Generating Temporally Contingent Plans

Janae N. Foss **Nilufer Onder**
Department of Computer Science
Michigan Technological University
1400 Townsend Drive
Houghton, MI 49931
{jnfoss,nilufer}@mtu.edu

Abstract

Uncertainty applies to many aspects of planning problems. Much research has been done to deal with problems where actions have uncertain effects. In reality, many planning problems also involve actions with uncertain durations, but this type of uncertainty has not been widely studied in planning until the recent development of several planners which incorporate durational uncertainty. Additionally, theoretical work has been done on characterizing the level of controllability in plans involving actions with uncertain durations. We have developed an approach for finding temporally contingent plans, i.e., plans with branches that are based on the duration of an action at execution time. More specifically, the problems we are studying satisfy the following criteria: (1) there is more than one solution plan, (2) solution plans are ranked by a metric that is not fully based on makespan, (3) actions have uncertain durations, (4) the start and/or end times of some actions are constrained, and (5) as actions require more time to complete, plans that are judged highly by the metric become invalid. We describe our approach for determining the time points that cause an unsafe situation and for inserting temporal contingency branches. Experimental results with both sequential and parallel plans are discussed.

1 Introduction

Uncertainty applies to many aspects of planning problems. Much research has been done to deal with problems where actions have uncertain effects. A classification of planners that can handle this kind of uncertainty is given in [Dearden *et al.*, 2003]. In reality, many planning problems also involve actions with uncertain durations, but this type of uncertainty has not been widely studied in planning until the recent development of several planners which incorporate durational uncertainty [Younes and Simmons, 2004; Mausam and Weld, 2005; Little *et al.*, 2005]. Additionally, there has been theoretical work on characterizing the level of controllability in plans involving actions with uncertain durations [Vidal and Fargier, 1999]. Conservative planning (i.e., finding plans that are likely to be safe regardless of action duration) is one way to handle durational uncertainty. The advantage to this approach is that the resulting plans are robust. However, conservative planning often results in missed opportunities. To take advantage of available opportunities while still having a robust plan, we have developed an approach for finding temporally contingent plans (TCPs), i.e., plans with branches that are based on the duration of an action at execution time. Specifically, the problems we are studying satisfy the following criteria: (1) there is more than one solution plan, (2) solution plans are ranked by a metric that is not fully based on makespan[1], (3) actions have uncertain durations, (4) the start and/or end times of some actions are constrained, and (5) as actions require more time to complete, plans that are judged highly by the metric become invalid. We take an optimistic approach by first finding a plan that is valid when all actions complete quickly. We then use the methods described in [Dechter *et al.*, 1991] to determine when the plan may fail. At time points that cause an unsafe situation, temporal contingency branches are inserted.

As an example, consider the problem of traveling from home to a conference. One solution plan is to drive to the airport, fly to the destination city, take a shuttle to the conference venue, and finally register for the conference. Another solution plan could involve taking a taxi instead of a shuttle to the venue. Assuming the metric is to minimize money spent, the plan with the shuttle action would be preferred. However, the taxi may be faster than the shuttle and since there are constraints on the time one can register for the conference, depending on how long the flight takes, there may only be enough time for the more expensive taxi option. To always have a safe plan, and be able to save money when possible, our approach would generate a TCP to drive to the airport, fly to the destination, take the shuttle if there is enough time, otherwise take the taxi, and register for the conference. In this way, our approach ensures enough time for the worst case while making use of better options when time allows. Throughout this paper our running example will be the conference domain.

Our contributions are threefold: (1) we introduce the no-

[1] As we define it, a metric may either combine makespan with some nontemporal measure or simply be stated as a nontemporal measure.

tion of TCPs, (2) we provide a greedy iterative algorithm that inserts branches based on time rather than world conditions, (3) we show that viable plans can be generated in this framework. In the remainder of this paper we first define temporal uncertainty and explain our algorithm for creating TCPs. We then give a theoretical framework for characterizing solution plans. This is followed by preliminary experiments, general remarks, and a description of future work.

2 Planning with Temporal Uncertainty

Temporal planning and reasoning activities involve actions that have a temporal extent, such as an action duration, and general temporal constraints, such as a deadline for when an action must begin. Problems with temporal aspects may have actions with uncertainty present in one of the following three ways. First, the temporal aspects are certain while the effects on the world are uncertain. For example, eating a meal may take 30 minutes, but it is uncertain if hunger will be satisfied. Second, the changes in the world are certain, but the temporal aspects are uncertain. For example, hunger will be satisfied after eating a meal, but the duration of the meal is uncertain. Third, both the temporal aspects and changes in the world are uncertain. For example, a meal will be eaten but it is uncertain how long it will take and whether hunger will be satisfied. For simplicity, our current work is concerned with only the first type of uncertainty. We plan to deal the other two types of uncertainty in future work.

In our framework, uncertainty in action duration is represented with the interval [*min-d*, *max-d*], where *min-d* and *max-d* are minimum and maximum reasonable durations[2], respectively (*min-d* > 0 and *max-d* $< \infty$). We have extended PDDL2.2 [Edelkamp and Hoffman, 2004] to represent *interval durative* actions as opposed to single point durative actions. In temporal reasoning literature when an interval duration is assigned to an action, it is generally assumed that the user can select any value from the interval. In our work we build on the model described in [Vidal and Fargier, 1999] and assume that some action durations will be known only at execution time and thus are uncertain. Hence we define two types of interval durative actions. If the duration is *assignable*, the executing agent (user) can choose a duration between *min-d* and *max-d*. If the duration is *uncertain* the action will consume some time between *min-d* and *max-d*, but the exact duration is beyond the control of the agent. In the conference domain, the duration of a flight is uncertain, but the duration of eating a meal is assignable.

In Figure 1 our coding of the conference domain is given. Note that the `eat-meal` action has an assignable duration but the other actions have unassignable (i.e., uncertain) durations. As defined, `fly_airport2_airport1` has a duration between 45 and 90 time units, starting at time 30 and conference registration has a duration between 5 and 10 time units, starting between times 84 and 141, exclusive. These two actions show the syntax we have added for associating actions with their execution time constraints. In our framework we assume that there is no penalty involved with waiting to begin execution of an action. The addition of assignable and unassignable interval durations is a conceptual extension to PDDL2.2, whereas the `execution-time` syntax provides convenience in coding but can be represented indirectly by the timed initial literals of PDDL2.2 which are used to define temporal windows.

Domain description

```
(define (domain conference-travel)
  (:requirements :fluents :equality
   :interval-durative-actions :execution-times)
  (:predicates (at_airport1) (at_airport2) (at_hotel)
               (not_hungry) (attending_conference))
  (:functions   (money_spent))

  (:interval-durative-action fly_airport2_airport1
   :unassignable-interval-duration
        (and (min ?duration 45) (max ?duration 90))
   :condition (at start (at_airport1))
   :effect (and (at end (at_airport2))
    (at start (not (at_airport1)))
    (at start (increase (money_spent) 200)))
   :execution-time (start at 30))

  (:interval-durative-action taxi_hotel_airport2
   :unassignable-interval-duration
       (and (min ?duration 15) (max ?duration 20))
   :condition (at start (at_airport2))
   :effect (and (at end (at_hotel))
    (at start (not (at_airport2)))
    (at start (increase (money_spent) 120))))

  (:interval-durative-action shuttle_hotel_airport2
   :unassignable-interval-duration
        (and (min ?duration 30) (max ?duration 60))
   :condition (at start (at_airport2))
   :effect (and (at end (at_hotel))
    (at start (not (at_airport2)))
    (at start (increase (money_spent) 20))))

  (:interval-durative-action eat_meal
   :assignable-interval-duration
           (and (min ?duration 20) (max ?duration 60))
   :condition (at start (attending_conference))
   :effect (at end (not_hungry))
    (at start (increase (money_spent) 20))))

 (:interval-durative-action register_for_conference
  :unassignable-interval-duration
           (and (min ?duration 5) (max ?duration 10))
  :condition (over all (at_hotel))
  :effect (at end (attending_conference))
  :execution-time (and (start after 84) (start before 141))))
```

Problem description

```
(define (problem conference-travel-1)
  (:domain conference-travel)
  (:init        (at_airport1)
                (= (money-spent) 0))
  (:goal        (attending_conference))
  (:metric      minimize (money-spent)))
```

Figure 1: Conference travel domain and problem.

3 Creating Temporal Contingency Plans

When creating a TCP it is important to find a plan that is both *safe* and ranked highly by the plan metric. Intuitively, a plan

[2]These durations may be determined using a probabilistic distribution. The 95th percentile has been used to produce conservative plans (e.g. [Fox and Long, 2002]) and may be a good selection for *max-d*. Experimental work must be done to determine a reasonable percentile for *min-d*.

is safe if its validity is guaranteed, even when all of its uncertain actions require their maximum duration to complete. One approach to building safe plans in this context is to assume that all actions always require their maximum duration. Though the result is a robust plan, in our framework this pessimistic assumption leads to a plan that is not ranked well by the metric. We take an optimistic approach and assume that actions require only their minimum duration. Since this assumption may yield an unsafe plan, we build temporally contingent branches into it using a general Just-In-Case style algorithm [Drummond *et al.*, 1994] where we generate a seed plan, find points where it is likely to fail, and then insert contingency branches at those points. Our algorithm is given in Figure 2. To generate the seed plan, we make the optimistic

TCP Algorithm

(1.1) Generate a seed plan, *P* with *n* actions, assuming all actions require only their minimum duration

(1.2) Construct the distance graph *D* of *P*

(1.3) *TCP* ← MakeSafePlan(*P*, *D*)

MakeSafePlan(Plan *P*, DistanceGraph *D*)

(2.1) Create *TCP* with *P* as main branch

(2.2) For each action *i* = *n* to 1 in *P*

(2.3) *maxAllowedDuration* ← shortestPathDistance(s_i, e_i, *D*)

(2.4) While *maxAllowedDuration* < maxDuration(*i*)

(2.5) *newMinDuration* ← *maxAllowedDuration* + 1

(2.6) *TCB* ← GetContingencyBranch(*i*, *newMinDuration*)

(2.7) Insert *TCB* into *TCP* to be executed when *i* requires more time than *newMinDuration*

(2.8) *maxAllowedDuration* ← shortestPathDistance(s_i, e_i, *DB*)

(2.9) Modify *D* so that *i* is constrained to start at the latest time *i* can safely start

(2.10) return *TCP*

GetContingencyBranch(Action *i*, Duration *newMinDuration*)

(3.1) Modify domain to assume *i* requires *newMinDuration*

(3.2) Modify initial conditions of problem to reflect state of world at start of *i*

(3.3) *earliestStartTime* ← -1 × shortestPathDistance(s_i, s_0, *D*)

(3.4) Modify problem to constrain *i* to start at *earliestStartTime*

(3.5) Using the modified domain and problem, generate a branch *B*

(3.6) Construct the distance graph *DB* of *B*

(3.7) *TCB* ← MakeSafePlan(*B*, *DB*)

(3.8) return *TCB*, *DB*

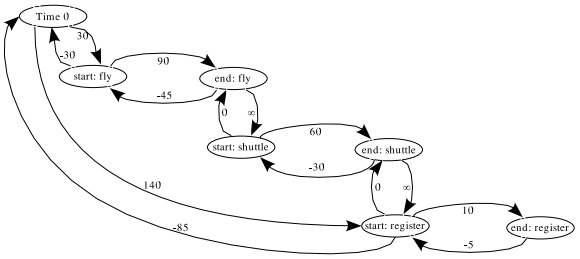
Figure 2: Algorithm for creating TCPs.

assumption, i.e., for each action assign *min-d* as the duration. This yields a plan that is ranked highly by the metric (step 1.1). Next, we analyze the seed plan to find out, temporally speaking, when it becomes unsafe (steps 1.2 and 1.3). At any time point where the seed plan becomes unsafe, we generate and insert a branch that is safe. This technique creates a plan that includes a path that can be safely executed when all of its actions require their maximum duration, but also includes branches yielding a more desirable result that are executed when actions require less time.

As stated above, when generating the seed plan (and subsequent branches), the duration of each action is set at *min-d*, removing all uncertainty at planning time. (For the remainder of this section, the term *plan* is used to refer to either a seed plan or branch plan.) This allows expression of the problem and domain in PDDL2.2 and plans can be generated with any planner that understands PDDL2.2. A plan *P* returned by such a planner will be temporally deterministic. Our algorithm factors in temporal uncertainty by converting *P* to a directed, edge-weighted graph called a *distance graph*, thus expressing *P* as a simple temporal network (STN) [Dechter *et al.*, 1991]. STNs are widely used in temporal reasoning and include nodes representing time points and edges between pairs of nodes representing temporal constraints between time points. Figure 3 shows (a) the seed plan that would be generated for the problem in Figure 1 and (b) the corresponding distance graph. An in-depth description of how to perform step 1.2 of the algorithm is given next.

Execution Time	Action
30	fly_airport2_airport1
76	shuttle_hotel_airport2
107	register_for_conference

(a)



(b)

Figure 3: (a) A seed plan for the problem in Figure 1. Note that the times given by the seed plan assume actions require their minimum durations. (b) The distance graph for the seed plan in *(a)*, incorporating temporal uncertainty. For clarity, only the most important edges are shown.

Since the execution times in *P* are based on a deterministic temporal assumption, they are ignored during the construction of the distance graph *D*. Only the actions of *P* are considered. In construction of *D*, each action is dealt with individually to allow any possible concurrency in *P* to be present in *D*. The first step in constructing *D* is to add two nodes for each action *i*, one for its start time s_i and one for its end time e_i, and a node s_0 representing time 0. Edges are then added in pairs representing temporal relations and weighted with temporal distances. For each action *i*, a pair of edges is added between s_i and e_i. The edge $s_i \rightarrow e_i$ is weighted with *max-d* of *i* and the edge $e_i \rightarrow s_i$ is weighted with -1 × (*min-d* of *i*). Next, pairs of edges are added between s_0 and each s_i node. (Pairs of edges can also be added from s_0 to each e_i, but are not necessary and add no new temporal information.) If an action *i* has a constrained start time, the edge $s_0 \rightarrow s_i$ is weighted with the *latest start time* of *i* and the edge $s_i \rightarrow s_0$ is weighted with -1 × (*earliest start time* of *i*). This is shown with the fly and register actions in Figure 3(b). When an action does not have a constrained start time, the edge $s_0 \rightarrow s_i$ is weighted with ∞ and the edge $s_i \rightarrow s_0$ is weighted with 0, signifying that the start of action *i* comes after time 0, but

there are no other constraints. For clarity, these edges are not included in Figure 3(b).

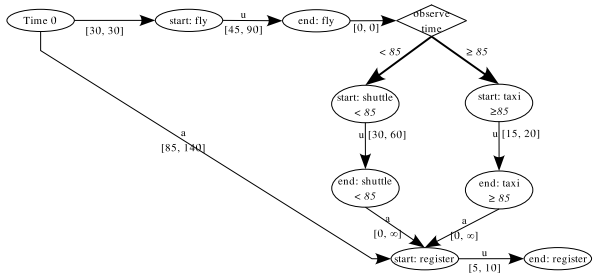
The final step in constructing the distance graph is to insert edges that represent relationships between actions. Though *P* contains a sequence of steps, some concurrency may be possible. To properly discover and encode this in *D*, causal links and threats in *P* must be identified. This is done using an algorithm similar to the one described by [R-Moreno *et al.*, 2002]. For every condition *c* of each action *i*, a causal link is added to the closest action *j* in the plan that appears before *i* and produces *c* as an effect. The causal link forces the producer action to occur before the consumer. A threat link is added between an action *i* and an action *j* when an effect of *j* negates a condition of *i*.[3] This algorithm discovers no knowledge about temporal distance, so pairs of edges labeled with 0 and ∞ are added to the graph simply expressing that one action must occur before the other. There is no concurrency in the plan of Figure 3, so edges are added from the start of each action to the end of the previous action, representing causal links. There are no threats in this example.

Since *D* contains all temporal constraints given in the domain, it can be used to determine when *P* becomes unsafe. This procedure is given by the *MakeSafePlan* function of Figure 2. In [Dechter *et al.*, 1991] it is proved that the absolute bounds on the temporal distance between any two time points represented by nodes *a* and *b* (assuming $a \prec b$) in *D*, is given by the interval [-1 $\times$ *(weight of shortest path from b to a), weight of shortest path from a to b*]. The shortest path can be found using an algorithm such as the *Bellman-Ford* single source shortest path algorithm with a runtime of $O(|V||E|)$. In Figure 3(b) we see that the duration of the fly action is expressed by the interval [45, 90]. However, using the shortest path method (step 2.3 in Figure 2), it is found that the absolute bounds on the duration of the fly action are expressed by the interval [45, 80]. This indicates that if the fly action takes more than 80 time units, the rest of the plan becomes unsafe. To have a safe solution, a contingency must be generated that can reach the goal safely when the fly action takes more than 80 time units, so the loop in step 2.4 will be entered. Currently, the loop considers actions in reverse order. We plan to experiment with different orderings in the future. This loop allows multiple contingency branches to be generated for the same time point. The *GetContingencyBranch* function modifies the domain and problem to reflect the state of the world when the branch occurs and then generates a branch plan which is verified in the same way as the seed plan. Hence, branches may themselves contain branches. After a safe contingency branch has been generated, it is inserted into the seed plan (step 2.7). For the example problem, the contingency branch that will be generated is `taxi_hotel_airport2`, `register_for_conference`. After leaving the loop of 2.4, it is known that action *i* can safely execute, even if it requires its maximum duration. In step 2.9, *D* is modified to ensure that actions occurring before *i* complete early enough so that *i* has enough time to consume its maximum duration if necessary. Once all actions have been verified, the TCP is safe. The TCP of the example problem is given in Figure 4(a). In the next section, we formally explain a data structure that can be used to represent TCPs.

[3] In the future, we plan to extend this algorithm to discover threats that may be caused by actions consuming the same resource. Currently, actions of this sort are disallowed.

```
At time 30: fly_airport2_airport1
IF (time < 85)
   Before time 85: shuttle_hotel_airport2
   Before time 140: register_for_conference
ELSE
   Before time 120: taxi_hotel_airport2
   Before time 140: register_for_conference
```

(a)



(b)

Figure 4: (a) A TCP for the problem in Figure 1. (b) The TCN for the plan in *(a)*.

4 Temporal Contingency Networks

We represent TCPs using a new data structure called a *Temporal Contingency Network* (TCN). TCNs are an extension of STNs and are inspired by the STPU model defined in [Vidal and Fargier, 1999]. TCNs extend STNs in two dimensions. First, interval durations are labeled as user assignable or not; second, some nodes represent decisions based on observations of time. The second aspect enables the representation of TCPs. Part (b) of Figure 4 depicts a TCN for the TCP in part (a).

Formally, a TCN is a quadruple <**T, O, E, B**>. **T** is a set of nodes representing start and end times of actions. A node representing the absolute start time is also included in **T**. Each node in **T** is referred to as a *time point*. Nodes in **T** that are not included in all paths of execution contain a context label [Peot and Smith, 1992] identifying the branch of execution they belong to. Oval nodes in the figure belong to **T**. The shuttle and taxi nodes contain context labels because these actions do not belong to all paths of execution. **O** is a (possibly empty) set of observation nodes representing decisions about which subsequent actions to execute. Observations of time are assumed to be executable at any time (no preconditions) and instantaneous; and should be executed immediately after the preceding time point. A TCN with observation nodes is safe if all the possible paths are safe. In the figure, the diamond represents an observation node. **E** is a set of interval labeled edges representing constraints between time points. Edges in **E** can be marked as uncertain, assignable, or unmarked. The non-bold

edges in the figure belong to **E**. Edges representing assignable durations are marked with *a* and those representing uncertain durations are marked with *u*. Edges with intervals representing an exact amount of time (such as *Time 0 → start: fly*) are unmarked. **B** is a set of temporally labeled edges leaving observation nodes. The bold edges in the figure belong to *B*. As shown, these edges are given a label indicating when each branch can safely be taken. This data structure provides a rich context for reasoning about TCPs.

5 Experiments and Discussion

In this section we provide preliminary experimental results. To the best of our knowledge, there are no planners that prepare contingency branches based on time. We therefore designed our experiments to show that our algorithm works and to help identify the ways in which it can be improved. LPG-*TD* [Gerevini *et al.*, 2004] was the planner that we used for generating seed plans and branches. We choose LPG-*TD* because it can handle the timed initial literals of PDDL2.2 and can optimize for temporal and nontemporal metrics. All experiments were performed on a machine with a 3.0GHz Pentium 4 CPU and 1GB of RAM.

The domain used in the experiments is another version of the domain in Figure 1. The experimental domain has no meal action, but three new actions are added. First, there is a drive action that must occur in order to arrive at *airport1*. Next, actions are added to include the possibility of taking a flight from *airport1* to a new airport, *airport3*, and from *airport3* to *airport2*. The direct flight is more costly than flying through *airport3*. Finally, an action is added to the domain for taking public transportation as the least costly way to get to the hotel from *airport2*. The domain was created in this way to allow many different possible solutions when registering for the conference is the only goal.

The first set of experiments involved plans with no possible parallelism. These experiments were done to compare the runtime as more conditional branches were added to the seed plan. The domain was modified for each run to force different numbers and combinations of branches. The same seed plan was generated every time. Table 1 shows the results of these experiments. In the first run, the seed plan was always safe and thus no branches were created. In successive runs, the number of created branches ranged from 1 to 4. There were 2 different runs that each produced 3 branches. In general, as more branches were created, the runtime increased. However, one of the runs with 3 branches took much longer than the other, and even longer than the run with 4 branches. There are two reasons for this. First, as is clear in the table, generation of the seed plan and branches by LPG-*TD* accounts for nearly all of the runtime. Due to the conditions in this experiment, LPG-*TD* required extra time to create one of the branches. Second, this run contained longer branches than the other run with 3 branches, thus requiring more time for verification time by our algorithm.

The second set of experiments involved parallel plans. In our algorithm, no dependencies are assumed between actions in the plan. As described previously, dependencies are discovered when the distance graph is built. In this way, our algorithm inherently allows parallel plans. To test this aspect, three new actions were added to the domain. The first was an action for grading exams that had to be completed before arriving at the hotel. The other two actions were for reading papers (one for a long paper and one for a short paper) which had to be completed after grading the exams and before arriving at the hotel. More knowledge was gained by reading the long paper. The problem was then modified by adding two goals to read some paper (either or both) and grade the exams. The metric was modified to rank plans higher when more knowledge was gained while still trying to minimize the money spent. The sequential experiments were re-run with the new modified domain and problem. The addition of the parallel actions caused no significant change in runtime among these tests. But, when the domain was modified so that a branch had to be inserted to read the short paper when the exams took a long time to grade, there was a spike in runtime. The reason for this is that our algorithm does not currently identify parallel paths of execution and treat them separately. In the plan that LPG-*TD* created, grading the exams was the first action, reading the paper was the second action, and the rest of the plan followed. In creating a branch to insert after the grading action, our algorithm had to rediscover the entire rest of the plan, though the grading action only directly affected reading the paper. Our algorithm produces a valid result, but it is inefficient. We plan to research this topic in the future.

branches created	LPG-*TD* runtime	total runtime
0	260	260.1
1	520	527.0
2	780	788.5
3	1040	1054.0
3	1290	1302.6
4	1130	1144.9

Table 1: Run times in milliseconds of experiments with problems requiring different numbers of conditional branches.

6 Related work

The main framework of our algorithm is very close to Just-In-Case (JIC) scheduling [Drummond *et al.*, 1994]. The JIC scheduler analyzes a seed schedule, finds possible failure points, and inserts contingency branches so that valuable equipment time is not lost when an experiment fails. Our work extends this framework to multiple planner goals, parallel plans, and nontemporal metrics, but does not consider probability of failure. Presently we insert contingent branches for every action that might not have sufficient time. We plan to improve our algorithm by systematically evaluating and selecting branch insertions points as in [Onder and Pollack, 1999; Dearden *et al.*, 2003]. Dearden et al's [2003] approach involves generating a seed plan and adding contingent plans based on a rich utility metric involving goal values and continuous resources.

There are a number of domain independent planners that can handle durative actions. We used LPG-*TD* because it can optimize based on a nontemporal metric. Other plan-

ners include TGP [Smith and Weld, 1999], a planner that uses mutual exclusion reasoning in a temporal context, SAPA [Do and Kambhampati, 2002], a heuristic forward chaining planner; HSP [Haslum and Geffner, 2002] a heuristic planner with time and resources; and CPT an optimal temporal POCL planner based on constraint programming [Vidal and Geffner, 2004].

Vidal and Fargier [1999] present an analysis of three levels of controllability given a plan. The plans analyzed have actions with uncertain durations and temporal constraints but the start times of actions and the durations of assignable intervals have not been assigned yet. A *control sequence* is an assignment of times to time points and durations to assignable intervals such that all the constraints are respected regardless of the actual durations of the uncertain intervals. A control sequence is *strong* if it can be determined prior to execution. A control sequence is *weak* if (parts of) it can only be determined during execution right after some actual action durations have been observed. Finally, a control sequence is *dynamic* if the completion of an action is too late to make an assignment of start times to the remainder of the actions but there exists a time point t such that if an actual duration is learned in advance at time t safe assignments can be made. In our work we are concerned with generating plans with strong control sequences.

Tsamardinos et al. describe an algorithm for merging existing plans with assignable durations and nontemporal conditional branches [2000]. We plan to extend our algorithm by using their plan merging framework. In particular, we will be generating two plans, one with the minimum duration and one with the maximum duration for each uncertain interval and merging those two plans into a conditional plan such that common actions are executed unconditionally and actions that differ are executed under appropriate contexts.

Tempastic [Younes and Simmons, 2004] is a planner that models continuous time, probabilistic effects, probabilistic exogenous events and both achievement and maintenance goals. It uses a "generate-test-debug" algorithm that generates an initial policy and fixes the policy after analyzing the failure paths. In producing a better plan, the objective is to decrease the probability of failure. Nontemporal resources are not modeled.

Mausam and Weld [2005] describe a planner that can handle actions that are concurrent, durative and probabilistic. They use novel heuristics with sampled real-time dynamic programming in this framework to generate policies that are highly optimal. The quality metric includes makespan but nontemporal resources are not modeled in the planning problem.

Prottle [Little *et al.*, 2005] is a planner that allows concurrent actions that have probabilistic effects and probabilistic effect times. Prottle uses effective planning graph based heuristics to search a probabilistic AND/OR graph consisting of advancement and placement nodes. Prottle's plan metric includes probability of failure but not makespan. Prottle does not model metric resources.

Finally, we would like to mention work from the field of microarchitecture where the objective is to analyze a bottleneck situation which consists of parallel events and determine which events are worthwhile to optimize so that the total makespan of the bottleneck decreases [Fields *et al.*, 2003]. In our future work we will employ techniques from this work to identify portions of a plan that can be optimized rather than to prepare contingent plans as a response to suboptimal execution times.

7 Conclusions and Future Work

We have presented a framework for characterizing and directly dealing with temporal uncertainty. We define temporal uncertainty by assigning actions an interval duration, rather than a single point duration. Our approach is to make an optimistic assumption that all actions complete as quickly as possible. We then generate a plan with inexpensive actions that may become invalid at some point when the optimistic assumption proves wrong. We create more costly contingency plans to be executed only when actions in the inexpensive plan run long enough that an unsafe situation occurs.

Our algorithm is greedy and thus it can return locally optimal solutions which are not globally optimal. In the future, we plan to develop several heuristics to help avoid this problem. One idea is to re-generate the entire plan when a temporally unsafe situation is found. This new plan could be compared to the current plan to determine whether a contingency branch should be added to the current plan or if the new plan should replace the seed plan. In addition to avoiding locally optimal solutions, this approach would generate an appropriate solution in the case that no valid contingency branch existed. The main drawback to this approach is that regenerating the entire plan can be very time consuming. Another way to avoid locally optimal solutions would be to take an MDP based approach where every state in the world contains a time stamp. The naive approach of creating a state representing every possible time point would not be efficient, but improvements could be gained by using states representing intervals of time.

As we continue this work, we plan to extend it in several ways. Our algorithm improves on a strictly conservative approach, but the safe TCPs that it generates may still include missed opportunities. We plan to develop algorithms that can find idle time in a TCP and then insert opportunities as defined by [Fox and Long, 2002]. We would also like to extend our work to be able to handle actions with uncertain effects, including uncertain consumption of nontemporal resources. Finally, we will develop a test bed of problems involving not only our conference domain, but other domains that may benefit from our approach, such as the Rover and Satellite domains.

Acknowledgements

We thank anonymous reviewers for their helpful comments and discussion. Janae N. Foss' research was supported by the Harriett G. Jenkins Predoctoral Fellowship Program and a grant from the Michigan Council of Women in Technology.

References

[Dearden *et al.*, 2003] Richard Dearden, Nicolas Meuleau, Sailesh Ramakrishnan, David Smith, and Richard Washington. Incre-

mental contingency planning. In *ICAPS-03 Workshop on Planning Under Uncertainty*, June 2003.

[Dechter *et al.*, 1991] Rina Dechter, Itay Meiri, and Judea Pearl. Temporal constraint networks. *Artificial Intelligence*, 49:61–95, 1991.

[Do and Kambhampati, 2002] Minh B. Do and Subbarao Kambhampati. Planning graph-based heuristics for cost-sensitive temporal planning. In *Proc. 6th Int. Conf. on AI Planning & Scheduling*, 2002.

[Drummond *et al.*, 1994] M. Drummond, J. Bresina, and K. Swanson. Just-incase scheduling. In *Proc. 12th National Conf. on Artificial Intelligence*, pages 1098–1104, 1994.

[Edelkamp and Hoffman, 2004] Stefan Edelkamp and Jörg Hoffman. PDDL2.2: The language for the classical part of the 4th international planning competition. Technical Report 195, Computer Science Department, University of Freiburg, January 2004.

[Fields *et al.*, 2003] Brian A. Fields, Rastislav Bodik, Mark D. Hill, and Chris J. Newburn. Using interaction costs for microarchitecture bottleneck analysis. In *Proc. 36th international Symposium on Microarchitecture (MICRO-36'03)*, 2003.

[Fox and Long, 2002] Maria Fox and Derek Long. Single-trajectory opportunistic planning under uncertainty. In *Proceedings of the 21st Workshop of the UK Planning and Scheduling Special Interest Group*, November 2002.

[Gerevini *et al.*, 2004] Alfonso Gerevini, Alessandro Saetti, Ivan Serina, and Paolo Toninelli. Planning in PDDL2.2 domains with LPG-*TD*. In *International Planning Competition booklet (ICAPS-04)*, 2004.

[Haslum and Geffner, 2002] Patrik Haslum and Hector Geffner. Heuristic planning with time and resources. In *Proc. 6th European Conf. on Planning*, 2002.

[Little *et al.*, 2005] Iain Little, Douglas Aberdeen, and Sylvie Thiebaux. Prottle: A probabilistic temporal planner. In *Proc. 20th National Conf. on Artificial Intelligence (AAAI-05)*, 2005.

[Mausam and Weld, 2005] Mausam and Daniel S. Weld. Concurrent probabilistic temporal planning. In *Proc. 15th International Conf. on Automated Planning and Scheduling (ICAPS-05)*, 2005.

[Onder and Pollack, 1999] Nilufer Onder and Martha E. Pollack. Conditional, probabilistic planning: A unifying algorithm and effective search control mechanisms. In *Proc. 16th National Conf. on Artificial Intelligence*, pages 577–584, 1999.

[Peot and Smith, 1992] Mark A. Peot and David E. Smith. Conditional nonlinear planning. In *Proc. 1st International Conf. on Artificial Intelligence Planning Systems*, pages 189–197, 1992.

[R-Moreno *et al.*, 2002] M Dolores R-Moreno, Angelo Oddi, Daniel Borrajo, Amedeo Cesta, and Daniel Meziat. Integrating hybrid reasoners for planning and scheduling. In *The Twenty-First Workshop of the UK Planning and Scheduling Special Interest Group*, pages 179–189, 2002.

[Smith and Weld, 1999] David E. Smith and Daniel Weld. Temporal planning with mutual exclusion reasoning. In *Proc. 16th International Joint Conf. on Artificial Intelligence*, 1999.

[Tsamardinos *et al.*, 2000] Ioannis Tsamardinos, Martha E. Pollack, and John F. Horty. Merging plans with quantitative temporal constraints, temporally extended actions, and conditional branches. In *Proc. 5th International Conf. on Artificial Intelligence Planning and Scheduling*, pages 264–272, 2000.

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

[Vidal and Geffner, 2004] Vincent Vidal and Hector Geffner. Branching and pruning: An optimal temporal POCL planner based on constraint programming. In *Proc. 19th National Conf. on Artificial Intelligence*, pages 570–577, 2004.

[Younes and Simmons, 2004] Hakan L.S. Younes and Reid G. Simmons. Solving generalized semi-markov decision processes using continuous phase-type distributions. In *Proc. 19th National Conf. on Artificial Intelligence (AAAI-04)*, 2004.