Trade-offs Between Communication, Rescheduling, and Success Rate in Uncertain Multi-Agent Schedules

David A. Chu and **Grace Diehl** and **Marina Knittel** and **Judy Lin** and **Liam Lloyd** and **James C. Boerkoel Jr.**
Computer Science Department
Harvey Mudd College
Claremont, California 91711
{dchulasso, gdiehl, mknittel, julin, wlloyd, boerkoel}@hmc.edu

Jeremy Frank
NASA Ames Research Center
Mountain View, California 94035
jeremy.d.frank@nasa.gov

Abstract

Generating and executing multi-agent schedules is difficult in uncertain environments. The current state-of-the-art algorithm maintains a high success rate by rescheduling frequently, but this approach involves substantial resource overhead due to computing and communicating new schedules. Aggressive rescheduling could thus reduce overall mission duration in situations where agents have limited energy and computing power. We thus explore the trade-off between the number of reschedules and success rate. Specifically, we propose three new algorithms that strategically decide when rescheduling is most likely to meaningfully increase the probability of success. Additionally, we empirically show that, while there is a trade-off between the number of reschedules and schedule success rate, it is possible to reduce the number of reschedules without proportionally decreasing success. We find that one of our approaches, Allowable Risk, allows us to gracefully trade reductions in success rate for significant reductions in the number of reschedules, and thus communication, of a state-of-the-art dynamic scheduling algorithm.

Introduction

Generating and executing schedules in multi-agent systems is an enabling technology for many applications, such as cooperative teams of airborne, surface operating, or underwater robots. Providing this capability requires effective multiagent coordination, since these applications involve uncertain environments that may challenge the success of a mission. In this paper, we focus on a scenario where this requirement is met by scheduling centrally and broadcasting a joint schedule to individual agents. However, a fixed, predefined schedule only uses the information that was available when it was created. As the mission progresses, uncertain events (e.g. unexpectedly long task durations) may disrupt the schedule. To take advantage of this new information, we need an algorithm to reschedule in response to these uncertain events.

Dynamic execution algorithms reschedule in response to new information, potentially increasing the chance that the mission succeeds. However, they may reschedule frequently, which means that the centralized scheduler would have to frequently send out new schedules to agents. While this frequent communication would not be a problem in some circumstances, in many applications, conserving battery power is an issue and communications are energy intensive, so any extra communication detracts from the time agents can spend completing their mission. To address this problem, we propose three new execution algorithms that limit how often schedules are sent:

- *Sufficient Improvement* sends out a new schedule only if it is likely to significantly increase the predicted probability of success;
- *Allowable Risk* reschedules more often when schedules are more risky and less often when schedules are less risky; and,
- *Coordination Targeting* uses constraints between agents to determine when to reschedule.

We conduct an empirical evaluation showing that Sufficient Improvement and Allowable Risk can decrease rescheduling without proportionally decreasing success rate, and evaluate the trade-offs between rescheduling frequency and success rate. We find that Allowable Risk allows us to gracefully trade reductions in success rate for significant reductions in the number of reschedules, and thus communication, of a state-of-the-art dynamic scheduling algorithm.

Background

Unmanned aerial vehicles (UAVs) have been used for missions ranging from collecting data on wildlife to monitoring wildfires. More complex missions have been proposed and, on small scales, demonstrated using UAV teams that communicate over radio cross-links with each other (e.g. Cesare et al.; Li et al. (2015; 2016)). Despite numerous advances in energy storage, battery life generally limits UAV mission duration (Quach et al. 2013). In particular, communications between UAVs and between the UAV and base stations can consume considerable energy, spurring research in energy-efficient communications between UAVs to enable relays (Zhang, Zheng, and Zheng 2017). In a UAV team, conducting missions in the presence of uncertainty that may require

rescheduling, communicating the new schedule consumes energy, and could therefore shorten the mission.

Let us consider a wildfire surveillance coordination problem involving two UAVs, Agent A and Agent B. Both agents must capture infrared images of the wildfire from different locations and send them back to a base station. Suppose for the entirety of this example that Agent B is between Agent A and the base station, and can communicate with both, but Agent A can only communicate with Agent B. Agent A takes an infrared image, but must subsequently relocate to a safe position due to unsafe flying conditions in its vicinity. Once Agent A is in its new position, it must send the image to Agent B, which can later send the image to the base station. However, Agent B must also travel a (shorter) distance before relaying Agent A's image, because it must take a second image at the new location. Agent B's second image acquisition must take place immediately after its first image, and thus before receiving Agent A's communication. Agent A's and Agent B's navigation tasks take around 40 and 10 seconds, respectively, and these durations are uncertain due to wind and localization error. The image sending/receiving task also takes an uncertain amount of time–around 5 seconds. Because the agents would have to do much more than mentioned (i.e. this subproblem is a small portion of a much larger problem), we want the agents to finish their tasks within a specified amount of time, namely 60 seconds.

In general, this class of problem can be posed and solved as a Decentralized Partially Observable Markov Decision Problem (DEC-POMDP). For instance, Wu, Zilberstein, and Chen (2011) consider how to reschedule in such cases in the presence of little communication. For our work, we limit the expressiveness of the problem under consideration to Probabilistic Simple Temporal Networks (PSTNs), defined in the next section. While tractable, the PSTN framework still exposes the fundamental problem of deciding when to reschedule, and how to communicate changes of schedules between agents.

Probabilistic Simple Temporal Networks

A *Simple Temporal Network* (STN), $S = (T, C)$, consists of a set $T = \{t_0, t_1, \ldots, t_n\}$, where each timepoint t_i represents the time at which a distinct event happens, and a set C of binary constraints c_{ij} on events in T. These constraints are of the form $t_j - t_i \leq b_{ij}$, for some $b_{ij} \in \mathbb{R}$ (Dechter, Meiri, and Pearl 1991). The two constraints between t_i and t_j can be written concisely as $t_j - t_i \in [-b_{ji}, b_{ij}]$. STNs are often encoded as directed graphs, where events are vertices and constraints are edges. A *schedule* is an assignment of values to events such that all constraints are satisfied. An STN is considered *consistent* if it has at least one schedule.

Since the physical world is inherently uncertain, accounting for uncertainty in our representation allows it to better model real-world problems. In a *Simple Temporal Network with Uncertainty* (STNU), the set of constraints C is divided into two disjoint subsets, called C_R, the set of *requirement* edges, and C_C, the set of *contingent* edges. Requirement edges are identical to the constraint edges in a standard STN. A contingent edge, however, represents that the time that elapses from t_i to t_j, given by $\beta_{ij} \in [-b_{ji}, b_{ij}]$, is chosen by an uncontrollable process and is unknown prior to execution (Vidal and Ghallab 1996).

An event whose incoming edges are all requirement edges is known as an *executable* timepoint, because the agent executing the schedule controls when it happens, or *executes*. A timepoint with an incoming contingent edge is known as a *contingent* timepoint, since it happens automatically some time after the timepoint that initiates the contingent edge. When a contingent timepoint happens it is said to be *received*. We call the set of contingent timepoints T_C, and the set of executable timepoints T_X.

STNUs are *strongly controllable* if each executable timepoint can be restricted such that, for *all* possible contingent timepoint outcomes, *all* requirement constraints are satisfied. Not all STNUs satisfy this restrictive property. Some STNUs are *dynamically controllable*, which means a limited form of contingent schedule can be produced prior to execution; these schedules describe, in a compact way, when to schedule future executable timepoints in response to contingent timepoint outcomes. In this paper, we use dynamic rescheduling to address these uncertain outcomes.

A *Probabilistic Simple Temporal Network* (PSTN) extends an STNU by adding information about the uncertain processes that govern contingent edges. For a PSTN's contingent edges, the time that elapses from t_i to t_j is chosen by a random variable X_{ij}, whose value is determined at execution by some PDF P_{ij} (Tsamardinos 2002; Brooks et al. 2015). Since contingent edges in PSTNs are governed by unbounded probability distributions, they cannot be strongly controllable. However, as we will later discuss, some algorithms still use the idea of strong controllability to solve PSTNs.

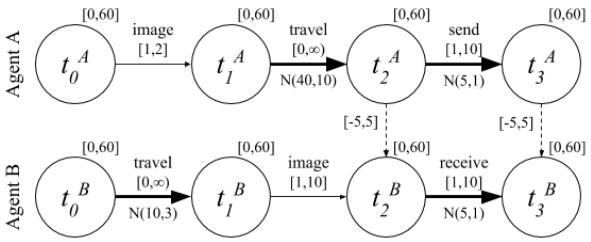


Figure 1: PSTN for our example problem.

Example Problem The PSTN representation of our running UAV example problem is shown graphically in Figure 1. Each vertex represents a timepoint. The agent to which a particular timepoint corresponds is indicated by the superscript. Two such timepoints could be the start and end times of Agent A's image taking task, which are represented as t_0^A and t_1^A, respectively. Directed edges represent temporal constraints and are labeled with the range of time that is allowed to elapse between the occurrence of the events represented by the source and target timepoints. There are three types of directed edges: thick edges represent contingent edges, dashed edges represent interagent constraints, and straight, slim edges represent requirement edges. All tasks must be completed within 60 seconds, shown by the constraint [0,60] above each vertex. Agent A's subproblem is contained in the top half of the figure, while Agent B's subproblem is contained in the bottom half of the figure.

Execution Algorithms for PSTNs

Early First Early First is a naïve algorithm for deciding when to execute the timepoints in a PSTN. As its name implies, it executes timepoints as soon as they can be executed–when they are both *live*, meaning that they are within their acceptable time range, and *enabled*, meaning that all predecessor timepoints have been executed.

The Static Robust Execution Algorithm While algorithms like Early First can be effective in practice, they are agnostic as to the impact of uncertainty on performance. In our UAV example problem, if both agents start navigating as soon as possible, it is highly likely that Agent B will arrive at its destination more than 10 seconds before Agent A, resulting in failure. To maximize the probability of success, Agent B should wait before navigating. The Static Robust Execution Algorithm (SREA) was motivated by this limitation (Lund et al. 2017). SREA tries to address this limitation by maximizing *robustness*, the probability that all events are executed without violating constraints (Brooks et al. 2015). Robustness is the complement of risk, introduced in (Fang, Yu, and Williams 2014).

In order to maximize robustness, SREA attempts to create a strongly controllable STNU with a minimum probability of failure. SREA sets a maximum probability α that a contingent edge in the original PSTN fails because it is too short or too long. This makes $1 - \alpha$ the minimum probability mass of the contingent edge captured by a corresponding interval in the STNU. To find the optimally robust schedule, SREA does a binary search over α. For each α, it uses a linear program to maximize the probability mass captured by the interval over each contingent timepoint. In a sense, SREA maximizes the probability that uncertain events will occur during these intervals. Once it finds the optimally robust schedule, it can execute this more constrained schedule using Early First. In our running example, SREA would constrain Agent B to wait before navigating to maximize the probability that the arrival times of both agents overlap.

SREA is good at using initial information to maximize the probability of success. However, it can fail when uncertain timepoints fall outside of their designated intervals during execution. In addition, SREA is limited because it cannot re-optimize constraints when new real-time information, such as the actual time of an uncertain event, is gained. This is because SREA is a *static* algorithm, in that it does not change the schedule in real-time. Therefore, dynamically updating the guiding schedul can be beneficial in situations with uncertain events.

The Dynamic Robust Execution Algorithm The Dynamic Robust Execution Algorithm (DREA) builds on SREA with the goal of maximizing robustness by adding the ability to incorporate new information during execution Lund et al. (2017). It creates an initial schedule by running SREA and uses it to guide execution. Whenever a contingent timepoint is received or enabled, DREA updates the PSTN with this new information, and calls SREA to create a new schedule.

Reducing Communication in DREA

DREA has a significantly higher success rate than Early First on the set of benchmark PSTNs in Lund et al. (2017). However, this high success rate comes with the cost of large amounts of rescheduling. In many scenarios, including as our UAV example, communicating new schedules is costly, so DREA may have undesireably high computational overhead.

With this in mind, we have investigated three possible methods for limiting when DREA reschedules such that communication is reduced without drastically diminishing success rate. *Sufficient Improvement* seeks to reduce rescheduling frequency by only rescheduling when the new schedule has a better predicted probability of success than the previous one. *Allowable Risk* makes rescheduling frequency proportional to the quantity of risk present in the problem. *Coordination Targeting* attempts to focus rescheduling only in preparation for events subject to inter-agent constraints.

Our algorithms are modifications to DREA. They act exactly as DREA does except when a timepoint is enabled or received, in which case, instead of always rescheduling, they more judiciously decide whether to reschedule. Sufficient Improvement and Allowable Risk use the minimum improvement thresholds m and x, respectively, in their calculations to make this decision. Allowable Risk also uses a counter k that increments whenever a timepoint is received and resets when rescheduling. The value of this counter and the threshold inputs is explained in our algorithm details. We rewrite DREA as a routine that utilizes our subroutines in Algorithm 1.

Algorithm 1: Modified DREA

```
Input  : A PSTN S, a rescheduling strategy s
Input  : A min improvement threshold 0 < m < 1
Input  : A min success threshold 0 < x < 1
guideSTN, α ← SREA(S) ;
k ← 0 ;
while S.isConsistent() and not S.allExecuted() do
    if any t ∈ T_C is received or enabled then
        S.update(t);
        if t is received then
            k ← k + 1 ;
        (guideSTN, α, k) ←
          maybeReschedule(guideSTN, s, m, x, α, k)
    else
        foreach live & enabled t ∈ T_X according to
          guideSTN do
            S.execute(t);
            guideSTN.execute(t);
```

Sufficient Improvement

The concept underlying Sufficient Improvement (SI) is to only use schedules that sufficiently improve the chances of success. Like DREA, it will always create a new schedule any time it gets new information. Unlike DREA, SI will only send out this new schedule if it calculates that the new schedule has a significantly larger probability of success than the schedule currently in use.

We represent SI as part of the $maybeReschedule$ subroutine of DREA presented as Algorithm 2. The

$maybeReschedule$ subroutine is called whenever a time point is received or enabled. SI requires an input minimum threshold for improvement $0 < m < 1$. When we run SI, it first calculates a new potential schedule with SREA, then records the number of contingent events left in our STN. SI then uses these values to determine if the difference between the probability of success of the new schedule and the probability of success as calculated at the last rescheduled is sufficient. When DREA runs SREA to generate a schedule, as noted above, it creates an interval around each uncertain edge that captures some amount of the probability mass of that edge. For each duration, this interval captures at least $1 - \alpha$ of the probability mass. SI calculates the probability of a schedule executing without violating a constraint by multiplying the captured probability mass for each uncertain edge. Generally speaking, this probability is bounded by $p \leftarrow (1-\alpha)^n$, where n is the number of uncertain edges. Note that this is actually an underestimate, since SREA expands the intervals somewhat beyond $1 - \alpha$ of the probability mass. SI then takes the difference between the new probability and the old probability. If this difference is above our threshold value m, it uses the new schedule.

If a contingent edge invalidates the current schedule generated by SREA by falling outside of its allowed range, SI continues to use the now-violated schedule until rescheduling is triggered. The execution is not considered failed unless one of the constraints from the original STN is violated. All of our rescheduling strategies handle violations of the SREA-generated schedule in this fashion.

Algorithm 2: $maybeReschedule()$ Subroutine

Input : An STNU $guideSTN$
Input : A rescheduling strategy s
Input : A min robustness α
Input : A received contingent event count k
Input : A min improvement threshold $0 < m < 1$
Input : A min success threshold $0 < x < 1$
if $s ==$ "SI" **then**
  $(maybeGuideSTN, \alpha_1) \leftarrow \text{SREA}(S)$;
  $n \leftarrow maybeGuideSTN.numCEventsLeft()$;
  $p_0 \leftarrow (1-\alpha)^n$;
  $p_1 \leftarrow (1-\alpha_1)^n$;
  if $p_1 - p_0 > m$ **then**
    $guideSTN \leftarrow maybeGuideSTN$;
    $\alpha \leftarrow \alpha_1$;
  return $(guideSTN, \alpha, k)$
else if $s ==$ "AR" **then**
  $n \leftarrow 0$;
  while $((1-\alpha)^{n+1} > x)$ **do**
    $n \leftarrow n + 1$;
  if $k \geq n$ **then**
    $(guideSTN, \alpha) \leftarrow \text{SREA}(S)$;
    $k \leftarrow 0$;
  return $(guideSTN, \alpha, k)$

SI's threshold influences how often the algorithm reschedules. If the threshold is low, we expect the algorithm will reschedule often, resembling DREA. Alternatively, if the threshold is high, the algorithm will reschedule less often, performing like SREA. In between extreme values, we expect to see a trade-off, where we reduce communication but also decrease the success rate as we increase the threshold.

Allowable Risk

Our next rescheduling strategy reduces rescheduling depending on the riskiness of a schedule. Allowable Risk (AR) decides how many uncertain events it can allow to occur statically with an acceptably high probability of success, as defined by a threshold. Then it simply allows those events to occur without intervening, and reschedules when they have all happened. This strategy limits communication by setting the rescheduling frequency in direct proportion to the risk associated with the intervals generated by SREA, since lower risk will allow larger groups of uncertain events to execute statically.

AR is also designed as part of the $maybeReschedule$ subroutine of DREA in Algorithm 2. It requires an input $0 < x < 1$ to represent the minimum robustness threshold. AR first finds the largest integer value of n such that $(1 - \alpha)^n > x$ for threshold x. AR then reschedules if the received event counter k exceeds n (see Alg. 1). When it reschedules, AR resets k to 0. Rescheduling will also produce a new α, leading to a new value for n (in the **while** loop of Alg. 2). Thus, if SREA generates a schedule with high α, AR will reschedule sooner than if SREA had generated a schedule with low α. As a proxy for probability of failure, α ties rescheduling frequency to the risk present in the schedules generated by SREA.

Since AR treats a threshold as a minimum probability of success allowed, a higher threshold could lead to frequent rescheduling, resembling DREA. However, a lower threshold could lead to low success rate from too little rescheduling, performing like SREA.

Coordination Targeting

Our last rescheduling strategy seeks to use the structure of STNs to identify occasions when rescheduling will be most meaningful. Intuitively, it seems likely that rescheduling in preparation for constraints between events belonging to different agents, or interagent constraints, will be particularly impactful, because these constraints represent more complex interactions involving multiple agents.

Based on this observation, we design Coordination Targeting (CT) to reschedule right before we execute executable events that are subject to interagent constraints, and right before we execute the last executable events preceding contingent events that are subject to interagent constraints. First, it sets a flag to track whether a timepoint has been received or enabled since last rescheduling. Next, if any executable timepoints are enabled, it creates a copy of the current PSTN with all requirement edges removed. The algorithm then checks whether any timepoints involved in interagent constraints are reachable from the currently enabled executable timepoints. If that condition is true and the flag is set to true, then CT calls SREA to generate a new schedule, and sets the flag to false. This way, we are able to focus rescheduling around the decisions we expect to matter the most: the final executable timepoints before an interagent constraint.

To further explain CT, consider running CT on the example PSTN illustrated in Figure 1. For the interagent constraint $t_2^B - t_2^A \in [-5, 5]$ the algorithm reschedules right before t_1^A and right before t_2^B, since t_2^B is an executable timepoint and t_1^A is the last executable before t_2^A, a contingent timepoint. For the interagent constraint $t_3^B - t_3^A \in [-5, 5]$, the algorithm provides no additional rescheduling, because there are no executable timepoints between t_2^A and t_3^A, nor between t_2^B and t_3^B. In other words, rescheduling at this point would not be useful, since we don't get to make any new decisions before t_3^A and t_3^B are received.

This algorithm could fail to reduce rescheduling if all events are constrained by interagent constraints. If there are not enough events under interagent constraints, this algorithm may act similarly to SREA, and therefore have lower robustness. Unlike our other two algorithms, the limitations of this algorithm are entirely dependent on the input. Thus, this algorithm cannot be tuned.

Algorithm 3: Coordination Targeting

```
Input : A PSTN S
guideSTN ← SREA(S) ;
while S.isConsistent() and not S.allExecuted() do
    new ← False ;
    resched ← False ;
    foreach t received or enabled ∈ T_C do
        S.update(t) ;
        new ← True ;
    foreach t enabled ∈ T_X do
        if t is controllable then
            pGraph ← guideSTN.justCEdges ;
            foreach interagent constrained s ∈ T do
                if t.reachs(s, pGraph) and new then
                    resched ← True
    if resched then
        guideSTN ← SREA(S) ;
        new ← False ;
    foreach live & enabled t ∈ T_X according to
     guideSTN do
        S.execute(t);
        guideSTN.execute(t);
```

Experimental Setup

Next we describe our experimental setup for empirically evaluating our approaches.

Scheduling Problem Testbed

Our evaluation methods attempt to recreate the experiments of Lund et al. (2017) to yield comparable results. First, our data set contains the same PSTNs used to evaluate DREA in Lund et al. (2017). These PSTNs were generated by the random robot navigation problem generator of Brooks et al. (2015). They generally have 20 timepoint variables divided among 2 to 4 agents, 20 to 35 total constraints, and a maximum of 15 contingent edges. We generated 30 PSTNs per combination of input features, totaling to 1620 schedules.

To evaluate a wide range of problems, our PSTNs were constructed from varying three input features: *degree of synchronization*, *interagent constraint density*, and *standard deviation*. The first feature is *degree of synchronization*, which sets a time upper bound between any two constrained events for different agents. In other words, it is the "tightness" of bounds on interagent constraints. From the Lund et al. (2017) data set, degree of synchrony varies between 1000, 2000, and 4000 times the standard deviation (described later in this section).

Another input feature is *interagent constraint density*, which defines the fraction of requirement constraints that are between agents. Varying our results across different values for interagent constraint densities reveals the impact of agent coupling intensity on the success of our algorithms. Interagent constraint density is set to 0.4 or 0.8.

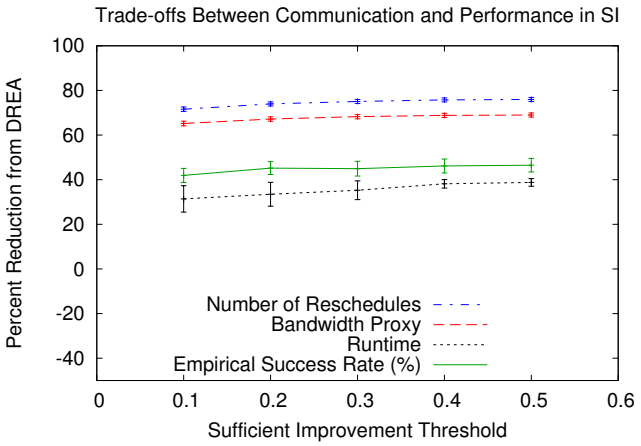
The last input feature is *standard deviation* of uncertain events to measure the degree of uncertainty in the input PSTN. The higher the degree of uncertainty, the wider the constraint probability distributions, and the harder it is for SREA to obtain a large α value. Varying the degree of uncertainty informs our understanding of our execution algorithms' performance since they aim to find a successful schedule in uncertain situations. The standard deviation was measured by a kurtosis metric for contingent edge distributions, which took values 1, 3, and 5. Kurtosis is a measure of how "peaky" a distribution is. A kurtosis value of 3 corresponds to a normal distribution, while a kurtosis of 1 corresponds to a flatter distribution with higher standard deviation, and a kurtosis of 5 corresponds to a more "peaky" distribution with lower standard deviation.

These problem features may generally impact the trade-off between robustness and communication that our algorithms are designed to manage. If we find our algorithms behave consistently across different values of these three input features, we will have a stronger basis to claim that our results generalize to different types of input scenarios. On the other hand, if our results are inconsistent across different values of one feature, we gain more insights into the scenarios in which one algorithm outperforms another.

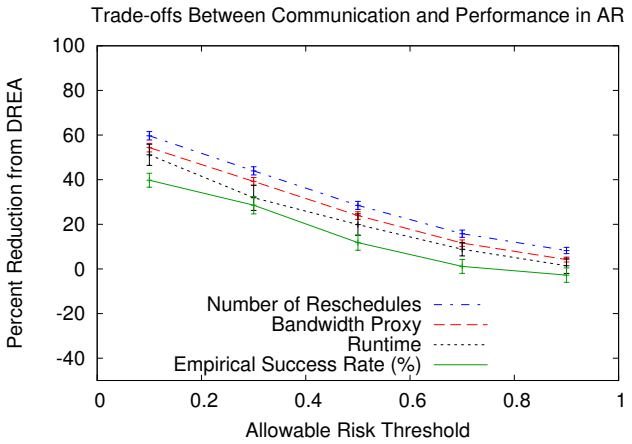
Simulation

We also adapted the simulation software used in Lund et al. (2017). The simulation uses our data set to measure average success rate and duration. We additionally measured two other metrics, number of reschedules and accumulated bandwidth, to analyze trade-offs. The number of reschedules counts how often an algorithm sends out a new schedule, and accumulated bandwidth sums the sizes of all sent schedules, where the sizes are calculated as the sum of the number of edges and number of timepoints in the schedule.

We evaluate each algorithm on each PSTN in the data set 390 times, sampling uncertainty distributions during execution. For SI, we evaluate thresholds from 0.1 to 0.5 in steps of 0.1. For AR, we vary the threshold from 0.1 to 0.9 in steps of 0.2. In varying the thresholds, we gain more insight into the trade-offs between relaxed rescheduling constraints and schedule success rate.



(a) Sufficient Improvement (SI) threshold variation results



(b) Allowable Risk (AR) threshold variation results

Figure 2: Simulated results for Sufficient Improvement (SI) and Allowable Risk (AR). We plot the success rate, number of reschedules, and bandwidth for each threshold as percent reduction relative to DREA.

Counting Rescheduling in DREA

DREA reschedules whenever a contingent timepoint is received or enabled. This means that between the time a contingent timepoint becomes enabled and the time it is received, DREA is continuously rescheduling. However, in practice, the simulator that we adapted is event-based, which only yields opportunity to reschedule when timepoints are received or executed. This behavior prevents DREA from rescheduling continuously between events, significantly reducing how often it reschedules. We expect that our rescheduling strategies would dramatically reduce continuous scheduling between timepoints in DREA implementations that use a time-based simulator, so their reduction in rescheduling would be even greater.

Empirical Evaluation

The ultimate goal of our analysis is to understand the trade-off between rescheduling and success rate in our algorithms. We consider the thresholds of SI and AR, because thresholds control how selective an algorithm is about sending out new schedules, which then impacts success rate. We compare the success rates and number of reschedules of our algorithms to those of DREA and the minimally scheduling algorithms, Early First and SREA. Finally, we further examine the trade-off between rescheduling and success rate for different thresholds of AR. Through such analysis, individuals can determine which algorithm and threshold are most appropriate for their purposes.

Impact of Thresholds

First, we compare the performance SI and AR with different thresholds against DREA. In Figure 2, we graph a percent reduction in success rate, number of reschedules, bandwidth, and simulation duration from DREA in tandem across all thresholds for a single algorithm. Our percent decrease for SI, for example, is calculated as in Equation 1.

$$Dec = 100 \cdot \frac{x_{DREA} - x_{SI}}{x_{DREA}}. \quad (1)$$

Here, x_{DREA} is the average value of that metric for DREA, x_{SI} is that for SI, and Dec is the percent decrease in the metric. We expect a positive value, because DREA generally has higher values for all the dependent metrics. A near zero value signifies this algorithm performs similarly to DREA, and a larger value indicates a larger decrease in that metric for the algorithm. Ideally, for the new algorithms, success rate would not decrease relative to DREA; however, rescheduling, bandwidth proxy, and runtime ideally would decrease.

We make three significant observations about SI from Figure 2a. First, note that all values are positive. Since the graph depicts the percent reduction from DREA, that means our algorithms have a lower success rate as well as less rescheduling than DREA, as expected. Additionally, the metrics do not vary much across thresholds for SI; the total ranges are only $3.66\% \pm 2.52\%$ for success rate, $4.34\% \pm 0.70\%$ for number of reschedules, and $3.78\% \pm 0.76\%$ for bandwidth. Thus, our thresholds (0.1 - 0.5) have no significant impact on the functionality of the algorithm. We theorize that if we had expanded our range of thresholds to contain negative values (i.e. the new schedule is riskier than the old), then as the threshold approached -1, the performance would approach that of DREA. Such behavior occurs because a very low threshold allows the algorithm to reschedule in more cases, approaching the functionality of DREA. Likewise, in the higher range, SI may reschedule infrequently, and thus act more similarly to SREA. Finally, the percent reduction for success rate ($42.45\% \pm 0.78\%$ on average) is significantly lower than the percent reduction in number of reschedules and bandwidth ($74.26\% \pm 0.22\%$ and $67.44\% \pm 0.24\%$ on average respectively). This comparison signifies that the proportional loss in success rate is, on average, $31.81\% \pm 0.81\%$ and $24.99\% \pm 0.82\%$ smaller than the proportional loss in the respective communication metrics. Thus, SI results are promising in reducing communication to a large extent while preserving a higher success rate.

Figure 2b plots the same data across different thresholds for AR. From this plot, we see many of the same patterns present in the corresponding SI graph. All values are within error of or above zero, so AR consistently has lower success rates and communication metrics, as expected. The gaps

AR Threshold	Reschedules	Success Rate
0.1	1.7	21%
0.3	2.8	22%
0.5	3.9	27%
0.7	5.8	30%
0.9	6.7	32%

Table 1: For each threshold of Allowable Risk (AR), we also compute the average number of reschedules and the average success rate across all runs over all inputs.

between the success rate with number of reschedule and bandwidth reductions are smaller ($16.84\% \pm 0.84\%$ and $12.14\% \pm 0.84\%$ on average respectively) so AR is less effective at lowering rescheduling than SI. However, AR does a better job of maintaining success rate. Its success rate is within error of DREA's at threshold values of .7 and .9, where success rate is reduced by $1.14\% \pm 3.12\%$ and $-2.78\% \pm 3.22\%$ compared to DREA, respectively.

The main difference is that the performance of AR varies with respect to threshold. As the threshold decreases, the success rate and rescheduling reductions all decrease at approximately the same rate (on average, $9.67\% \pm 1.17\%$ and $12.75\% \pm 0.46\%$ per 0.2 units of threshold respectively). We expect this result, because at a threshold of 0, AR will always reschedule, and thus act like DREA, while at a threshold of 1, it will never reschedule, and thus act like SREA. This graph explores thresholds in between and depicts a reasonable downward trend. AR also decreases communication more than success rate. In addition, varying the parameter depicts a clear trade-off between low thresholds (following low communication) with low success rate. Therefore, AR is tunable across the range 0.1 to 0.9.

In Figure 2, we also map the percent reduction for runtime. For AR, the runtime depicts the same general downward trend across threshold as the other metrics. This downward trend is expected, because AR refrains from computing new schedules when it decides against sending one. SI does not exhibit this same property; it always calculates a new schedule, then decides whether to send it out.

Trade-offs Within Allowable Risk

We also attempt to more clearly understand the magnitude trade-offs between number of reschedules and success rate in AR. In Table 1, we calculate AR's average success rate and number of reschedules for a given threshold. Note, this table represents the same experiment as in Figure 2b, but this time we present the absolute (vs. relative) success rate and number of reschedules. We observe that average success rate tends to increase as average number of reschedules increases. We would expect this trade-off, because the higher the number of reschedules, the more information used in rescheduling, the better the performance. Note this pattern is not evident for SI.

Across our analyses, we explore a number of ways to represent the trade-offs between success rate and communication. Our algorithms successfully explore the space between SREA and DREA in terms of amount of communication and success rate. In addition, we find the AR threshold corresponds well with amount of communication, and thus provides a good basis for directly analyzing the trade-off between communication and success rate.

Impact of Problem Features

Beyond evaluating threshold effects, we compare our algorithms with DREA, SREA, and Early First across different classes of inputs. Figure 3 show two examples of simple average metric comparison across different algorithms. For all algorithms, we graph the success rate, number of reschedules, and bandwidth as functions of degree of synchrony, interagent constraint density, and standard deviation. This arrangement yields a unique graph for each pairing of a dependent and independent variable (ie, success rate and degree of synchrony). With these graphs, we can directly compare the performance of all parameters for each metric across all input features.

In this paper, we only show success rate and number of reschedules for degree of synchrony in Figure 3. We selected these graphs to show clear trends from several input values. However, the same trends are also present for interagent constraint density and standard deviation.

Figure 3a shows that the success rate of all of our algorithms is consistently between that of DREA and SREA, as expected. Our algorithms all are modifications of DREA that schedule less, and therefore don't utilize as much information. However, they all still reschedule more than SREA. It is also noteworthy that SI and AR outperform Early First, while only CT seems to track Early First, even though Early First never communicates. Therefore, CT failed to preserve a large enough success rate to make rescheduling worthwhile in comparison.

In addition, we note that as degree of synchrony increases, success rate increases. Such a correlation makes sense, because the higher the value, the larger the possible acceptable time range for constraints between agents. The constraints are simply less strict, and thus easier to satisfy.

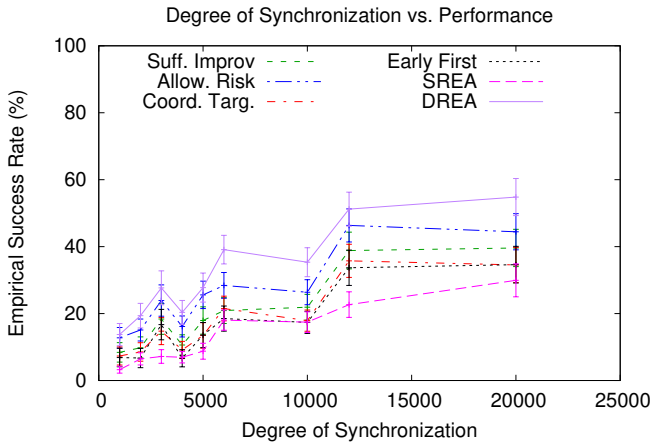
Similarly, Figure 3b shows that the number of reschedules of all of our algorithms is consistently between DREA and SREA. SI reschedules at a rate close to SREA, which is optimal. CT and AR both consistently outperform DREA, but are not as good as SI in this sense. Lastly, there is no trend in number of reschedules as degree of synchrony increases.

Figure 3, as well as all graphs of this class, additionally express that the aforementioned results hold across different input feature values. The input features are designed to represent important aspects of scenarios, implying our conclusions hold for many different types of scenarios. Our results are thus more generalizable.

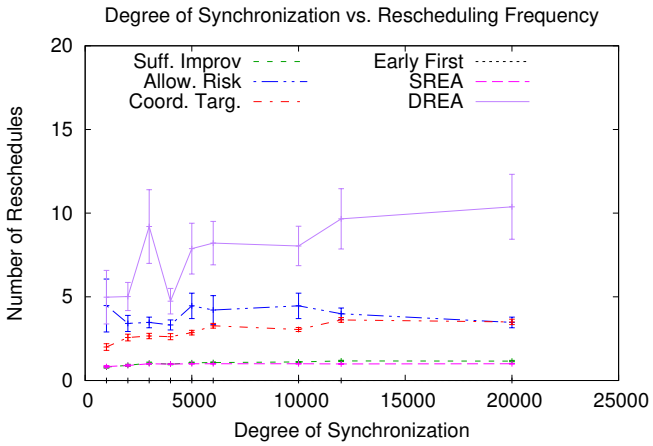
Discussion

In this paper, we augment Lund et al. (2017)'s DREA to maintain its high success rate while simultaneously reducing its high number of reschedules. To this end, we propose three new algorithms: Sufficient Improvement (SI), Allowable Risk (AR), and Coordination Targeting (CT).

Our exploration of these algorithms and their parameters shows a clear trade-off between rescheduling and success. We are therefore unable to conclude which of our algorithms is best, because it depends on the relative importance of rescheduling and success for a given application.



(a) Success rate across degree of synchrony



(b) Number of reschedules across degree of synchrony

Figure 3: Results of all our algorithms across different degrees of synchronization. The thresholds were selected to be intermediate values in the tested range, 0.3 for Sufficient Improvement (SI) and 0.5 for Allowable Risk (AR).

SI and AR show promise, where CT was unable to outperform Early First. Results for SI and AR are consistent for simulations that were run and evaluated on PSTNs varying across a number of identified features. Our analysis is thus generalizable to many different kinds of input scenarios.

SI has an impressive gap between its percent reduction of success rate and communication with respect to DREA. This difference means that the loss in algorithm success for the amount it reschedules is relatively low, resulting in a less costly trade-off between success rate and communication. Unfortunately, changing the SI threshold does not appear to affect success rate or communication. Therefore, threshold tuning within our explored range has a negligible effect on the performance of the algorithm. Future work might consider exploring other values of the SI threshold outside our range, particularly negative values.

While AR does not decrease communication as much as SI, for certain thresholds it reaches much higher success rates. Most importantly, AR exhibits a clear trade-off: as threshold increases, amount of communication decreases and success rate increases. Thus, AR is more tunable to be appropriate for the situation in which it is used. In scenarios where communication is especially costly, like in our UAV example, lower thresholds are preferable. However, if communication is only slightly limited, the AR threshold can be increased to capture a larger success rate. Thus, AR is more adjustable for various situations than SI.

As we can see, there is no clear winner between SI and AR; each algorithm may be preferable in different situations. Moreover, in scenarios where communication cost is negligible, DREA may be more suitable, and in scenarios where communication cost is extreme, Early First might be best. In any case, our contributions have successfully explored the trade-offs between success rate and communication in the region between DREA and static algorithms.

Future research could explore combining these algorithms in an attempt to reduce scheduling further. SI might, for example, be layered on top of AR. Such an algorithm might vary its rescheduling frequency based on the risk present in the system (like AR), and then only sends out new schedules if they constitute a significant improvement over the current one. SI itself might attain higher robustness (probably accompanied by a higher rescheduling rate) by rescheduling when new schedules constitute a significant *change* in predicted success rate, rather than a significant *improvement* in predicted success rate.

Another possible direction for future work is to test these algorithms in other kinds of simulations. For instance, one could run tests in a time-based simulation, as opposed to an event-based simulation. It would be interesting to see how DREA, or an adapted version of our algorithms, would perform in this context. Finally, physics-based or real-world simulations would yield results more directly relatable to practical situations. This would create a stronger basis for justifying the value of SI and AR, and defining the trade-off between success rate and communication.

Acknowledgements

We would like to thank the Clinic organizers, Professor Zachary Dodds and DruAnn Thomas. We would also like to thank Jordan Abrahams, Hamzah Khan, Kyle Lund, and Brenner Ryan for lending us their code and expertise. This research was supported by the NASA Advanced Exploration Systems (AES) program.

References

Brooks, J.; Reed, E.; Gruver, A.; and Boerkoel, J. C. 2015. Robustness in probabilistic temporal planning. In *Proc. of the 29th National Conference on Artificial Intelligence (AAAI-15)*, 3239–3246.

Cesare, K.; Skeele, R.; Yoo, S.; Zhang, Y.; and Hollinger, G. 2015. Multi-UAV exploration with limited communication and battery. In *Proc. of the IEEE Conference on Robotics and Automation (ICRA)*, 2230 – 2235.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. In *Knowledge Representation*, volume 49, 61–95.

Fang, C.; Yu, P.; and Williams, B. C. 2014. Chance-constrained probabilistic simple temporal problems. In

Proc. of the 28th National Conference on Artificial Intelligence (AAAI-16), 2264–2270.

Li, B.; Jiang, Y.; Sun, J.; Cai, L.; and Wen, C.-Y. 2016. Development and testing of a two-UAV communication relay system. *Sensors* 16(10).

Lund, K.; Dietrich, S.; Chow, S.; and Boerkoel, J. 2017. Robust execution of probabilistic temporal plans. In *Proc. of the 31st National Conference on Artificial Intelligence (AAAI-17)*, 3597–3604.

Quach, C.; Bole, B.; Hogge, E.; Vazquez, S.; Daigle, M.; Celaya, J.; Weber, A.; and Goebel, K. 2013. Battery charge depletion prediction on an electric aircraft. In *Proc. of the Annual Conference of the Prognostics and Health Management Society (PHM 2013))*.

Tsamardinos, I. 2002. A probabilistic approach to robust execution of temporal plans with uncertainty. In *Methods and Applications of Artificial Intelligence*. Springer. 97–108.

Vidal, T., and Ghallab, M. 1996. Dealing with uncertain durations in temporal constraint networks dedicated to planning. In *Proc. of European Conference on Artificial Intelligence (ECAI-96)*, 48–54.

Wu, F.; Zilberstein, S.; and Chen, X. 2011. Online planning for multi-agent systems with bounded communication. *Artificial Intelligence* (175):487 – 511.

Zhang, J.; Zheng, Y.; and Zheng, R. 2017. Spectrum and energy efficiency maximization in UAV-enabled mobile relaying. In *Proc. of IEEE International Conference on Communications (ICC))*.