Enumerating Preferred Solutions to Conditional Simple Temporal Networks Quickly Using Bounding Conflicts

Eric Timmons and **Brian C. Williams**
Massachusetts Institute of Technology
Computer Science and Artificial Intelligence Laboratory
32 Vassar Street, Cambridge, MA 02139
{etimmons,williams}@csail.mit.edu

Abstract

To achieve high performance, autonomous systems, such as science explorers, should adapt to the environment to improve utility gained, as well as robustness. Flexibility during temporal plan execution has been explored extensively to improve robustness, where flexibility exists both in activity choices and schedules. These problems are framed as conditional constraint networks over temporal constraints. However, flexibility has been exploited in a limited form to improve utility. Prior work considers utility in choice or schedule, but not their coupling. To exploit fully flexibility, we introduce conditional simple temporal networks with preference (CSTNP), where preference is a function over both choice and schedule.

Enumerating best solutions to a CSTNP is challenging due to the cost of scheduling a candidate STPP and the exponential number of candidates. Our contribution is an algorithm for enumerating solutions to CSTNPs efficiently, called *A star with bounding conflicts (A*BC)*, and a novel variant of conflicts, called *bounding conflicts*, for learning heuristic functions. A*BC interleaves Generate, Test, and Bound. When A*BC bounds a candidate, by solving a STPP, it generates a bounding conflict, denoting neighboring candidates with similar bounds. A*BCs generator then uses these conflicts to steer away from sub-optimal candidates.

Introduction

Complex autonomous vehicle missions, in space, air, and sea, are enabled through temporal plan executives that perform coordinated, time-critical missions both robustly and with high utility. Past work has focused significantly on achieving robustness. In this paper we extend planning and execution systems to achieve high levels of utility as well.

This research is developed in support of autonomous ocean observing, in joint collaboration with the Woods Hole Oceanographic Institute. Autonomous underwater vehicles exploit on board instruments to perform surveys over a period of several days. A mission involves several survey activities over different areas of interest, where activities include bathymetric mapping, characterization of physical and chemical attributes, and classification of biological phenomena. Scientists specify preference in terms of the areas to be explored, phenomena to be mapped, time allocated to each area and the relative importance of these science activities. Additional operational constraints are introduced, for example, due to weather, resupply operations, and traffic in the area. High utility and robustness are both key to these missions, in terms of science information gained, and risk of mission loss.

Research on flexible execution has focused dominantly on robustness. In this approach, mission coordination is specified through temporal plans that include both concurrent and sequential activities, related through metric temporal constraints. Robustness is achieved by introducing flexibility in the temporal plan representation, which is exploited by the executives to make run-time decisions.

Flexibility typically takes two forms. First, flexibility in the metric temporal constraints is used to specify a family of consistent schedules to start times and duration of activities. Two commonly used formalisms are temporal constraint and simple temporal constraint networks (Dechter, Meiri, and Pearl 1991). Additional flexibility is commonly achieved by representing a family of alternative plans, by expressing a "contingent choice" between activities or subplans. Two commonly used representations for offline planning and online temporal plan execution, respectively, are hierarchical task networks (HTN) (Tate 1976; Nau et al. 2003) and temporal plan networks (TPNs) (Kim, Williams, and Abramson 2001). For TPNs these two forms of flexibility, contingent choice and metric temporal constraints, are encoded as a *conditional, simple temporal network (CSTN)*, a conditional constraint network (CCN) restricted to simple temporal constraints.[1] Solutions to CSTNs are extracted efficiently by using conflict-directed search to identify and prune large sets of infeasible candidates (Ginsberg 1993; Dago and Verfaillie 1996; Williams and Ragno 2003; Effinger and Williams 2006), and by using incremental temporal consistency algorithms to test consistency quickly, while extracting conflicts(Shu, Effinger, and Williams 2005).

This paper focuses on developing temporal planners that

[1]A conditional constraint network is one in which variables and constraints are "activated" by assignments to variables. A consistent assignment to a CCN is an assignment to active variables that satisfies active constraints.

exploit flexibility, for example in TPN specifications, to achieve high utility, and that generate solutions quickly using novel conflict learning methods that focus on utility, rather than feasibility. High utility, as well as robustness, can be achieved by exploiting flexibility in plan choice and its schedule. The activities chosen can substantially influence utility. For example, when performing an observing mission, the information gained by performing the mission can be influenced significantly by the area that is selected for observation, and the search pattern used to observe that area; for example, edge-following patterns versus "lawn mower" patterns. Both choices can be encoded through a choice between alternative activities or sub-plans, and utility can be associated with the activities that comprise the alternative sub-plans. In addition, utility is often a function of the time allocated to each plan activity, where this function depends on the activities selected for the plan. For example, the amount of time allocated to a particular search area influences the resolution at which the area is observed, and hence the information gained.

TPN planners, for example, associate utility with activities and choice, and return the optimal consistent plan, by performing conflict-directed branch and bound or best-first search (Effinger and Williams 2006; Williams and Ragno 2003). These planners, however, do not associate utility with duration. Conversely, significant attention has been devoted to developing scheduling algorithms that return the schedule with highest utility for problems framed as simple temporal problems with preference (STPPs) (Khatib et al. 2001; Morris et al. 2004). However, these schedulers do not allow alternative activities to be chosen. In this paper we unify these formalisms by introducing *conditional simple temporal networks with preference (CSTNP),* which can be used, for example, to encode TPNs with preference.

Mission operators often prefer to be supplied a set of good alternative options to choose from, rather than a single "best" option. Hence we frame our problem as one of best-first enumeration, rather than simply optimization.

Our main contribution is a new search algorithm for enumerating the best solutions to CSTNPs efficiently, called *A* with bounding conflicts (A*BC)*, and a novel variant of conflicts, called *bounding conflicts*, which are used to learn heuristic functions. The technical challenge of enumerating CSTNPs is the significant computational cost of finding the optimal schedule for a single candidate solution, coupled with the large number of candidate solutions considered, which is worst case exponential in the number of discrete choices. Our approach begins with a conflict-directed best-first search algorithm, similar to those employed in past optimal TPN planners. (Feasibility) conflicts are used to learn and prune large sets of infeasible candidates, while an admissible heuristic (equivalently a bounding function) is used to guide the search for an optimal plan. Computing a bound by solving a simple temporal problem with preference can be costly; for some preference models it involves solving a linear or convex program.

Our solution is to learn a heuristic bounding function automatically through a process analogous to feasibility conflicts. Traditionally, whenever conflict-directed search finds an infeasible candidate, it learns sets of infeasible states, which it encodes as a *(feasibility) conflict*, denoting a small partial assignment that is inconsistent with a problems constraints. The search algorithm's candidate generator then use these conflicts to generate candidates outside the known conflicts.

Likewise, whenever A*BC bounds a candidate by solving a STPP, it analyzes the STPP solution to learn a bound over similar candidates. The results are then recorded as a *bounding conflict*, a set of states, denoted by a partial assignment, and a corresponding bound over that set. A*BC's candidate generator then uses both bounding and feasibility conflicts to efficiently guide search away from sub-optimal and infeasible sets of candidates, towards the next best feasible candidate. As our empirical results illustrate, the use of these two types of conflicts substantially reduce the number of candidates that need to be tested for bounding and consistency. The remaining generated candidates are then tested through incremental temporal consistency, and bounded using an STPP algorithm that is suitable to the particular preference function being employed.

In the remainder of this paper we first elaborate on related work. We next introduce a pedagogical example taken from our ocean observing research, and use this example to illustrate our problem statement, consisting of the best-first enumeration of solutions to a Conditional Simple Temporal Network with Preferences. The core of our paper is a presentation of bounding conflicts and the A* with bounding-conflicts algorithm, for enumerating CSTNPs. Finally, we compare the results of enumerating CSTNP problems using A*BC and a mixed-integer linear program encoding, illustrating a significant improvement using A*BC.

Related Work

Existing work addresses many of the elements of our problem individually, but not in combination. As discussed above, temporal plan networks (TPNs) (Kim, Williams, and Abramson 2001; Effinger 2006; Robertson, Effinger, and Williams 2006) combine nested discrete choice with temporal constraints, and include preference over choice, but not over duration, meanwhile Simple Temporal Problems with Preference (Khatib et al. 2001; Morris et al. 2004) offer a rich preference model over duration, but not over discrete choices. (Santana and Williams 2013) generalizes TPNs to both controllable and uncontrollable choice, but associates preference with choice only, not duration.

Turning to other temporal representations, disjunctive temporal problems with preference (Peintner and Pollack 2004) provide preference over individual simple temporal constraints, and a choice between simple temporal constraints, but these choices can not be nested or coupled, as in a TPN or conditional CSP.

Controllable conditional temporal problems (Yu and Williams 2013) include conditional choice and preference over time, conditioned on choice, but preference is on the degree of relaxation of a temporal constraint, rather than duration itself.

Conditional temporal problems with preference (CTPPs)(Falda, Rossi, and Venable 2007;

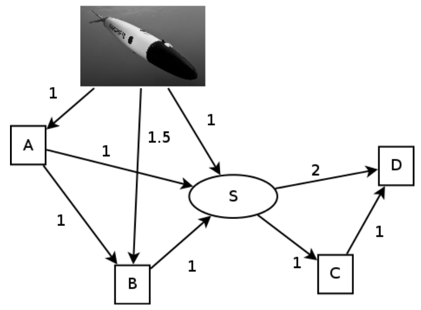

Figure 1: Minimum travel times between starting position, science locations (A-D), and satellite uplink location (S).

2010) include events and constraints that are activated by assignments to discrete variables, and offer a rich set of preference functions over durations that are conditioned on discrete choice. However, CTPPs presume that these assignments are made by the environment, and hence are uncontrollable.[2] Our motivation is instead to support controllable decisions, made by the agent.

Ocean Observing Example

Throughout this presentation we illustrate our work with a simplified underwater science example. Consider a mission to observe several areas of the ocean with an underwater glider. There are four different areas of the ocean to observe: A, B, C, and D. Additionally, there is a special safe location (S) where the glider can surface, achieve a satellite link, and upload its data. The scientists in charge of the expedition have decided that surveying each region is optional. However, due to prevailing currents, any science locations visited must be visited in alphabetical order (A before the rest, B before C and D, and C before D). Additionally, the glider must upload any data it has before proceeding to C or D. Figure 1 shows the minimum travel times in hours between all regions.

If a location is observed, the glider must spend at least one hour at the location to justify the visit. Additionally, the scientists would like to spend as much time as possible at each site, but no more than two hours. The scientists express their preference on the duration at each site as a linear function of the duration, specifically:

$$r(d) = 5d - 4 \quad (1)$$

where d is the time spent in the region.

Due to constraints on the satellite uplink, the glider much reach location S within five hours (we assume for the purposes of this example that the uplink is instantaneous). Last, due to battery constraints, the entire mission must last no more than ten hours.

[2](Tsamardinos et al. 2003) introduced the term "conditional temporal problem" to refer to temporal constraints that are conditioned on uncontrollable, sensing actions. In this paper we adopt the more common use of the term "conditional," introduced in the constraint programming community, in which choices are controllable.

The goal of the solver is to enumerate decisions regarding which regions to visit, together with the most preferred schedule for each decision, in decreasing order of the scientists' preference.

Problem Statement

Past work specifies a temporal plan with choice as a TPN, which maps simply to a conditional STN. In this paper we extend CSTNs to preference. The analogous extension to TPNs is straight forward. A Conditional Simple Temporal Network with Preference (CSTNP) describes a set of temporal events, constraints, and discrete decisions. Each constraint, event and decision can be guarded so that it becomes active only if certain decisions are made, represented by an assignment to finite domain decision variables. Additionally, a CSTNP provides a preference function that, given a set of decisions and a schedule for the events, returns a reward. A CSTNP extends a Conditional Simple Temporal Network (CSTN)(Yu and Williams 2013) to include preferences over both decision variables and events.

In this paper, we describe a solver for CSTNPs that uses bounding conflicts to direct its enumeration of the best candidate solutions.

Definition 1. *A CSTNP solver based on bounding conflicts receives as input:*

- *a CSTNP,*
- *an admissible heuristic $H : E \rightarrow \Re$ for preference function R of CSTNP, and*
- *an STNP solver: $STNP \rightarrow S \times B$, where S is an optimal consistent schedule of STNP, and B is a bounding conflict of S and STNP.*

It then enumerates consistent assignments to the CSTNP's decision variables in best-first order, based on the value of the assignment's most preferred schedule, generated by the STNP solver for the assignment's active constraints.

We now define CSTNPs.

Definition 2. *A conditional simple temporal network with preference (CSTNP) is defined by a tuple $\langle D, E, C, L_D, L_E, L_C, R\rangle$, where:*

- *D is a set of finite domain decision variables,*
- *E is a set of events, ranging over the reals, representing time points,*
- *C is a set of simple temporal constraints between pairs of events $e_1, e_2 \in E$, of the form $e_2 - e_1 \in [l, u]$,*
- *$L_D : D \rightarrow Q$ is a function that attaches guards, conjunctions of assignment to variables in $Q \subseteq D$, to decision variables D,*
- *$L_E : E \rightarrow Q$ is a function that attaches guards, conjunctions of assignment to variables in $Q \subseteq D$, to events E,*
- *$L_C : C \rightarrow Q$ is a function that attaches guards, conjunctions of assignments to variables in $Q \subseteq D$, to temporal constraints C,*
- *$R : D \times T \rightarrow \mathbb{R}$ is a preference function that maps a complete assignment to the decision variables and events (a schedule) to the reals.*

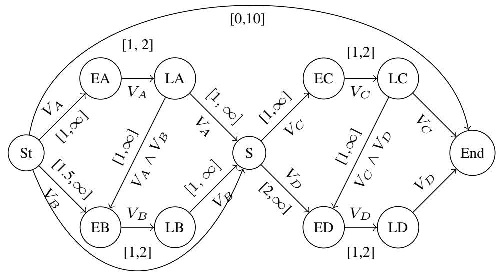

Figure 2: The example problem, modeled as a CCTPP. The variables V_A, V_B, V_C, and V_D represent the decision to explore regions A, B, C, and D, respectively.

The functions L_D, L_E and L_C describe conditions under which discrete variables, events, and constraints are active, in the form of guards. This allows decision variables, events and constraints to be conditioned ("guarded") on the decisions made over other variables in D. Each guard is a partial assignment to the decision variables D. Given an assignment A to D an event or discrete variable is considered *active* if its guard holds in A. A constraint is considered *active* if its guard holds, and if both temporal events in its scope are active as well.

Additionally, we assume that the dependency diagram for the activation of decision variables through guards contains no cycles. That is, the activation of variable D_i cannot depend on variable D_j being assigned a certain value if the activation of D_j is dependent on variable D_i taking on a specific value.

Definition 3. *A consistent solution to a CSTNP is described by a pair $\langle D, T \rangle$, where:*

- *D is a full assignment to all active decision variables of the CSTNP and*
- *T is a schedule consisting of a full assignment to all active events of the CSTNP that is consistent with all active constraints.*

Figure 2 shows the example problem modeled as a CSTNP. The decisions in the example are whether regions A, B, C, and/or D should be visited. To represent these decisions, we introduce the binary valued variables $V_A, V_B, V_C,$ andV_D. Events such as EA and LA represent entering region A and leaving region A, respectively. These events are guarded on the choice to explore that region (not pictured). The arcs between events represent simple temporal constraints. If a constraint is guarded, its guard appears next to the constraint. Note that, in the example, no decision variable is guarded.

Approach

Our proposed approach splits the enumeration problem into two components. The first component is a best candidate generator. In this component, a discrete, best-first search algorithm produces a candidate best assignment to the CSTNP's decision variables. The generator uses an A*-like algorithm and a heuristic function to guide search as partial assignments are extended. In addition, as candidates are tested, the generator uses bounding and infeasibility conflicts learned during testing, to improve its search. This is the key to the efficiency of our method.

Once the best-first search algorithm generates a full candidate assignment to the decision variables, the second component performs optimal scheduling to both test feasibility of the candidate and to bound its value, returning the results to the generator. In addition, the second component generalizes the scheduling results to similar candidates, through a process of conflict learning. If a schedule exists, candidates are identified whose optimal schedules have similar bounds, and this set is summarized as a bounding conflict. If no schedule exists, candidates are identified that are similarly infeasible, and this set is summarized by a standard (feasibility) conflict. Bounding and feasibility conflicts are both returned to the generator, to help focus search.

This process generalizes upon the generate and test process, performed by conflict-directed A*(Williams and Ragno 2003), which is used, for example, to enumerate the preferred plans of a TPN. Several differences are key. First, conflict-directed A* was designed to solve problems in which utility is a function of the discrete decision variables alone. Our constraint networks, CSTNPs, specify utility over a combination of discrete decision variables and real-valued events. Second, the tester for Conflict-directed A* tests consistency, but does not need to solve an optimization problem or compute a bound. This is because only the discrete decision variables influence cost. Our approach, however, requires that the tester solve an optimization sub-problem, in order to compute a bound, as well as to test feasibility. Third, as discussed earlier, solving this optimization sub-problem is often costly, hence our tester learns conflicts that generalize the results of bounding, as wells as consistency tests. Finally, the learned conflicts passed to the generator improve upon the generator's heuristic, through bounding conflicts, as well as the generator's constraints, through feasibility conflicts.

Thus far, for pedagogical purposes we have introduced bounding and feasibility conflicts as two separate concepts. In our development below we encode feasibility conflicts as a special case of bounding conflicts, by making their cost bound infinitely positive, or equivalently, their utility bound infinitely negative. With this treatment, bounding conflicts and the A*BC algorithm offer strict generalizations of feasibility conflicts and the conflict-directed A* algorithm.

In the following, we first describe the optimal scheduling and conflict learning module (Tester). In addition, we define bounding conflicts and explain how they can be used as a bounding function. We then describe the search algorithm underlying the best candidate generator. To remain consistent with the example problem, we present the algorithms as maximizing utility. The modifications to minimize cost are straightforward.

Optimal Scheduling and Conflict Learning

The role of the tester is to determine an optimal schedule, given a set of decisions made by the generator. The separation of the scheduler from the generator allows for a range of schedulers to be used, and a range of corresponding temporal constraint and preference representations, without modifying the core reasoning algorithms in the generator.

The input to the scheduler is a CSTNP and D an assignment to the problem's decision variables. Given this assignment, the scheduler simplifies the CSTNP to a simple temporal network with preference (STNP) (Khatib et al. 2001), by removing all events and constraints whose guards are not activated by the decisions. We denote the resulting STNP by CSTNP$_D$.

Once the CSTNP$_D$ is determined, the scheduler extracts its optimal schedule. Any STPP scheduler can be used that supports the preference function provided. In our experiments we use a linear preference function per simple temporal constraint, and employ a Linear Program solver to compute the optimal schedule.

The same approach applies to more general temporal constraint formulations as well. If disjunctive constraints were employed, then a DTPP scheduler, such as the one described in (Peintner and Pollack 2004), can be used. If looping temporal constraints are employed, a looping temporal problem with preference (LTPP) scheduler, such as the one described in (Paterson, Timmons, and Williams 2014), can be used.

The scheduler is also responsible for extracting bounding conflicts, which are supplied to the best candidate generator.

Bounding Conflicts A key innovation of our approach is to enable the discrete search algorithm (generator) to learn and exploit a tighter bounding function, as it performs search. We accomplish this by representing the learned bounding function using a set of rules, which we have already referred to as *bounding conflicts*. The rule antecedent is a partial assignment to decision variables, while its consequent is a bounding function. A rule triggers on any partial assignment that contains its antecedent.

Definition 4. *A bounding conflict is a pair $\langle z, b\rangle$ where:*

- *z, the antecedent, is a partial assignment to the decision variables of an informed search problem and*
- *$b : P_z \rightarrow \mathbb{R}$, the consequent, is a function that maps partial assignments that are extensions of z to an upper bound on the utility for any extension to the partial assignment.*

A*BC applies the "learned bounding function" to the partial assignment of a search node when it is queued for expansion. Given a partial assignment, y, the bound of y is the minimum of the original bounding function (b_0) and the bounding functions of each triggered bounding conflict. A bounding conflict γ is triggered if partial assignment y is an extension of γ's partial assignment. That is if:

$$y \subseteq \gamma[z] \tag{2}$$

This leads to the following algorithm (Algorithm 1) used by A*BC to compute the bound on the utility of a search node.

In our experiments, we use CPLEX to both optimally schedule and learn bounding conflicts. A bounding conflict learning phase is entered when the utility of the optimal schedule differs from the bound provided by Algorithm 1 by more than a specified threshold. In our experiments, we chose a threshold of 10%.

```
Input: Partial assignment $y$, set of bounding conflicts
       $\Gamma$, original bounding function $b_0$
Output: Optimistic bound on the cost of $y$
1 return $\min(\{b_0(\mathrm{y})\} \cup \{\gamma_i[b](c) \mid \gamma_i \in \Gamma \wedge y \subseteq \gamma_i[z]\})$
```

Algorithm 1: ABC-BOUND

Intuitively, if the actual utility does not match the utility predicted by the bounding function, it means there are some active constraints in the problem that are squeezing the duration between some pairs of events that have a temporal constraint with preference relating them. To learn the bounding conflict, these constraints that are squeezing the constraints with preference must be found. This is accomplished by first re-solving the linear program with the objective modified so that the objective function contains only the preferences from the squeezed constraints with preference. Next, a constraint is added to the problem stating that the objective function must return a higher utility than the utility obtained from solving this smaller LP.

This added constraint causes the problem to become infeasible and a feasibility conflict is extracted that specifies which constraints are involved. The assignments to the decision variables that activate these constraints are then determined (z). This is the antecedent of the bounding conflict.

A new scheduling problem is then constructed, containing only events and constraints activated by these decisions. The difference between the optimal utility (r^*) provided by this subproblem is then compared to the bound provided by the original bounding function b_0 for this partial assignment. The function $b(x) = b_0(x) - (b_0(z) - r^*)$ becomes the consequent of the bounding conflict.

Generator

The generator is responsible for producing candidate assignments to the decision variables and querying the tester for the optimal schedule. The generator uses a new algorithm, called A* with Bounding Conflicts (A*BC) to enumerate candidates.

The A*BC algorithm operates on an informed search problem.

Definition 5. *An informed search problem is described by the tuple $\langle Y, D, r, b\rangle, where:$*

- *$Y = \{y_1, \ldots, y_m\}$ is a set of m decision variables with finite domains $D = \{d_1, \ldots, d_m\}$.*
- *$r : Y \rightarrow \mathbb{R}$ is the reward function. r maps full assignments to the decision variables to a real valued reward.*
- *$b : Y \rightarrow \mathbb{R}$ is the bounding function. b maps partial assignments to the decision variables to an upper bound on the reward of any full assignment containing that partial assignment.*

A*BC is an optimal, best first search and enumeration algorithm. Optimal best first search algorithms operate by ranking candidate search nodes with respect to a function b

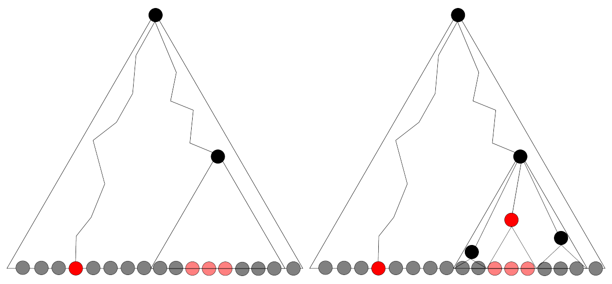
Figure 3: Conceptual image of A*BC.

that is an upper bound on the reward of any goal node reachable from that search node. So long as the bounding function is always optimistic, optimal best first search algorithms will find the optimal solutions.

A* uses a bounding function, in the form of an admissible heuristic, to generate candidate assignments to decision variables. When the exact reward is expensive to compute, the corresponding bounding function used is likely to be quite optimistic, as it represents an abstraction of the exact reward that is selected for speed of computation.

For example, in the underwater exploration example, determining the reward received by making a set of decisions requires determining an optimal schedule for all events. An upper bound on the reward can be quickly computed by assuming the constraints do not interact and by simply summing the maximum reward for each constraint that is activated by the set of decisions.

Due to the tendency of b to be overly optimistic in real world problems, a search algorithm such as A* may spend inordinate amounts of effort exploring a region of the search space that according to b is promising, but is of very low reward according to the actual reward function r. The main idea behind bounding conflict directed search is that a tighter bounding function can be learned for these regions based on experience.

As previously discussed, in A*BC, these tighter bounds are represented using bounding conflicts. Recall that a bounding conflict compactly represents a region of the search space and a tighter bounding function that is valid for that region. The learned bounding conflicts are then used during the search to proactively avoid exploring the regions covered by them until absolutely necessary. This is illustrated conceptually in Figure 3. In the search tree on the left, a bounding conflict has been learned that offers a tighter bound on the red, explored leaf node to the left, and three pink, unexplored leaf nodes sitting below a black, unexpanded node. In the figure on the right, the unexpanded node is expanded based on the bounding conflict. One child is introduced that contains the three pink leaf nodes of the bounding conflict; that child receives the tighter bound of the bounding conflict. The other children cover the remainder of the space; the bounding conflict bound does not apply to these children. In contrast, for a feasibility conflict, the child that contains the three pink leaf nodes of the conflict would be pruned.

A* with Bounding Conflicts

A*BC uses the bounding conflicts to focus search in a manner similar to Conflict-directed A*. This is accomplished by modifying the child expansion function, so that descendants in which a bounding conflict apply are pushed further down on the search queue, according to their learned bound.

A*BC builds on top of constraint-based A* (Williams and Ragno 2003). Constraint-based A* generates full assignments in best-first order, by expanding partial assignments through assigning unassigned variables, and by ordering search according to an optimistic bound over partial assignments. A*BC differs from constraint-based A*, in that it expands search nodes by splitting on unresolved conflicts, as well as unassigned variables. Given both options, A*BC chooses to split on *unresolved* conflicts first. A*BC splits on a conflict by adding assignments to the parent so that the children either *manifest* or *avoid* the conflict, while partitioning the parent's descendants into subtrees.

The process of splitting on a conflict is described later. The terms manifest, avoid and unresolved are defined as follows:

Definition 6. *Given partial assignment x and a conflict with partial assignment y:*

- *x manifests y if $y \subseteq x$,*
- *x avoids y if $x \cup y$ is inconsistent, and*
- *y is unresolved in x, otherwise.*

For example, given conflict assignment $y = \overline{V_C}$, then $\overline{V_C} V_A V_B$ manifests $\overline{V_C}$, $V_A V_C$ avoids $\overline{V_C}$, and $\overline{V_C}$ is unresolved in V_A.

```
Init: open ← {MAKE-ROOT-NODE()}
      $\Gamma \leftarrow \{\}$
1  $n \leftarrow \operatorname{argmax} n_i[f], n_i \in$ open;
2  open ← open $- n$;
3  if $n[z]$ is full assignment then
4      return n;
5  else
6      $\Gamma_n \leftarrow$ UNRESOLVED-CONFLICTS $(n[z], \Gamma)$;
7      if $\Gamma_n$ not empty then
8          children ← SPLIT-ON-CONFLICT($n, \Gamma_n$);
9      else
10         children ← SPLIT-ON-VARIABLE($n$);
11     end
12     foreach child $\in$ children do
13         if child$[z]$ is full assignment then
14             child$[f] \leftarrow c$(child[n]);
15         else
16             child$[f] \leftarrow$ ABC-BOUND(child$[z]$);
17         end
18         open ← open $\cup$ {child};
19     end
20 end
```
Algorithm 2: A* with Bounding Conflicts

Algorithm 2 describes the A* with Bounding Conflicts algorithm. In the remainder of this section, we explain the algorithm and demonstrate it on the AUV example problem.

A*BC starts off with an empty search queue. At this point, no bounding conflicts are known, so nodes popped off the search queue are expanded by choosing a variable unassigned in the node and by extending the search node's partial assignment to every possible assignment of the chosen variable. This is the SPLIT-ON-VARIABLE function. This process continues until the first full solution, $V_A V_B V_C V_D$, is found. The state of the search at this point is shown in Figure 4. The maximum reward attainable in a single region is 6 units, so the bounding function provides an optimistic bound of 24 for $V_A V_B V_C V_D$ and 18 for the remaining nodes.

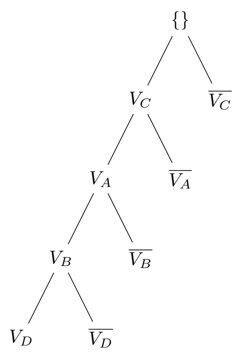

Figure 4: State of the search for the AUV example when the first full assignment to the decision variables is found.

At this point, the tester is invoked with the set of decisions corresponding to the AUV exploring all regions. The optimistic bounding function indicates a reward of 24, however the tester reports this only has a reward of 4, due to the amount of AUV travel between regions. In addition, analysis shows that any decision to have the AUV survey both regions A and B has a reward that is at least two units lower than the reward that the optimistic bounding function predicts. Hence the tester returns a bounding conflict with partial assignment $V_A V_B$, and a bounding function that decreases the original bounding function by 2.

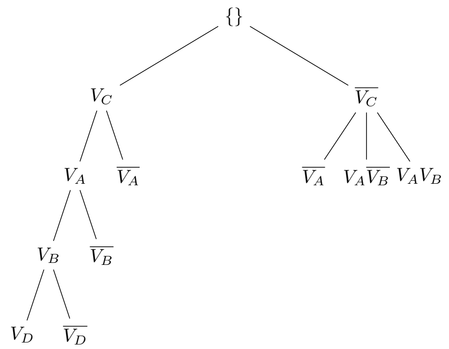

Figure 5: State of the search for the AUV example when the first bounding conflict is used to split. Each child of $\overline{V_C} V_A V_B$ includes the conflict's tighter bounding function.

At this point A*BC has a bounding conflict to process. Assume that A*BC next pops off the queue the node containing the partial assignment $\overline{V_C}$. At this point, the new conflict is unresolved by this node. Hence the SPLIT-ON-CONFLICT algorithm (Algorithm 4) is then used to generate new child nodes that either avoid or manifest the conflict, and places them on the queue. The child that manifests the conflict is generated by unioning the conflict's partial assignment with that of the parent node being expanded. The children that avoid the conflict are generated through the identical process to conflict-directed A* (see (Williams and Ragno 2003)). The children are generated such that they partition the space of all extensions to the parent's partial assignment. The state of the search at this point is shown in Figure 5. Two children are generated such that they avoid the conflict. The last child ($\overline{V_C} V_A V_B$) is generated such that it manifests the conflict. Any descendant of this last child $\overline{V_C} V_A V_B$ can use the tighter bound provided by the bounding conflict, which subtracts 2 from the heuristic.

This process continues until an optimal solution of $V_A \overline{V_B} V_C V_D$ is found.

```
   Input: Node to expand $n$, set of conflicts $\Gamma$
   Output: Exhaustive children of $n$
1  $\gamma \leftarrow$ choose conflict in $\Gamma$;
2  children $\leftarrow \{$MAKE-NODE$(n[z] \cup \{\gamma[z]\}, n)\}$;
3  temp $\leftarrow \{\}$;
4  foreach assignment $\in \gamma[z]$ do
5      $x \leftarrow$ variable of assignment;
6      $y \leftarrow$ value of assignment;
7      foreach $v \in (dom(x) - y)$ do
8          children $\leftarrow$ children $\cup$
           MAKE-NODE$(\{n[z] \cup$ temp $\cup \{x = v\}\}, n)$;
9      end
10     temp $\leftarrow$ temp $\cup$ assignment;
11 end
12 return children;
```
Algorithm 3: SPLIT-ON-CONFLICT

Experimental Results

To benchmark our CSTNP solver and the impact of bounding conflicts on solution time, we compared its performance to a solver with the same generate and test framework that used only feasibility conflicts to guide the search.Our test scenarios were randomly generated CSTNP instances with the same structure as the glider example in this paper. Parameters varied include the number of required uplinks, the number of regions that can be surveyed between uplinks, and the number of gliders being planned for in parallel.

The preference functions on the duration spent surveying each site are linear in the duration. The optimal scheduling problem was encoded as a linear program and solved using CPLEX. Additionally, CPLEX was used to learn bounding conflicts as described in previous sections. The results are summarized in Figure 6. The tests were run with a timeout of 30 seconds. With this timeout, the non-bounding conflict directed approach was able to solve only 63% of the test cases that A*BC was able to solve.

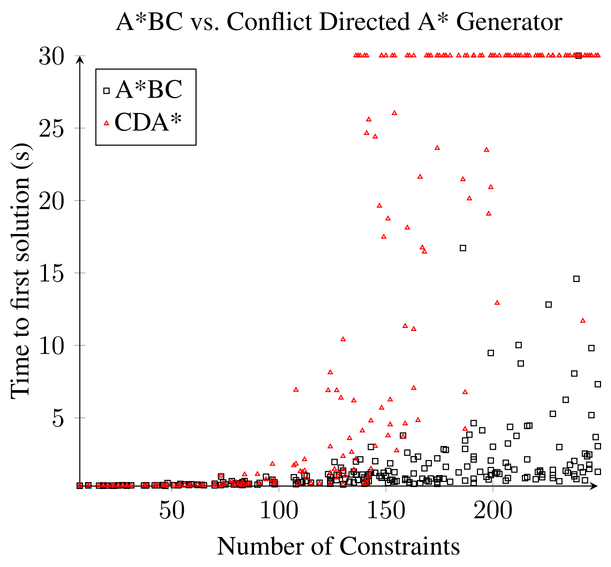

Figure 6: Performance of candidate generator with bounding conflicts (A*BC) vs without bounding conflicts (CDA*).

Conclusion

In this paper we introduced conditional simple temporal networks with preference (CSTNP) in order to represent real-world contingent planning and scheduling problems in which preference is a function of both discrete choice and schedule. In addition, we provided a solution method for enumerating the best solutions to the CSTNP, through best-first generate and test. Our key contribution underlying this method is a new algorithm, A* with bounding conflicts, A*BC, that uses bounding conflicts, learned when bounding a candidate solution, to learn an improved heuristic function, used to guide the candidate generator. A*BC offers a strict generalization of the conflict-directed A* algorithm. Experiments demonstrate substantive runtime gains over a feasibility conflict directed best-first search approach.

References

Dago, P., and Verfaillie, G. 1996. Nogood recording for valued constraint satisfaction problems. In *Tools with Artificial Intelligence, 1996., Proceedings Eighth IEEE International Conference on*, 132–139. IEEE.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49(1-3):61 – 95.

Effinger, R., and Williams, B. C. 2006. Extending dynamic backtracking to solve weighted conditional csps. In *Proceedings of the 21st National Conference on Artificial Intelligence*, 28–35.

Effinger, R. 2006. Optimal temporal planning at reactive time scales via dynamic backtracking branch and bound. Master's thesis, Massachusetts Institute of Technology.

Falda, M.; Rossi, F.; and Venable, K. 2007. Strong, weak, and dynamic consistency in fuzzy conditional temporal problems. In *Proceedings of COPLAS 2007, CP'07 workshop on planning and scheduling*.

Falda, M.; Rossi, F.; and Venable, K. B. 2010. Dynamic consistency of fuzzy conditional temporal problems. *Journal of Intelligent Manufacturing* 21(1):75–88.

Ginsberg, M. 1993. Dynamic backtracking. *Journal of Artificial Intelligence Research* 1:25–46.

Khatib, L.; Morris, P.; Morris, R.; Rossi, F.; et al. 2001. Temporal constraint reasoning with preferences. In *IJCAI*, 322–327.

Kim, P.; Williams, B. C.; and Abramson, M. 2001. Executing reactive, model-based programs through graph-based temporal planning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 487–493.

Morris, P.; Morris, R.; Khatib, L.; Ramakrishnan, S.; and Bachmann, A. 2004. Strategies for global optimization of temporal preferences. In *Principles and Practice of Constraint Programming–CP 2004*. Springer. 408–422.

Nau, D.; AU, T.; Ilghami, O.; Kuter, U.; Murdock, W.; Wu, D.; and Yaman, F. 2003. Shop2: An htn planning system. *Journal of Artificial Intelligence Research* 20:379–404.

Paterson, J.; Timmons, E.; and Williams, B. C. 2014. A scheduler for actions with iterated durations. In *AAAI-14*.

Peintner, B., and Pollack, M. E. 2004. Low-cost addition of preferences to dtps and tcsps. In *AAAI*, 723–728.

Robertson, P.; Effinger, R.; and Williams, B. C. 2006. Autonomous robust execution of complex robotic missions. In *Proceedings of the 9th International Conference on Intelligent Autonomous Systems*, 595–604.

Santana, P. H. R. Q., and Williams, B. C. 2013. Chance-constrained consistency for probabilistic temporal plan networks. In *International Conferences on Automated Planning and Scheduling*.

Shu, I.-h.; Effinger, R.; and Williams, B. C. 2005. Enabling fast flexible planning through incremental temporal reasoning with conflict extraction. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling*, 252–261.

Tate, A. 1976. Project planning using a hierarchic non-linear planner. Technical report, Department of Artificial Intelligence, University of Edinburgh. D.A.I. Research Report No. 25.

Tsamardinos, I.; Vidal, T.; Pollack, M.; and Tsamardinos, I. 2003. Ctp: A new constraint-based formalism for conditional, temporal planning.

Williams, B. C., and Ragno, R. 2003. Conflict-directed a* and its role in model-based embedded systems. *Special Issue on Theory and Applications of Satisfiability Testing, Journal of Discrete Applied Math* 155(12):1562–1595.

Yu, P., and Williams, B. C. 2013. Continuously relaxing over-constrained conditional temporal problems through generalized conflict learning and resolution. In *International Joint Conferences on Artificial Intelligence*.