Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)

Managing Communication Costs under Temporal Uncertainty

Nikhil Bhargava, Christian Muise, Tiago Vaquero, Brian Williams
Massachusetts Institute of Technology
{nkb, cjmuise, tvaquero, williams}@mit.edu

Abstract

In multi-agent temporal planning, individual agents cannot know a priori when other agents will execute their actions and so treat those actions as uncertain. Only when others communicate the results of their actions is that uncertainty resolved. If a full communication protocol is specified ahead of time, then *delay controllability* can be used to assess the feasibility of the temporal plan. However, agents often have flexibility in choosing when to communicate the results of their action. In this paper, we address the question of how to choose communication protocols that guarantee the feasibility of the original temporal plan subject to some cost associated with that communication. To do so, we introduce a means of extracting *delay controllability conflicts* and show how we can use these conflicts to more efficiently guide our search. We then present three conflict-directed search algorithms and explore the theoretical and empirical trade-offs between the different approaches.

1 Introduction

In multi-agent temporal plans, individual agents are uncertain about the behavior of other agents and must communicate to resolve that ambiguity. While these agents might have the capability to promptly deliver updates, there may be reasons why delivering immediate updates is expensive. For example, an autonomous underwater vehicle may have the equipment on-board to immediately transfer information to any other agent, but sending it may use precious battery power needed for the rest of its mission. It may make sense to use a slower and less expensive means of communicating about its actions or to omit relaying information altogether.

In this paper, we consider the problem of how to construct a minimum-cost communication protocol that still guarantees the feasibility of the original plan. This protocol is chosen subject to a supplied cost function which characterizes how expensive it is to communicate at certain moments in time. To verify whether any particular communication protocol is feasible, we can evaluate the protocol in the context of a temporal network using delay controllability [Bhargava *et al.*, 2018], but there does not yet exist an efficient way to generate a protocol with low cost. Our paper focuses on this problem and offers two main contributions.

First, we demonstrate how to efficiently prune the available search space by providing an algorithm for deriving delay controllability conflicts. We augment existing delay controllability detection algorithms to output a conflict whenever a temporal plan is uncontrollable due to overly delayed communication. Adding this capability is essential, as it makes our search over an otherwise continuous and unbounded space tractable through the use of conflict-directed search [Williams and Ragno, 2007]. Our work on delay controllability conflict extraction matches the best-known $O(n^3)$ runtime of dynamic controllability conflict extraction techniques [Bhargava *et al.*, 2017].

Second, we explore three different variants of conflict-directed search, two sub-optimal and one optimal, analyzing their performance and guarantees. We show that for certain networks the sub-optimal algorithms can provide at best a polynomial approximation of the true optimal cost, but that in practice they provide a reasonable approximation while executing much faster than optimal search.

It is also possible to interpret this problem as a temporal plan relaxation problem where delays associated with observation of uncertain events undergo relaxation. Under such a view, it is natural to ask whether it is possible to combine delay relaxation with temporal constraint relaxation techniques [Fang *et al.*, 2014; Yu *et al.*, 2014; Cui *et al.*, 2015]. While we believe it is straightforward to unify the two concepts and their corresponding algorithms, such work is outside the scope of this paper.

2 Background

Simple Temporal Networks with Uncertainty (STNU) provide a way to reason about events that are outside an agent's control through the encoding of uncertainty [Vidal and Fargier, 1999]. In STNUs, events whose times can be assigned by the acting agent are called *activated timepoints* and events whose time values cannot be assigned are called *received timepoints*. Temporal constraints restrict the amount of time that can elapse between two events (e.g. event A must happen no more than 20 minutes after event B) and are also split into those that must be satisfied by the acting agent, *free constraints*, and those that are guaranteed to be satisfied by some assignment of value to a received timepoint, *contingent*

Edge Generation Rules

	Input	Conditions	Output
No-Case	$A \xrightarrow{u} B$, $B \xrightarrow{v} C$	N/A	$A \xrightarrow{u+v} C$
Upper-Case	$A \xrightarrow{u} D$, $D \xrightarrow{C:v} B$	N/A	$A \xrightarrow{C:u+v} B$
Lower-Case	$A \xrightarrow{c:x} C$, $C \xrightarrow{w} D$	$w < \gamma(C)$, $C \neq D$	$A \xrightarrow{x+w} D$
Cross-Case	$A \xrightarrow{c:x} C$, $C \xrightarrow{B:w} D$	$w < \gamma(C)$, $B \neq C \neq D$	$A \xrightarrow{B:x+w} D$
Label Removal	$B \xrightarrow{C:u} A$, $A \xRightarrow{[x,y]} C$	$u > -x$	$B \xrightarrow{u} A$

Table 1: Edge generation rules for a labeled distance graph

constraints. All contingent constraints relate some activated timepoint to some received timepoint that follows it, and there is exactly one contingent constraint per received timepoint.

To determine whether there exists a guarantee that we can satisfy all constraints of an STNU, we examine its *controllability*. In this paper we focus in particular on *delay controllability* [Bhargava *et al.*, 2018]. Delay controllability considers whether a just-in-time assignment of values to timepoints exists if for every received timepoint x_e, the agent learns about the true time of the received timepoint after $\gamma(x_e)$ additional time has passed, where γ is the associated delay function. Delay controllability of an STNU can be computed efficiently in worst-case $O(n^3)$ time [Bhargava *et al.*, 2018]. Delay controllability allows us to model disjointed communication that is often representative of multi-agent plan execution; if an external agent is delayed in communicating the results of some action, then for that particular action we assign $\gamma(x_e) = t$.

When analyzing STNUs and their controllability, we often find it easier to consider their *labeled distance graph* representation [Morris, 2006; 2014]. Each timepoint of the original STNU corresponds to a node in the graph, and for each temporal constraint of the form $l \leq x_B - x_A \leq u$ (or $A \xrightarrow{[l,u]} B$), we add two edges, $A \xrightarrow{u} B$ and $B \xrightarrow{-l} A$. For each contingent link of the form $A \xRightarrow{[l,u]} C$, we also add two additional labeled edges of the form $A \xrightarrow{c:l} C$ and $C \xrightarrow{C:-u} A$. These labeled edges represent conditional constraints; the lower-case labeled constraint applies when $A \Rightarrow C$ takes on its lowest value and the upper-case labeled constraint applies when $A \Rightarrow C$ takes on its largest value.

By inferring additional constraints using these edges, it is possible to assess whether an STNU is delay controllable with respect to a delay function γ (see rules reproduced from [Bhargava *et al.*, 2018] in Table 1).

Using these rules, assessing delay controllability reduces to finding a *semi-reducible negative cycle* in the STNU's labeled distance graph. A semi-reducible negative cycle is a negative cycle that, after applying a series of reductions, has no lower-case edges. Finding a semi-reducible negative cycle serves as a certificate that the STNU is not delay controllable with respect to a particular delay function γ. Efficiently finding and extracting these semi-reducible negative cycles will be an important building block as we consider how to find minimum-cost communication strategies that still let us achieve our goals.

Input: Labeled distance graph, $G = \langle V, E \rangle$; delay function γ
Output: Whether the STNU derived from the distance graph is delay controllable and if not, the edges embodying the conflict
Initialization:
1 $negNodes \leftarrow$ the set of all vertices with incoming negative edges;
2 $novel \leftarrow []$; list of newly added edges;
3 $preds \leftarrow \{\}$; mapping of function call to predecessors;
DELAYCONTROLLABLE?:
4 **for** $v \in negNodes$ **do**
5 $\quad cycleFree?, edges \leftarrow$ DELAYDIJKSTRA(G, γ, v, $preds$, $novel$, $[v]$, $negNodes$);
6 $\quad$ **if** !$cycleFree?$ **then**
7 $\quad\quad$ **return** $false$, EXTRACTCONFLICTS($edges$, $novel$, $preds$)
8 **return** $true, \emptyset$
Algorithm 1: Delay Controllability algorithm that reports conflicts

2.1 Modeling Communication Costs

We now define the communication cost minimization problem. For a given STNU, S, we wish to provide a delay function γ that minimizes the total communication cost $C(\gamma)$ subject to the constraint that S is delay controllable with respect to γ. We assume that the cost function C is component-wise monotonically decreasing. That is to say, given two delay functions γ_1 and γ_2, such that for all received timepoints x_e, $\gamma_1(x_e) \leq \gamma_2(x_e)$, we have that $C(\gamma_1) \geq C(\gamma_2)$.

This approach matches our intuitions for how to assign costs to communication patterns; we would never pay more to learn about an event at a later time. From a delay controllability perspective, we know that decreasing the communication delay for any given timepoint preserves controllability. Therefore, if there were a shorter delay with lower cost, we would always prefer to use that shorter delay.

2.2 Modeling Uncertain Communication and Risk

Our choice of allowable cost functions gives us significant leeway in modeling problems. Of particular note is the ability to model uncertainty in communication and to determine whether the system is controllable with some probability. This mirrors previous work that assesses the controllability of over-constrained temporal problems subject to some risk bound [Wang and Williams, 2015; Yu *et al.*, 2015].

Imagine that instead of picking an exact delay when communication happens, an agent provides a probability distribution over when the communication will happen. To get a controllability guarantee, it suffices to ask whether the model STNU is controllable when every communication event happens at the maximum of the allowable range. However, this

approach puts undue importance on the tails of these distributions. Instead a better question to ask is whether we can get some probabilistic guarantee of success (e.g. 98% of the time we can satisfy all constraints given this probability distribution).

Using our cost function framework, it is relatively straightforward to model this kind of problem. If we let d be a function mapping received timepoints to proposed bounds on their delay, we can use a cost function $C(\gamma) = 1 - P\left(\bigwedge_{x_e}[\gamma(x_e) \leq d(x_e)]\right)$. If we find an optimal cost solution, then our probability of success then becomes $1 - C(\gamma)$.

It is worth noting that this approach does not assume independence between timepoints. So long as there is a function that faithfully represents the distribution, we can determine whether our problem satisfies the provided risk bounds.

3 Conflict Extraction

Searching for an optimal-cost communication protocol can be expensive since the search space is continuous and unbounded. Rather than using uninformed search, we can learn how to fix low-cost communication strategies that are uncontrollable to guide our search towards an optimal result. To do so, we rely on *conflicts*, which serve as certificates explaining why our original network is uncontrollable.

3.1 Finding Conflicts

In the case of delay controllability, we know that the presence of a semi-reducible cycle in our STNU is such a certificate, and we present a way to extract those conflicts and a technique for trying to resolve them. Our conflict-extraction algorithm builds on top of the original delay controllability algorithm [Bhargava *et al.*, 2018] through the maintenance of some additional state.

The original delay controllability algorithm (Algorithm 1) works by invoking a variant of Dijkstra's algorithm from each negative weight edge. The calls to Dijkstra's algorithm are responsible for finding the shortest semi-reducible paths from each node to all others in the graph. The subroutine is recursively invoked any time another negative edge is found, and when an infinite recursion is detected, we know that we have found a cycle composed of several semi-reducible negative paths. More detail on the correctness of the algorithm can be found in [Bhargava *et al.*, 2018].

To efficiently extract delay controllability conflicts, we augment this algorithm to have it return the set of edges from the original graph composing the detected conflict. Lines 19-24 of Algorithm 2 are where we assemble the edges that compose the semi-reducible negative cycle. Whenever a recursive call to DELAYDIJKSTRA returns false, we know that at some point in the call stack, we discovered a semi-reducible negative cycle. However, the entire chain of edges is not necessarily part of the cycle. We use the third return value of DELAYDIJKSTRA to specify one node that is known to be part of the cycle. At line 23, we augment the list of edges that compose the negative cycle, and at line 24, we signal that we have fully specified a semi-reducible negative cycle because we have returned to a node we have already visited.

Input: Labeled distance graph $G = \langle V, E\rangle$, delay function γ, start node s, list of predecessor edges $preds$, list of new edges, $callStack$, and $negNodes$
Output: Whether the current walk is cycle-free, and the edges composing a semi-reducible negative cycle

Initialization:
1 $Q \leftarrow PriorityQueue()$;
2 $labelDist \leftarrow []$; shortest distances for labeled path;
3 $unlabDist \leftarrow []$; shortest distances for unlabeled path;
4 $labelDist[s] \leftarrow \langle 0, \emptyset\rangle$;
5 $unlabDist[s] \leftarrow \langle 0, \emptyset\rangle$;
6 **for** $e \in s.incomingEdges()$ *if* $e.weight < 0$ *and* $!e.lowerCase()$ **do**
7 $\quad Q.add(\langle e.from, e.label\rangle, e.weight)$;
8 $\quad (e.label == \emptyset\ ?\ unlabDist : labelDist)[e.from] \leftarrow \langle e.weight, e\rangle$

DelayDijkstra:
9 **if** $s \in callStack[1:end]$ **then**
10 $\quad$ **return** $false, \emptyset, s$;
11 $preds[s] \leftarrow \langle labelDist, unlabelDist\rangle$;
12 **while** $Q.size() > 0$ **do**
13 $\quad v, label, weight \leftarrow Q.pop()$;
14 $\quad$ **if** $weight \geq 0$ **then**
15 $\quad\quad G.add(\langle v, s, weight\rangle)$;
16 $\quad\quad novel.add(\langle v, s, weight\rangle)$;
17 $\quad\quad continue$;
18 $\quad$ **if** $v \in negNodes$ **then**
19 $\quad\quad newStack \leftarrow [v].concat(callStack)$;
20 $\quad\quad result, edges, end \leftarrow$ DELAYDIJKSTRA(G, γ, v, $preds$, $novel$, $newStack$, $negNodes$);
21 $\quad\quad$ **if** $!result$ **then**
22 $\quad\quad\quad$ **if** $end \neq \emptyset$ **then**
23 $\quad\quad\quad\quad edges.add($EXTRACTEDGEPATH($s$, v, $labelDist$, $unlabelDist$)$)$;
24 $\quad\quad\quad end \leftarrow (end == s)\ ?\ \emptyset : end$;
25 $\quad\quad\quad$ **return** $false, edges, end$;
26 $\quad$ **for** $e \in v.incomingEdges()$ *where* $e.weight \geq 0$ *and* $(!e.isLowerCase()$ *or* $e.label \neq label)$ **do**
27 $\quad\quad w \leftarrow e.weight + weight$;
28 $\quad\quad l \leftarrow (label \neq \emptyset$ and $w > -\gamma(label)\ ?\ \emptyset : label$;
29 $\quad\quad dist \leftarrow l \neq \emptyset\ ?\ labelDist : unlabDist$;
30 $\quad\quad$ **if** $Q.addOrDecKey(\langle e.from, l\rangle, w)$ **then**
31 $\quad\quad\quad dist[e.from] \leftarrow \langle w, e\rangle$;
32 $\quad\quad lower \leftarrow s.incomingLowerEdge()$;
33 $\quad\quad$ **if** $lower \neq null$ *and* $e.weight < \gamma(lower.label)$ *and* $Q.addOrDecKey(\langle lower.from, l\rangle, w + lower.weight)$ **then**
34 $\quad\quad\quad dist[lower.from] \leftarrow \langle w + lower.weight, e\rangle$;
35 $negNodes.remove(s)$;
36 **return** $true, \emptyset, \emptyset$;

Algorithm 2: Function DELAYDIJKSTRA

The introduction of additional processing to extract conflicts does not damage the algorithm's $O(n^3)$ runtime, as maintaining the additional data structures only incurs a constant overhead. Each call to EXTRACTEDGEPATH adds at most n edges to our list, and EXTRACTEDGEPATH is called

at most once per call to DELAYDIJKSTRA. Because DELAYDIJKSTRA is called at most once per node, it adds an additional overhead of $O(n^2)$, which is dominated by the normal runtime of the algorithm.

Finally, the call to EXTRACTCONFLICTS in line 7 of Algorithm 1 takes the list of edges composing the cycle and replaces any newly added edges with the original edges that were used to derive them. The resulting output is our conflict.

3.2 Resolving Conflicts

Now that we have a way of extracting conflicts when our STNU is uncontrollable, we need a way to resolve them. Given a semi-reducible negative cycle, there are two ways to eliminate it: make the cycle non-negative or make it non-semi-reducible. Since our communication protocol has no impact on the length of edges in the STNU's labeled distance graph representation, modifying our communication protocol will not affect the weight of the cycle. As a result, we must instead focus our attention on how to modify γ to eliminate semi-reducibility.

Only two of the reduction rules involve γ: the lower-case reduction rule and the cross-case reduction rule, and we explain how to change γ to resolve delay controllability conflicts and how this guarantees that iterative resolutions will eventually lead us to a valid solution.

Theorem 1. *If an STNU is controllable when $\gamma = 0$ (or in other words, is dynamically controllable), then if the STNU has a controllability conflict for any particular choice of γ, we can always adjust γ to eliminate a lower-case or cross-case reduction.*

Proof. A semi-reducible negative cycle is one where we can apply edge reductions to eliminate lower-case edges. In particular, the lower-case and cross-case reduction rules are the rules directly responsible for eliminating those edges.

For every lower-case edge from our conflict's semi-reducible negative cycle, we use the following approach for invalidating the reduction. From a lower-case edge with label b, we find the shortest subpath of the cycle that immediately follows the lower-case edge such that its total weight is less than $\gamma(B)$. We know such a subpath exists because all lower-case edges have non-negative weight, the total weight of the cycle is negative, and $\gamma(B) \geq 0$. If the weight of the subpath is non-negative, we adjust our $\gamma(B)$ to be equal to its weight.

If we cannot adjust any value of γ because all of the successive subpaths are negative, then we have a contradiction. This same semi-reducible negative cycle would still be present when $\gamma = 0$, and the original STNU would not be dynamically controllable.

□

4 Minimum-Cost Communication

With an efficient way to extract delay controllability conflicts, we can now focus on identifying low-cost and ultimately minimum-cost values of γ which yield delay controllable networks. In this section, we present three solutions that are based on a form of conflict-directed search. The first two have no optimality guarantees but are fast in practice, and the third is a conflict-directed best-first search that is guaranteed to yield an optimal value of γ.

Input: Labeled distance graph, $G = \langle V, E \rangle$ for STNU;
Output: A valid communication protocol γ or $\emptyset$ if one does not exist;

GREEDYCOMMCOST:
1 **if** *!*DELAYCONTROLLABLE?$(G, \gamma(\_) = 0)$ **then**
2 | `return` $\emptyset$;
3 $candidate \leftarrow \gamma(\_) = \infty$;
4 $controllable, conflict \leftarrow$ DELAYCONTROLLABLE?(G, $candidate$);
5 **while** *!controllable* **do**
6 | $candidate \leftarrow$ $candidate.pickResolution(conflict)$;
7 | $controllable, conflict \leftarrow$ DELAYCONTROLLABLE?(G, $candidate$);
8 `return` $candidate$;

Algorithm 3: Algorithm that performs different variants of greedy search to find a communication protocol for an input STNU that is delay controllable.

4.1 Conflict-Directed Search

Our initial approach at finding a feasible communication protocol (Algorithm 3) uses conflicts to iteratively refine its choice of delay function γ. Before we proceed with a potentially costly search process, we first check at line 1 to see whether the original STNU is dynamically controllable (or whether it is delay controllable with respect to $\gamma = 0$). Since decreasing observation delays always preserves controllability, we know that if the STNU is not dynamically controllable, we have no chance of finding a suitable delay function and can safely skip the search process.

Once we have a guarantee that there does exist some value of γ which makes the STNU delay controllable, we can begin our search. Each conflict returned from the delay controllability check (lines 4, 7) represents a disjunction of modifications we could make to our delay function to eliminate this particular conflict.

Our first two approaches, blind search and lowest-cost-resolution search (LCRS) employ different conflict resolution strategies to find an approach that works and are represented by different implementations of the $pickResolution$ function at line 6. Rather than keeping track of all possible branches, we choose a single disjunct to resolve and continue checking for delay controllability. In the case of blind search, we nondeterministically commit to any of the possible conflict resolutions, and in the case of LCRS, we commit to the conflict resolution with lowest possible cost. In expectation, blind search is quicker to pick a conflict resolution, but LCRS may find solutions that are overall lower in cost. While neither approach is optimal, both approaches are guaranteed to eventually find a satisfying delay function γ that yields a controllable STNU since they will eventually resolve all conflicts.

4.2 Conflict-Directed Best-First Search

While blind search and LCRS are appealingly simple, that they are not guaranteed to be optimal is cause for concern. We

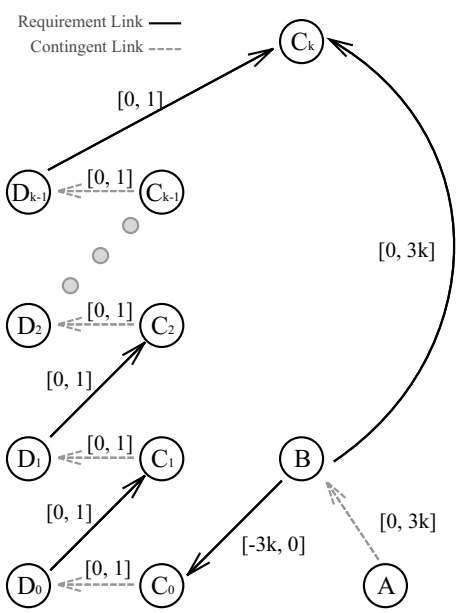

Figure 1: This k-contrived STNU is dynamically controllable but is not delay controllable if $\gamma = \infty$. Requiring that $\gamma(B) = 0$ is necessary and sufficient to make the STNU delay controllable.

are unable to provide guarantees that our search procedures are within a constant factor approximation of optimal, and in extreme instances, these search processes can yield results that are polynomially worse than optimal.

Theorem 2. *Blind search and LCRS are not guaranteed to yield results within a constant factor of optimal.*

Proof. Consider the STNU in Figure 1, which we call a k-contrived STNU, and imagine that we want to find a minimum-cost delay function γ making it controllable. For simplicity, assume that our cost function is $C(\gamma) = \sum_{x_e} \frac{1}{1+\gamma(x_e)}$. When called on this STNU, DELAYCONTROLLABLE? identifies a semi-reducible negative cycle that includes $A \Rightarrow B$ and $C_i \Rightarrow D_i$ for all $0 \leq i < k$. The corresponding resolution to that conflict requires that either $\gamma(B) = 0$ or for some $0 \leq i < k$, $\gamma(D_i) = 1$.

With a blind approach, we pick random disjuncts to satisfy until the STNU is controllable. However, each choice to relax $\gamma(D_i)$ makes no overall progress towards the controllability of the STNU. Only when $\gamma(B) = 0$ does the STNU become controllable. In expectation, $\frac{k}{2}$ relaxations happen before blind search relaxes $\gamma(B)$. Since each relaxation of $\gamma(D_i)$ to 1 incurs a cost of $\frac{1}{2}$ and relaxing $\gamma(B)$ to 0 incurs a cost of 1, in expectation blind search incurs a cost of $1 + \frac{k}{4}$.

With LCRS, the results are even worse. LCRS always elects to resolve the conflict by letting some $\gamma(D_i) = 1$ since this incurs a cost of $\frac{1}{2}$ whereas letting $\gamma(B) = 0$ incurs a cost of 1. As such, LCRS updates $\gamma(D_i)$ for all k such contingent links before updating $\gamma(B)$, meaning it incurs a cost of $1 + \frac{k}{2}$, whereas the optimal approach has a cost of just 1.

Input: Labeled distance graph, $G = \langle V, E \rangle$ for STNU, and a cost function C;
Output: A communication protocol γ that is of minimal cost or $\emptyset$ if one does not exist;
Initialization:
1 $queue \leftarrow []$ // queue of candidate γ functions;
MINCOMMCOST:
2 **if** !DELAYCONTROLLABLE?$(G, \gamma(\_) = 0)$ **then**
3 `return` $\emptyset$;
4 $queue.append(\gamma(\_) = \infty)$;
5 $\gamma \leftarrow queue.pop()$;
6 $controllable, conflict \leftarrow$ DELAYCONTROLLABLE?(G, γ);
7 **while** *!controllable* **do**
8 $queue.add(\gamma.allResolvedConflicts(conflict))$;
9 $queue.sortBy(C)$;
10 $\gamma \leftarrow queue.pop()$;
11 $controllable, conflict \leftarrow$ DELAYCONTROLLABLE?(G, γ);
12 `return` γ;

Algorithm 4: Algorithm that computes minimum-cost communication protocol for an STNU.

These results motivate our interest in developing an optimal algorithm.

□

Our optimal algorithm for finding a minimal communication cost (Algorithm 4) is a form of conflict-directed best-first search (CDBFS) [Williams and Ragno, 2007] and works as follows. Like with the greedy algorithm, we immediately check if the STNU is controllable with respect to $\gamma = 0$ or whether it is dynamically controllable (lines 2-3).

Once we know that a solution can be found, we now have to find the lowest-cost solution. Since we know that our cost function C is component-wise monotonically decreasing, we start with the lowest cost communication pattern possible, $\gamma = \infty$, and add it to our queue.

Every time we try a communication protocol that is too limiting and get back a conflict, we use the conflict to generate a set of delay functions to try later. For the value of γ we last checked, we derive modifications that each satisfy one of the provided disjuncts. We always choose the lowest cost value that satisfies the disjunct, which occurs when the delay for a received timepoint is exactly equal to the conflict's upper bound. By Theorem 1, we know that we will eventually reach a solution and because we are using best-first search, we know it will be optimal.

5 Experimental Results

The search algorithms we have presented differ in key ways. Blind search and LCRS have much lower overhead but have no guarantees of optimality. In contrast, CDBFS is guaranteed to output an optimal communication protocol but can be quite slow in practice. In this section, we present our empirical analysis as a way to better characterize exactly how different these approaches are and when it might be preferable to

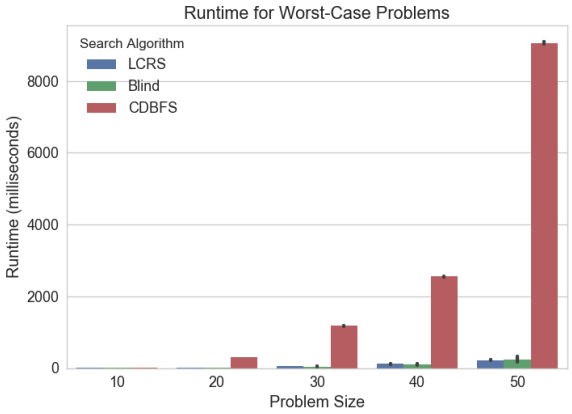

Figure 2: The runtimes of the solutions when run on k-contrived STNUs.

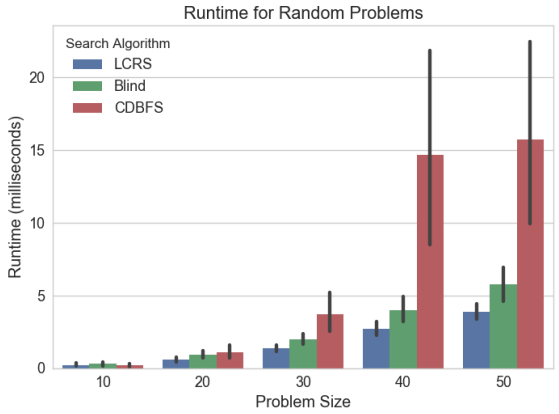

Figure 3: The runtimes of the solutions when run on random graphs.

use one over the other. All tests were run on a machine with an Intel i7 processor and 39GB of RAM.

We start by examining the performance on k-contrived STNUs. We performed 50 trials each with parameters $k = 10, 20, 30, 40, 50$, and for all experiments, we assumed a cost function of $C(\gamma) = \sum_{x_e} \frac{1}{1+\gamma(x_e)}$. As expected, LCRS always outputted a solution with cost $1 + \frac{k}{2}$, blind search outputted a cost that converged in expectation to $1 + \frac{k}{4}$, and CDBFS always outputted a unit cost solution.

As we scale problem size, the difference between the suboptimal approaches and CDBFS starts to grow (Figure 2). For each problem size, CDBFS has a significantly slower runtime than the suboptimal searches ($p << 0.01$) with no significant difference found between blind search and LCRS.

While there is a clear trade-off between the approaches, the data we have presented so far is for one specific type of graph. To validate that these trends hold more generally, we ran our experiments for randomly generated STNUs as well. Our random STNUs were composed of k independent contingent links which each had a lower bound of 0 and an integer upper bound that was uniformly chosen between 1 and 4. For each pair of endpoints between distinct contingent links, we added a requirement link between the two with probability $\frac{1}{4k}$. Each requirement link also had a lower bound of 0 and an integer upper bound chosen uniformly between 1 and 4. For our analysis, we ensured that our 50 trials were selected from the set of random STNUs that were dynamically controllable but not delay controllable. In instances where the STNU is either not dynamically controllable or is already delay controllable, the algorithms behave identically.

Our experiments demonstrate that the differences we saw in runtime between the optimal and suboptimal approaches persist (Figure 3). At problem sizes of $k = 30, 40, 50$, CDBFS is significantly slower than both suboptimal searches ($p < 0.05$). We also start to see a difference between blind search and LCRS. At $k = 30, 40, 50$, LCRS is also significantly faster than blind search ($p << 0.01$).

When we turn our attention to cost, however, we see that

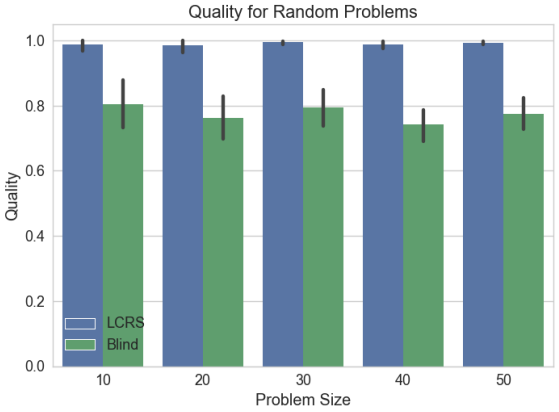

Figure 4: The quality of the solutions when run on random graphs. Quality is given by the optimal cost divided by the cost of the returned solution. A score of 1.0 represents the optimal solution.

on random graphs, we see a strong improvement in our results (Figure 4). Across all problem sizes, we see that while blind search is on average within 35% of optimal, remarkably, LCRS is on average within 1.5% of optimal. Given the massive difference in speed and the close approximation of optimal results, this provides strong support for the use of LCRS in low-cost communication protocol generation.

6 Conclusion

In this paper, we introduced the problem of producing a low-cost communication protocol that makes a temporal plan feasible in the face of uncontrollable actions. To produce such a protocol, we provided an efficient means of extracting delay controllability conflicts and used those conflicts as a means of guiding our search through a continuous state space. The three algorithms we presented, blind search, LCRS, and CDBFS, have very different properties with the first two being significantly faster with the last one guaranteeing optimality. While we provide theoretical results demonstrating that the suboptimal searches can be polynomially worse than

CDBFS, in practice we expect that LCRS provides highly competitive results with significant improvements in speed.

Our current efforts demonstrate an effective way to plan communication, but our approach makes the strong assumption that communication is reliable and that plans are immutable. In practice, however, communication is likely to have some probability success and as execution happens, plans can be changed to account for unexpected events. These areas of focus are both important when considering communication in the context of planning and are promising subjects for future areas of research.

Acknowledgments

This research was funded in part by the Toyota Research Institute under grant number LP-C000765-SR.

References

[Bhargava *et al.*, 2017] Nikhil Bhargava, Tiago Vaquero, and Brian Williams. Faster conflict generation for dynamic controllability. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-26, 2017*, pages 4280–4286, 2017.

[Bhargava *et al.*, 2018] Nikhil Bhargava, Christian Muise, Tiago Vaquero, and Brian Williams. Delay controllability: Multi-agent coordination under communication delay. In *DSpace@MIT*, 2018.

[Cui *et al.*, 2015] Jing Cui, Peng Yu, Cheng Fang, Patrik Haslum, and Brian Charles Williams. Optimising bounds in simple temporal networks with uncertainty under dynamic controllability constraints. In *ICAPS*, pages 52–60, 2015.

[Fang *et al.*, 2014] Cheng Fang, Peng Yu, and Brian C Williams. Chance-constrained probabilistic simple temporal problems. 2014.

[Morris, 2006] Paul Morris. A structural characterization of temporal dynamic controllability. In *International Conference on Principles and Practice of Constraint Programming*, pages 375–389. Springer, 2006.

[Morris, 2014] Paul Morris. Dynamic controllability and dispatchability relationships. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 464–479. Springer, 2014.

[Vidal and Fargier, 1999] Thierry Vidal and Helene Fargier. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental & Theoretical Artificial Intelligence*, 11(1):23–45, 1999.

[Wang and Williams, 2015] Andrew J. Wang and Brian C. Williams. Chance-constrained scheduling via conflict-directed risk allocation. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, January 2015.

[Williams and Ragno, 2007] Brian C Williams and Robert J Ragno. Conflict-directed a* and its role in model-based embedded systems. *Discrete Applied Mathematics*, 155(12):1562–1595, 2007.

[Yu *et al.*, 2014] Peng Yu, Cheng Fang, and Brian C Williams. Resolving uncontrollable conditional temporal problems using continuous relaxations. In *ICAPS*, 2014.

[Yu *et al.*, 2015] Peng Yu, Cheng Fang, and Brian C. Williams. Resolving over-constrained probabilistic temporal problems through chance constraint relaxation. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, Austin, TX, July 2015.