning," in *Proceedings of the IEEE International Conference on Robotics and Biomimetics*, 2013.

[4] K. Kim, G. Fainekos, and S. Sankaranarayanan, "On the revision problem of specification automata," in *Proceedings of the IEEE Conference on Robotics and Automation*, May 2012.

[5] K. Kim and G. Fainekos, "Approximate solutions for the minimal revision problem of specification automata," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[6] ——, "Minimal specification revision for weighted transition systems," in *Proceedings of the IEEE Conference on Robotics and Automation*, May 2013.

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

[8] T. C. Son, E. Pontelli, and C. Baral, "A non-monotonic goal specification language for planning with preferences," in *6th Multidisciplinary Workshop on Advances in Preference Handling*, 2012.

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

[10] V. Raman and H. Kress-Gazit, "Analyzing unsynthesizable specifications for high-level robot behavior using LTLMoP," in *23rd International Conference on Computer Aided Verification*, ser. LNCS, vol. 6806. Springer, 2011, pp. 663–668.

[11] M. Guo, K. H. Johansson, and D. V. Dimarogonas, "Revising motion planning under linear temporal logic specifications in partially known workspaces," in *Proceedings of the IEEE Conference on Robotics and Automation*, 2013.

[12] M. Göbelbecker, T. Keller, P. Eyerich, M. Brenner, and B. Nebel, "Coming up with good excuses: What to do when no plan can be found," in *Proceedings of the 20th International Conference on Automated Planning and Scheduling*. AAAI, 2010, pp. 81–88.

[13] D. E. Smith, "Choosing objectives in over-subscription planning," in *Proceedings of the 14th International Conference on Automated Planning and Scheduling*, 2004, p. 393401.

[14] M. van den Briel, R. Sanchez, M. B. Do, and S. Kambhampati, "Effective approaches for partial satisfaction (over-subscription) planning," in *Proceedings of the 19th national conference on Artifical intelligence*. AAAI Press, 2004, p. 562569.

[15] G. E. Fainekos, A. Girard, H. Kress-Gazit, and G. J. Pappas, "Temporal logic motion planning for dynamic robots," *Automatica*, vol. 45, no. 2, pp. 343–352, Feb. 2009.

[16] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus, "Optimal multi-robot path planning with temporal logic constraints," in *IEEE/RSJ International Conference on Intelligent Robots and Systems,*, 2011, pp. 3087 –3092.

[17] A. LaViers, M. Egerstedt, Y. Chen, and C. Belta, "Automatic generation of balletic motions," *IEEE/ACM International Conference on Cyber-Physical Systems*, vol. 0, pp. 13–21, 2011.

[18] E. Martins, M. Pascoal, D. Rasteiro, and J. Dos Santos, "The optimal path problem," *Investigacão Operacional*, vol. 19, pp. 43–60, 1999.

[19] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006. [Online]. Available: http://msl.cs.uiuc.edu/planning/

[20] K. Kim and G. Fainekos, "Revision of specification automata under quantitative preferences," Cornell University Library arXiv.org, Tech. Rep., 2014.

[21] LTL2BA modification. [Online]. Available: https://www.assembla.com/code/ltl2ba_cpslab/git/nodes

[22] A. Ulusoy, S. L. Smith, X. C. Ding, and C. Belta, "Robust multi-robot optimal path planning with temporal logic constraints," in *2012 IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[23] G. E. Fainekos, "Revising temporal logic specifications for motion planning," in *Proceedings of the IEEE Conference on Robotics and Automation*, May 2011.

[24] L. Bobadilla, O. Sanchez, J. Czarnowski, K. Gossman, and S. LaValle, "Controlling wild bodies using linear temporal logic," in *Proceedings of Robotics: Science and Systems*, Los Angeles, CA, USA, June 2011.

[25] E. M. Wolff, U. Topcu, and R. M. Murray, "Automaton-guided controller synthesis for nonlinear systems with temporal logic," in *International Conference on Intelligent Robots and Systems*, 2013.