A Scheduler for Actions with Iterated Durations

James Paterson, Eric Timmons and Brian C. Williams
Massachusetts Institute of Technology
Computer Science and Artificial Intelligence Laboratory
32 Vassar Street, Building 32-224, Cambridge, MA 02139
{paterson, etimmons, williams}@mit.edu

Abstract

A wide range of robotic missions contain actions that exhibit looping behavior. Examples of these actions include picking fruit in agriculture, pick-and-place tasks in manufacturing and search patterns in robotic search or survey missions. These looping actions often have a range of acceptable values for the number of loops and a preference function over them. For example, during robotic survey missions, the information gain is expected to increase with the number of loops in a search pattern. Since these looping actions also take time, which is typically bounded, there is a challenge of maximizing utility while respecting time constraints. In this paper, we introduce the Looping Temporal Problem with Preference (LTPP) as a simple parameterized extension of a simple temporal problem. In addition, we introduce a scheduling algorithm for LTPPs which leverages the structure of the problem to find the optimal solution efficiently. We show more than an order of magnitude improvement in run-time over current scheduling techniques and framing a LTPP as a MINLP.

Introduction

There is a growing need for autonomous robotic systems that can perform patterned search or mapping tasks. Unmanned Aerial Vehicles (UAVs) are being used to map agricultural land to provide farmers with crop estimates and locate areas of disease (Bryson et al. 2010), while Autonomous Underwater Vehicles (AUVs) do patterned search of the sea floor to search for hydrothermal vents and other interesting scientific events (Ferri, Jakuba, and Yoerger 2008). Search-and-rescue missions are another common area where UAVs perform patterned search (Goodrich et al. 2008). Aside from searching and mapping, there is a need for robots that can perform repetitive tasks, such as manufacturing pick-and-place tasks on an assembly line.

In all these tasks, timing and scheduling are crucial. For example, if Sentry (a Woods Hole Oceanographic Institute AUV) is mapping the ocean floor (Kinsey et al. 2011), it needs to ensure completion of tasks by designated time points to ensure the mission is completed before the battery runs out. Missions like these are often scheduled off-line, resulting in fixed schedules during execution time. However, temporal disturbances happen in any real-world mission and we would like systems that are robust these disturbances, degrading gracefully while preserving as much utility as possible rather than abruptly dropping entire tasks.

Previous work supports looping behaviors such as fixed search patterns during autonomous robotic scouting missions, but does not allow for the selection of parameters during scheduling (Bernardini et al. 2013). Elsewhere, execution algorithms have been explored that automatically perform scheduling, resource assignment and contingency selection, by making decisions on-line (Conrad and Williams 2011; Shah, Conrad, and Williams 2009; Morris and Muscettola 2005). However, these methods have not addressed looping actions. Scheduling problems such as Temporal Constraint Satisfaction Problems with Preference (TCSPPs) (Dechter, Meiri, and Pearl 1991) allow disjunctive temporal constraints and preference functions on time and disjunctive bounds, but they do not allow representation of looping ranges, or specifying preference over the number of loops without specifying a disjunctive bound and a utility value for each loop. Work in the mathematical optimization community on Mixed-Integer Non-linear Programs (MINLPs), allow maximization of a preference function, while satisfying real and integer constraints. While optimal scheduling problems can be expressed as a MINLP program, it is often unintuitive to do so.

In this paper, we present the Looping Temporal Problems with Preference (LTPP) as a means to handle loops in a temporal network, by a parametric extension to STPs (Dechter, Meiri, and Pearl 1991), combined with a preference over the number of loops. We also present a scheduler for the LTPP that produces an optimal schedule containing the number of loops to perform within each looping action and the times of when to perform them. This scheduler draws from techniques used to solve STPs, as well as Branch and Bound techniques from MINLP problems. While this work focuses on off-line scheduling of LTPPs, the resulting schedule may be dispatched on-line (Conrad and Williams 2011), providing some robustness to temporal disturbances. Finally, we present empirical results to show an improvement in run-time of more than an order magnitude over current schedulers and framing a LTPP as a MINLP.

Autonomous Scouts for Search and Rescue

Using aerial vehicles to gather information for search-and-rescue (Goodrich et al. 2008) or search-and-tracking (Bernardini et al. 2013) missions is an area of active research. Many of the search strategies employed in these missions may be modelled with looping actions and this work provides a way to autonomously choose the number of loops, while adhering to temporal constraints. As an illustrative example of a scouting mission with looping actions and to help explain our algorithm for solving LTPPs, we introduce an example from the search-and-rescue domain.

Consider the following example, where two hikers have not reached their destination on time. From the topology of the area and the route they were expected to take, they are most likely to be in Area A, Area B, or on the path between the two areas (See Figure 1). In order to locate the hikers, a UAV is sent out to perform a search. Suppose that a higher-level planner has, based on the geometry of the search areas, decided to use a lawnmower search pattern in Area A, a star search pattern in Area B, and path-following on the path between the two areas. The lawnmower pattern can be represented as a looping action with each pass representing a loop, while the star pattern is a looping action with each triangle as a loop.

Given the dynamics of the UAV and the size of the areas, each loop in lawnmower pattern can be performed in a controllable duration between 2 and 5 minutes and each loop in the star pattern between 3 and 4 minutes. The length of the path between Area A and B can be flown in 3 to 7 minutes. Given flight altitude restrictions and the sensor's field-of-view on board, the system determines that the UAV has to perform between 5 and 20 loops in Area A, and at least 5 loops in Area B. Based on prior beliefs and expected information gain, there is also a preference over the number of loops to perform in each area. These are $f_A(N_A) = 10\log(N_A)$ for the lawnmower pattern and $f_B(N_B) = 2N_B$ for the star pattern, where N denotes the number of loops. Since the number of loops cannot be fractional, these functions are discrete. Finally, information is required from Area A and the path within 35 minutes, and the UAV has a maximum flight time of 50 minutes. Figure 1 summarizes this information.

This problem can be encoded using a LTPP. Looping actions, constrained by time, are represented by looping temporal constraints (LTCs) in the LTPP. Path-following and time constraints such as the overall mission duration and reaching Area B before 35 minutes can also be represented as a looping action with only one loop. The preference functions map the numbers of loops in each LTC to a utility value and a global preference function calculated the overall utility of the mission.

The optimal solution to this problem would be to perform 7 loops in Area A and 11 loops in Area B.

Problem Statement

We define a new class of problem called the Looping Temporal Problem with Preference (LTPP) as an encoding of a problem containing temporal actions with loops. Our encoding contains the minimal variables and constraints that are sufficient to determine the number of loops and durations. Using our encoding, a looping program can be mapped to an LTPP in a straight forward manner.

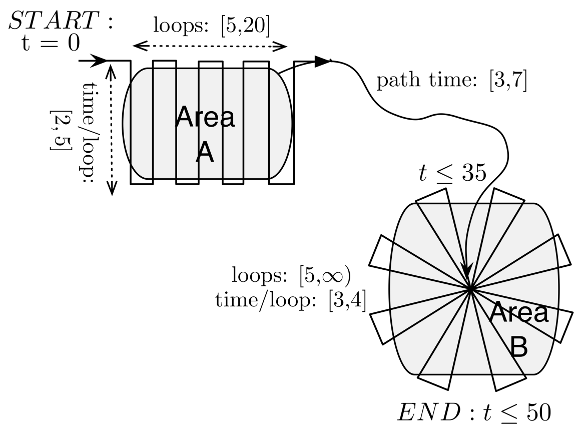

Figure 1: Search for two missing hikers.

Definition 1: A *LTPP* is a triple $< X, C, F >$ where:

- X is a set of events representing points in time.
- C is a set of looping temporal constraints (LTCs) between pairs of events as defined in Definition 2.
- F is a function that maps local utility values from the LTCs to a global utility value.

Definition 2: A *looping temporal constraint* (C_{ij}) is a repeated simple temporal constraint between two time events, x_i and x_j, and is specified by a tuple $< N_{ij}, \delta_{ij}, f_{ij}(N_{ij}) >$ where:

- δ_{ij} is a simple temporal constraint between the start and end times of *one iteration* and is of the form $\delta_{ij} \in [\delta_{ijl}, \delta_{iju}]$, $\delta_{ij} \in \mathbb{R}^+$.
- N_{ij} is an integer number of loops between the events x_i and x_j and is constrained in the range: $N_{ij} \in [N_{ijl}, N_{iju}]$, $N_{ij} \in \mathbb{Z}^+$.
- $f_{ij}(N_{ij})$ is a monotonic preference function that maps the number of loops N_{ij} to a utility value.

Since the choice of number of loops affects the temporal duration per loop in the final schedule, we specify preference over the number of loops only and not the temporal durations. This can be interpreted as a preference level for each loop duration.

The solution method proposed in this paper restricts the global preference function F to a monotonic function (that may be non-convex), i.e. when given any range over the number of loops, the preferred solution to each loop variable is always at one extreme of its range. In the experimentation section, we focus on one class of preference functions that exhibits this property. In this class of preferences, the local functions are monotonic and produce positive, real numbers. The global preference function then combines the local preferences using any combination of the $+$ and $\times$ operators.

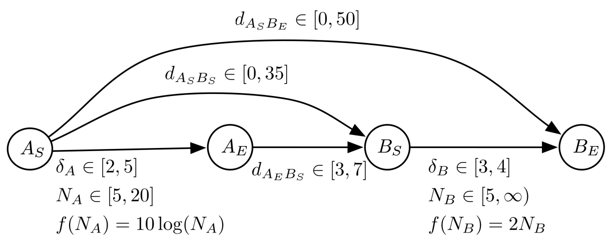

Figure 2: The search-and-rescue mission framed as a LTPP.

A simple temporal constraint is a special case of a looping temporal constraint, where the number of loops is 1. As these can be easily extracted from the set of all looping temporal constraints C, we will refer the set of simple temporal constraints as $STC \subseteq C$, with temporal bounds $d_{ij} \in [d_{ijl}, d_{iju}]$, $d_{ij} \in \mathbb{R}^+$ and the set of remaining looping temporal constraints as $LTC \subseteq C$.
Given these definitions, the search-and-rescue example is framed as a LTPP in Figure 2. The global preference function F combines the two local preference functions using the $+$ operator, i.e. $F = (10\log(N_A) + 2N_B)$.

Note that loops are atomic, and the system can not arbitrarily exit a loop in its middle. Consequently, the loop variables are restricted to be integer and thus a looping temporal constraint (LTC_{ij}) between two time events x_i and x_j can not simply be written as $x_j - x_i \in [N_{lij} \times \delta_{lij}, N_{uij} \times \delta_{uij}]$, since the integer constraint on N_{ij} causes a disjunctive temporal bound between x_i and x_j. For example the LTC:

$\{N_{ij} \in [1, 2]; \delta_{ij} \in [3, 4]\} \equiv \{[3, 4] \vee [6, 8]\}$
has a temporal gap between 4 and 6.

An *optimal solution* to a LTPP is an integer assignment to each LTC looping variable *N*, that maximizes the global preference function, while satisfying all temporal constraints.

Related Work

Candidate solution methods exist in the scheduling and mathematical optimization communities for solving a Looping Temporal Problem with Preference (LTPP).

First, for loops with finite bounds, the loops in a LTPP can be encoded in a Temporal Constraint Satisfaction Problem with Preference (TCSPP) (Peintner and Pollack 2004) or a Disjunctive Temporal Problem with Preference (DTPP) (Peintner and Pollack 2004; Stergiou and Koubarakis 2000), where each loop forms a disjunctive temporal bound between two events and the preference function over the number loops can be split into one function for each of the disjunctive bounds. For example, the lawnmower pattern in Area A of the search-and-rescue example could be framed as in Figure 3. However, a LTPP is inefficient to solve in the TCSPP framework as TCSPP solvers break the disjunctive constraints into component STPs to solve. Compared to a TCSPP or DTPP, the LTPP also allows a clean way of specifying looping ranges such as "anything more than", by setting infinity as an upper bound (such as the pattern in Area B of the original search-and-rescue mission example).

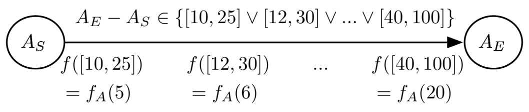

Figure 3: The TCSPP representation of a looping action.

The LTPP contains a mixture of integer and real-valued constraints, as well as a maximization over non-linear, non-convex functions. Thus, the second candidate solution method is to frame the problem as a Mixed-Integer Nonlinear Program (MINLP). MINLPs are among the class of theoretically difficult problems that are NP-complete. Solution methods for solving MINLPs include Outer Approximation methods (Abhishek, Leyffer, and Linderoth 2010; Fletcher and Leyffer 1994), Branch-and-Bound (B&B) (Belotti et al. 2009; Quesada and Grossmann 1992), Generalized Benders Decomposition (Geoffrion 1972) and Extended Cutting Plane methods (Westerlund et al. 1998). These approaches generally rely on the successive solutions of closely related Non-linear Program (NLP) problems. In particular, B&B starts out forming a pure continuous NLP problem by relaxing the integer constraints on the discrete variables. The LTPP can compiled to a MINLP as in Figure 4.

$$\begin{array}{ll}
maximize: & subject\ to: \\
F(\{N_{ij}\}) & N_{ij}\delta_{ijl} \leq x_j - x_i \leq N_{ij}\delta_{iju} \\
\forall\, i, j & N_{ijl} \leq N_{ij} \leq N_{iju} \\
 & N_{ij} \in \mathbb{Z}^+ \\
 & \forall\, i, j
\end{array}$$

Figure 4: The MINLP encoding of a LTPP.

Note that although the MINLP encoding in Figure 4 will find the optimal values for the loop ranges, it will not provide bounds for executing the temporal events as a scheduling technique would.

Approach

Drawing from techniques used by TCSPP, DTPP and B&B MINLP solvers, our approach solves the LTPP by casting the original problem as a series of STPs, relaxing the integrality requirements at each step and checking consistency. Where our approach differs from current scheduling techniques, is that is can handle infinite loops, by employing a domain filtering technique to prune loop ranges and then it searches through the remaining state-space by checking consistency of STPs that encompass *ranges* of loops rather than a single loop value combination. The consistency checks are performed incrementally and a tight heuristic is used to minimise changes between consistency checks.

Domain Filtering

The DOMAINFILTER algorithm (Algorithm 1) reduces the solution state-space by pruning out loop ranges that can never form part of a solution. In the example presented earlier, one of the loop ranges can take on a value between 5 and ∞. However, an infinite number of loops cannot form part of the solution as the mission is constrained to take less than 50 minutes. Reducing the loop ranges to a space of possible solutions greatly reduces the search space that the BOUNDSEARCH algorithm (Algorithm 2) has to search through.

DOMAINFILTER extends the all-pairs shortest path method to readily prune loops in the LTPP at each iteration. For simplicity, the pseudo code for the extension to the Floyd-Warshall algorithm (Cormen et al. 2001) is shown in Algorithm 1, but the implementation for benchmarking uses an extension to Johnson's algorithm (Cormen et al. 2001).

Algorithm 1: DOMAINFILTER

```
Input: A LTPP $< X, C, F >$.
Output: A LTPP, with loop ranges tightened to remove
        inconsistent solutions.
Algorithm
1  $STP \leftarrow makeSTP(LTPP)$
2  for $x_k \in X$ do
3      for $x_i \in X$ do
4          for $x_j \in X$ do
5              if $(d(x_i, x_k) + d(x_k, x_j)) < d(x_i, x_j)$ then
6                  $d(x_i, x_j) \leftarrow (d(x_i, x_k) + d(x_k, x_j))$
7                  $C_{ij} \leftarrow pruneLoops(C_{ij}, d(x_i, x_j))$
8                  $d(x_i, x_j) \leftarrow reTighten(d(x_i, x_j), C_{ij})$
9  for $x_i \in X$ do
10     if $d(x_i, x_i) < 0$ then
11         return $nil$
12 return $LTPP$
```

DOMAINFILTER begins by forming a STP from a LTPP (line 1), by relaxing the integer constraint on the loop ranges, thus forming a STC from each LTC as follows:

$$LTC : N \in [N_l, N_u]; \delta \in [\delta_l, \delta_u], \quad N \in \mathbb{Z}^+, \delta \in \mathbb{R}^+$$
$$\Rightarrow\ STC : d \in [N_l \times \delta_l, N_u \times \delta_u], \quad d \in \mathbb{R}^+$$

If the STC between event x_1 and event x_2 has the bound $d \in [d_l, d_u]$, then $d(x_1, x_2)$ is the distance edge from x_1 to x_2 (equal to d_u) and $d(x_2, x_1)$ is the distance edge from x_2 to x_1 (equal to $-d_l$).

The algorithm runs exactly the same as the Floyd-Warshall algorithm until line 7. At this point, if the algorithm tightens a bound on the STC (line 6), the loop range of the corresponding LTC may be tightened too as any loop number whose corresponding temporal bound falls outside the tightened STC bound is infeasible and can be pruned. This is done by the $pruneLoops()$ function in line 7. The $pruneLoops()$ function accepts the tightened STC bound $d(x_i, x_j)$ and the corresponding LTC, C_{ij} with $N \in [N_l, N_u]$ and $\delta \in [\delta_l, \delta_u]$. If $d(x_i, x_j)$ is a lower-bound in the STC, it tightens loop ranges as in the left equation below, and if $d(x_i, x_j)$ is an upper-bound, it tightens as in the right equation:

$$N_l \geq ceil(|\tfrac{d(x_i,x_j)}{\delta_u}|), \quad N_u \leq floor(|\tfrac{d(x_i,x_j)}{\delta_l}|).$$

The $floor()$ and $ceil()$ operators ensure the loop range bounds remain integer, but since these operator further tighten the loop range bounds, the relaxed STC bound $d(x_i, x_j)$ needs to be tightened once more. This is done by $reTighten()$ (line 8) by tightening $d(x_i, x_j)$ according to the left equation if $d(x_i, x_j)$ is a lower-bound or to the right equation if $d(x_i, x_j)$ is an upper-bound:

$$min(d(x_i, x_j), N_u \times \delta_l), \quad min(d(x_i, x_j), N_l \times \delta_u)$$

The loop pruning process may be more clearly understood by noting that since δ is a deterministic range, the largest number of loops that would fall *within* the tightened temporal bound $d \in [d_l, d_u]$, would be when $N \times \delta_l \leq d_u$, and the smallest number of loops *within* the range d would be $N \times \delta_u \geq d_l$.

If the STP contains a negative cycle (line 10), we return nil (line 10) as it is inconsistent and thus the original LTPP is also inconsistent as each STC temporal range encompassing the entire corresponding LTC temporal range.

The run-time complexity of the Johnson's Algorithm version of DOMAINFILTER is $O(X^2 \log C + XC)$.

Although the network of the relaxed STP corresponding to the LTPP returned by DOMAINFILTER is consistent, no solution to the LTPP is guaranteed. The relaxation of the integer constraint on the loop variables to form a STP from a LTPP encompasses all looping temporal ranges, but also adds temporal ranges that lie between the loop temporal ranges and that don't exist in the original LTPP. Figure 5 shows an example of an inconsistent solution after running DOMAINFILTER, where the range of loops for each LTC is tightened from [1, 5] to [1, 2]. Although the network of the relaxed STP is consistent and the loops ranges have been pruned, no solution exists to the LTPP, since there is a requirement of 4.5 for the overall duration, whereas the duration of every looping constraint needs to be an integer of exactly 1 or 2 in this case.

Running DOMAINFILTER on the search-and-rescue example, results in a large reduction domain of looping ranges. The loop range of the lawnmower pattern is reduced from [5, 20] to [5, 16] and the loop range of the star pattern is reduced from [5, ∞) to [5, 12].

Searching for the Optimal Solution

After running DOMAINFILTER to reduce the search-space, BOUNDSEARCH (Algorithm 2) takes a best-first approach to search through the remaining space of loop ranges for an optimal, integer assignment to the looping variables, given the global preference function. It explores the search space by *incrementally* checking consistency over *ranges* of possible loop combinations, pruning large spaces of inconsistent combinations. When loop range combinations are consistent, our algorithm splits them into narrower combina-

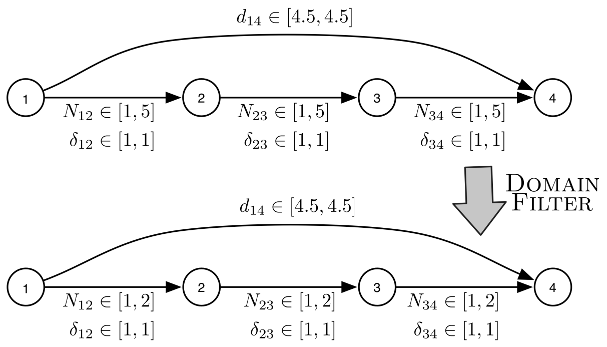

Figure 5: After running DOMAINFILTER, no solution is guaranteed.

tions until an integer solution is found. It uses an admissible heuristic to guide the search and expands the search tree in best-first order. Thus, the first consistent, integer assignment to the LTPP loop ranges is guaranteed to be the optimal solution.

The search tree for the first three iterations of the algorithm running on the search-and-rescue problem (Figure 6) is used to help explain BOUNDSEARCH.

BOUNDSEARCH begins with an initial *candidate* on the queue. Each *candidate*, D is a node in the search tree and is a tuple $< L, H, P, C >$, where L is a list of loop ranges to check consistency over, H is the maximum utility obtainable from that combination of loop ranges (used as the admissible heuristic), P is the parent of D, and C is the LTC that was split to produce D from P. The initial candidate ($D1$) contains the entire search-space.

Since the global preference function F is monotonically increasing, the maximum utility H for any candidate is found by simply evaluating F at the maximum loop range for each $L_i \in L$. In the example (Figure 6), $H_{D1} = f_A(16) + f_B(12) = 52$.

The Incremental Consistency Checker ITC is initialized (line 4) with a STP that is formed from the LTPP by relaxing all integer constraints on the looping variables (line 3). For the Rescue Mission example, ITC would be initialized to the STP corresponding to $D1$ in Figure 7.

Then, within each iteration, BOUNDSEARCH removes the best candidate D from the queue (according to H) (line 6). If D is found to be consistent, it is split into two children candidates that each contain a subset of the state-space of possible loop combinations (line 11). In Figure 6, $D1$ is found to be consistent, and is split into two candidates, $D2$ and $D7$. Only one loop interval is split to ensure the two children encapsulate all the possible loop combinations of the parent. The LTC loop interval that adds the most utility to the overall preference function is chosen as the one to split, in order to guide the search to the optimal solution faster.

In the process of splitting from a parent candidate, only one constraint C is modified to form each child (see Figure 7) and for each child, this constraint is saved (lines 11-13). Because a tight heuristic is used, we know that one of the children will be the next-best candidate on the queue

Algorithm 2: BOUNDSEARCH

```
Input: A LTPP with a LTC loop ranges pruned by
       DOMAINFILTER.
Output: A LTPP schedule, with integer assignments to
        each LTC loop range.
Initialization:
1  $D \leftarrow makeCandidate(LTPP)$
2  $priorityQ \leftarrow \{D\}$
3  $STP \leftarrow makeSTP(LTPP)$
4  $ITC \leftarrow initializeITC(STP)$
Algorithm:
5  while $priorityQ \neq \emptyset$ do
6      $D \leftarrow pop(priorityQ)$
7      $C \leftarrow getModifiedConstraint(D)$
8      if consistentITC?(ITC, C) then
9          if integerAssignment?(D) then
10             return $makeSolution(LTPP,\ D)$
           else
11             $\{children, C\} \leftarrow split(D)$
12             for child in children do
13                 $setModifiedConstraint(child,\ C)$
14                 $setParent(child,\ D)$
15             $addToQ(priorityQ,\ \{children\})$
       else
16         $P \leftarrow parent(peek\ (priorityQ))$
17         resetITC(P)
18 return $nil$
```

and thus when it is popped off the queue, we check consistency incrementally by simply modifying ITC with the single constraint C.

The algorithm continues splitting candidates and narrowing the possible loop combinations of each candidate. If an inconsistent candidate is found ($D3$), it is pruned, removing all its possible loop combinations from the search space. At this stage ITC is in an inconsistent state and it is reset to the consistent state it was in for the parent ($D2$) of the next-best candidate on the queue ($D4$) (line 17). Then, at the next iteration of the algorithm, consistency is checked by simply modifying ITC again with the single constraint C that was altered when $D4$ was formed from $D2$.

Figure 7 shows how only one constraint is modified while splitting a parent candidate into two children.

If BOUNDSEARCH keeps splitting loop ranges until an integer loop combination is found (i.e. the lower and upper bound of each loop range is the same), it assigns the candidate loop values to the LTPP and runs Johnson's algorithm to calculate the bounds on the temporal events X, before returning it as the optimal schedule (line 10). If BOUNDSEARCH loops until the queue is empty, no solution is possible to the LTPP and nil is returned (line 18).

Empirical Validation

To validate our approach, we compare BOUNDSEARCH to a TCSPP encoding using a best-first component STP search

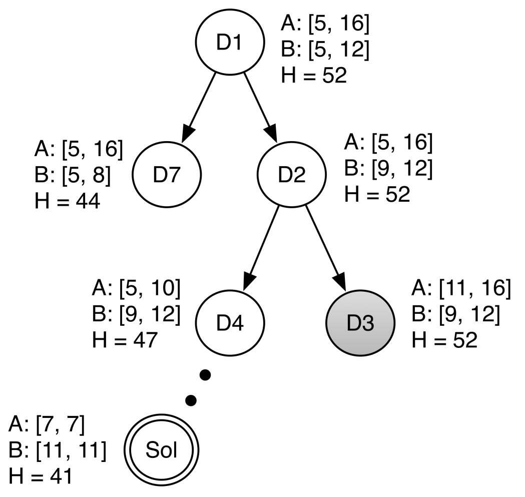

Figure 6: The search tree after three iterations of the search-and-rescue problem. Looping ranges are marked A and B to denote the two search areas. Candidates are labeled in the order they are checked for consistency and the solution is shown as a double circle node.

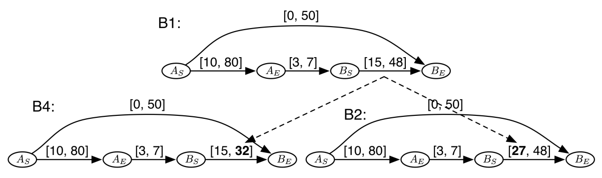

Figure 7: STP representation of candidates after the first split. The loop range that adds the most to the overall utility is [5, 12] and it is split to [5, 8] and [9, 12]. Since each loop has a temporal duration of $\delta_B \in [3, 4]$, STCs of [5 × 3, 8 × 4] and [9 × 3, 12 × 4] are formed.

(CompSTP), as well as to a MINLP encoding using SCiP as a solver (Achterberg 2009). CompSTP is similar to the way a TCSPP would be solved on this problem instance using preference levels (Peintner and Pollack 2004). For BoundSearch and CompSTP, the DomainFilter algorithm was run before the search process.

Setup

We benchmarked the above methods on LTPPs modelling single vehicle missions, with looping actions interleaved with actions that move between locations or perform some task. The missions are constrained by an overall temporal constraint between the start and end events, as well as other temporal constraints on events in the mission.

Since the difficulty of the problem increases with the size of the state-space, we test performance as we increase the number of looping actions (LTCs). The LTCs are each of the form: $\{N \in [5, 20]; \delta \in [5, 10]; f(N) = a.N\}$, where a is a random number between 0 and 5. The overall temporal constraint was generated randomly to ensure the solution is varied throughout the state space of possible solutions over multiple runs. Simple temporal constraints (STCs) denoting non-looping activities were randomly interleaved between event pairs with probability 0.2. A non-linear global preference function was created by generating a random expression tree, combining pairs expressions using $+$ and $\times$ operators with equal probability.

Each data point represents the medians of the data for 100 runs. If the median was above the cut-off time of 20 minutes, that data point was not plotted.

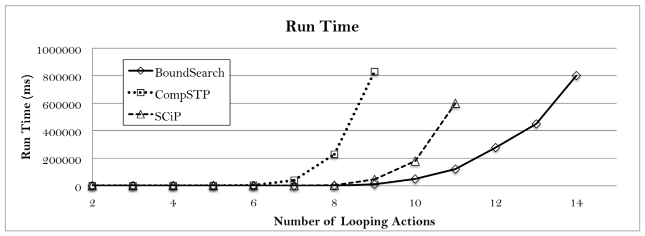

Figure 8: Run Time comparisons between CompSTP, SCiP and BoundSearch.

Results

Since BoundSearch checks consistency over ranges of loops and prunes large parts of the state-space, it shows multiple orders of magnitude improvement in run-time over CompSTP above 7 looping constraints (Figure 8). We also show an order of magnitude improvement in run-time over SCiP at 11 looping constraints and above. This is largely due to efficient constraint propagation to reduce the state space during domain-filtering and tailoring our algorithm to use a monotonic global preference function. It must be mentioned that SCiP can handle non-monotonic functions, which Bound search does not in its current capacity.

Although empirical results suggest exponential growth in run-time of BoundSearch as looping actions increase, BoundSearch find a solution extremely quickly (under 1 second) for problems containing 7 looping constraints or less, allowing it to be used in mobile, online robotic missions of this size.

Contributions

In this paper we presented the Looping Temporal Problem with Preference (LTPP), as a formalism for framing scheduling problems that contain looping actions and preferences over them. We then presented a scheduling algorithm that finds the number of loops for each looping action to maximize a global preference function of the looping variables. Finally, we showed how our algorithm leverages the structure in the LTPP to improve run-time over best-first component STP search and MINLP solvers.

References

Abhishek, K.; Leyffer, S.; and Linderoth, J. 2010. Filmint: An outer approximation-based solver for convex mixed-integer nonlinear programs. *INFORMS Journal on computing* 22(4):555–567.

Achterberg, T. 2009. Scip: solving constraint integer programs. *Mathematical Programming Computation* 1(1):1–41.

Belotti, P.; Lee, J.; Liberti, L.; Margot, F.; and Wächter, A. 2009. Branching and bounds tighteningtechniques for non-convex minlp. *Optimization Methods and Software* 24(4-5):597–634.

Bernardini, S.; Fox, M.; Long, D.; and Bookless, J. 2013. Autonomous search and tracking via temporal planning. In *Twenty-Third International Conference on Automated Planning and Scheduling*.

Bryson, M.; Reid, A.; Ramos, F.; and Sukkarieh, S. 2010. Airborne vision-based mapping and classification of large farmland environments. *Journal of Field Robotics* 27(5):632–655.

Conrad, P. R., and Williams, B. C. 2011. Drake: An efficient executive for temporal plans with choice. *Journal of Artificial Intelligence Research* 42(1):607–659.

Cormen, T.; Leiserson, C.; Rivest, R.; and Stein, C. 2001. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2nd edition. chapter 25.3, "Johnson's algorithm for sparse graphs", 636–640.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal Constraint Networks. *Artificial Intelligence* 49:61–95.

Ferri, G.; Jakuba, M. V.; and Yoerger, D. R. 2008. A novel method for hydrothermal vents prospecting using an autonomous underwater robot. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, 1055–1060. IEEE.

Fletcher, R., and Leyffer, S. 1994. Solving mixed integer nonlinear programs by outer approximation. *Mathematical programming* 66(1-3):327–349.

Geoffrion, A. M. 1972. Generalized benders decomposition. *Journal of optimization theory and applications* 10(4):237–260.

Goodrich, M. A.; Morse, B. S.; Gerhardt, D.; Cooper, J. L.; Quigley, M.; Adams, J. A.; and Humphrey, C. 2008. Supporting wilderness search and rescue using a camera-equipped mini uav. *Journal of Field Robotics* 25(1-2):89–110.

Kinsey, J.; Yoerger, D.; Jakuba, M.; Camilli, R.; Fisher, C.; and German, C. 2011. Assessing the deepwater horizon oil spill with the sentry autonomous underwater vehicle. In *The 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 261–267.

Morris, P. H., and Muscettola, N. 2005. Temporal dynamic controllability revisited. In *AAAI*, 1193–1198.

Peintner, B., and Pollack, M. 2004. Low-Cost Addition of Preferences to DTPs and TCSPs. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence.* Menlo Park, CA: AAAI Press. 723–728.

Quesada, I., and Grossmann, I. E. 1992. An lp/nlp based branch and bound algorithm for convex minlp optimization problems. *Computers and Chemical Engineering* 16(10):937–947.

Shah, J. A.; Conrad, P. R.; and Williams, B. C. 2009. Fast distributed multi-agent plan execution with dynamic task assignment and scheduling. In *ICAPS*.

Stergiou, K., and Koubarakis, M. 2000. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence* 120:81–117.

Westerlund, T.; Skrifvars, H.; Harjunkoski, I.; and Pörn, R. 1998. An extended cutting plane method for a class of non-convex minlp problems. *Computers and Chemical Engineering* 22(3):357–365.