Planning for Concurrent Action Executions Under Action Duration Uncertainty Using Dynamically Generated Bayesian Networks

Eric Beaudry and **Froduald Kabanza** and **Francois Michaud**
Universite de Sherbrooke
Sherbrooke, Quebec, Canada
eric.beaudry@usherbrooke.ca, kabanza@usherbrooke.ca, francois.michaud@usherbrooke.ca

Abstract

An interesting class of planning domains, including planning for daily activities of Mars rovers, involves achievement of goals with time constraints and concurrent actions with probabilistic durations. Current probabilistic approaches, which rely on a discrete time model, introduce a blow up in the search state-space when the two factors of action concurrency and action duration uncertainty are combined. Simulation-based and sampling probabilistic planning approaches would cope with this state explosion by avoiding storing all the explored states in memory, but they remain approximate solution approaches. In this paper, we present an alternative approach relying on a continuous time model which avoids the state explosion caused by time stamping in the presence of action concurrency and action duration uncertainty. Time is represented as a continuous random variable. The dependency between state time variables is conveyed by a Bayesian network, which is dynamically generated by a state-based forward-chaining search based on the action descriptions. A generated plan is characterized by a probability of satisfying a goal. The evaluation of this probability is done by making a query the Bayesian network.

Introduction

Concurrent actions and time uncertainty - that is, uncertainty related to the duration of actions and the occurrence of their effects - are among the key factors of complexity in a number of planning domains. They are in particular important in the Mars rovers planning domain which deals with the decision of daily activities for the rovers (Bresina et al. 2002). A rover may be navigating to a given point to collect samples, while at the same time initializing its sensors, or transmitting data of previously analyzed samples. In general the time at which the different effects of an operation occur is not predetermined. Navigation speed, for instance, is not constant.

Time uncertainty can in principle be dealt with using current probabilistic planning approaches. In particular, Concurrent Markov Decision Processes (CoMDP) (Mausam and Weld 2004) and Concurrent Probabilistic Temporal Planning (CPTP) (Mausam and Weld 2006) were designed to deal with domains involving actions with probabilistic effects and actions concurrency. These approaches rely on a discrete time model, so that the search space is a space of world states extended with numeric time stamps. An action having an effect with time uncertainty would be modeled by nondeterministic effects, as many as the possible time points at which the effect may occur. The application of an action to a world state would result in each nondeterministic effect yielding a corresponding time stamped state, resulting in a combinatorial explosion in the search space.

We are currently investigating an alternative planning approach avoiding the combinatorial state explosion incurred by the time stamping of states. We introduce a new approach addressing uncertainty related to the duration of actions or the occurrence of their effects. The search space is still modeled as a space of world states, but time is modeled using a set of continuous random variables instead of numeric time stamps. Time random variables are associated to features of states to track the time at which the assigned values become valid. A Bayesian network is used to maintain the dependency relationship between the time random variables. It is dynamically generated by the forward-chaining state-space search via the application of actions to world states. The search process queries the Bayesian network to determine the probability that the goal is achieved in a given state of the state-space explored so far. It also estimates the expected makespan of a plan. As a first step, our planner produces non-conditional plans which are robust to uncertainty.

The attachment of time random variables to state features and the inference in the Bayesian network to answer queries by the search process introduce a new overhead. On the other hand, we no longer have time stamps for all uncertain time points for action effects. Our hypothesis is that, for many interesting domains, the savings obtained by the elimination of the combinatorial state explosion due to time stamping outweighs the overhead incurred by the time tracking of state features. So far we validated this hypothesis on the International Planning Competition (IPC) Transport and Mars rovers domains, modified to introduce uncertainty on action durations.

In the next section, we introduce basic concepts and definitions necessary for the description of the planning algorithm. This is followed by the planning algorithm, experiments, related work discussion, and a conclusion.

Basic Concepts and Definitions

State variables are used to describe state features (Nebel 2000). The set $X = \{x_1, ..., x_n\}$ defines all state variables, each one describing a particular feature of the world state. A world state is an assignment of values to the state variables and action effects (state updates) are changes of variable values. A state variable with a corresponding value is also called a state feature. The domain of values for a variable x is noted $Dom(x)$. It is assumed that no exogenous events are present; only planned actions cause state changes.

There are two kinds of continuous random variables for representing uncertainty on time: time events (T) and action durations (D). A **time random variable** $t \in T$ marks the occurrence of an event, occurring either at the start or at the end of an action. An event represents a change of the values of a set of state variables. The time random variable $t_0 \in T$ is reserved for the time of the initial world state. Each action a has a duration represented by a random variable $d_a \in D$. Each time random variable $t \in T$ has an equation which defines the time at which t occurs as a function of other random variables. For instance, consider an action a starting at time t_0 and ending at time t_1. Then, t_1 is defined by the equation $t_1 = t_0 + d_a$.

The set of **action duration random variables** is defined by $D = \{d_a | a \in A\}$ where A is the set of actions. The function $PDF_{d_a}(u) : \mathbb{R}^+ \to \mathbb{R}^+$ gives the probability density that the action a has a duration of u units of time. We make the assumption that actions have independent durations. So d_a and d_b are independent random variables if and only if $a \neq b$. Several executions of a same action are considered to have the same duration. Executing action a twice has the total duration $2d_a$.

Hence a state is not associated with a fixed time stamp. It rather describes the current world state using a set of state features, that is, a set of value assignations for all state variables X. There is no uncertainty about the values being assigned to state variables. The only uncertainty is about **when** the assigned values of state variables become valid. The function $V(x)$ models this uncertainty by mapping each state variable to a corresponding time random variable.

The specification of actions follows the extensions introduced into PDDL 2.1 (Fox and Long 2003) for expressing temporal planning domains. Roughly, an **action** a is a tuple a=($name$, $cstart$, $coverall$, $estart$, $eend$, d_a) where $cstart$ is the set of *at start* conditions that must be satisfied at the beginning of the action, $coverall$ is the set of persistence conditions that must be satisfied *over all* the duration of the action, and $estart$ and $eend$ are respectively the sets of effects *at start* and *at end* of the action. The duration of the action is probabilistically specified by the random variable $d_a \in D$. The condition c is a boolean expression over state variables. The expression $vars(c)$ returns the set of all state variables that are referenced by a condition c. An effect $e = (x, exp)$ specifies the assignation of the value resulting from the evaluation of expression exp to the state variable x. Expressions $conds(a)$ and $effects(a)$ return respectively all conditions and all effects of action a. The set of all actions is denoted by A.

An action a is applicable in a state s if and only if s satisfies all *at start* and *over all* conditions of a (denoted $s \models a$). A condition $c \in conds(a)$ is satisfied in state s if c is satisfied by the current assigned values of state variables of s.

A **state** s is defined by $s = (U, V, P)$ where U is a total mapping function $U(x) : X \to Dom(x)$ that gives the current assigned value for each state variable x; V is a total mapping function $V(x) : X \to T$ that gives the time at which the assignation $x = U(x)$ becomes valid; and P a total mapping function $P(x) : X \to 2^T$ that gives the set of time random variables associated to persistence conditions on x. Persistence conditions are used to track *over all* conditions of actions. Each time random variable $t \in P(x)$ imposes that state variable x cannot be changed before time t. Time $t \in P(x)$ is also called the release time of a persistence condition on x. A state variable x has always an implicit persistence condition until its corresponding valid time, i.e. $V(x) \in P(x)$.

A **goal** G is a conjunction of timed goal state features. A timed goal state feature $g = (x, v, t)$ means that state variable x has to be assigned the value v within $t \in \mathbb{R}^+$ time. We use $t = +\infty$ to specify the absence of a deadline for goal achievement. A planning problem is defined by (A, s_0, G, m) where A is the domain definition (set of actions in PDDL), s_0 is the initial world state, G is the goal to reach, and m is a metric function to optimize.

Consider the Transport planning domain in which trucks have to deliver packages distributed over a map. We have a set of n trucks $R = \{r_1, ..., r_n\}$, a set of m packages $P = \{p_1, ..., p_m\}$ and a map of k locations $L = \{l_1, ..., l_k\}$. A package is either at a location or loaded onto a truck. There is no limit on the number of packages a truck can transport at the same time and on the number of trucks that can be parked at the same location. The specification of actions is given in Table 1. The action $Goto(r_i, l_a, l_b)$ describes the moving of a truck r_i from location l_a to location l_b. The duration of a *Goto* action is modeled using a normal distribution where both the mean and the standard deviation are proportional to the distance to be traveled. *Load* and *Unload* actions specify the loading and the unloading of a package by a truck. The duration of these actions is defined by a uniform distribution. The set of state variables $X = \{C[r] | r \in R\} \cup \{C[p] | p \in P\}$ specifies the current location of trucks and packages. The domain of variables is defined as $Dom(C[r]) = L$ ($\forall r \in R$) and $Dom(C[p]) = L \cup R$ ($\forall p \in P$). A goal G is a conjunction of n subgoals (p_k, l_k, dt_k) for $0 < k \leq n$ where each subgoal specifies that package $p_k \in P$ must be delivered to location $l_k \in L$ before due time $dt_k \in \mathbb{R}^+$.

Planning Algorithm

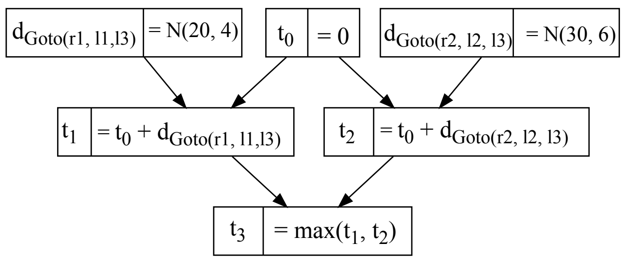
The planning algorithm performs a forward-chaining search in a space of states. The state-space explored at a given point is a graph, with nodes corresponding to states and transitions to actions. In addition to the search graph, the planner generates a Bayesian network to track the dependency relationship between the random variables which define the time of events and the duration of actions. This is a directed acyclic graph $B = (N, E)$ where $N = T \cup U$ is a set of random variables and E is a set of edges representing dependencies

$Goto(r_i, l_a, l_b)$	
$cstart$	$CurrentLocation[r_i] = l_a$
$eend$	$CurrentLocation[r_i] = l_b$
$duration$	$Normal(dist/speed, 0.2 * dist/speed)$
$Load(r_i, p_j, l_k)$	
$cstart$	$CurrentLocation[p_j] = l_k$
$coverall$	$CurrentLocation[r_i] = l_k$
$eend$	$CurrentLocation[p_j] = r_i$
$duration$	$Uniform(30, 60)$
$Unload(r_i, p_j, l_k)$	
$cstart$	$CurrentLocation[p_j] = r_i$
$coverall$	$CurrentLocation[r_i] = l_k$
$eend$	$CurrentLocation[p_j] = l_k$
$duration$	$Uniform(30, 60)$

Table 1: Actions specification of Transport domain

between random variables. Dependencies in the Bayesian network follow from the equations of the time random variables, which are derived by the application of an action to a current state.

Figure 1 illustrates an example of a Bayesian network composed of six random variables where the initial time is denoted by $t_0 = 0$. Random variables $d_{Goto(r_1,l_1,l_3)}$ and $d_{Goto(r_2,l_2,l_3)}$ follow a normal distribution and represent the duration of the corresponding actions. The time random variables t_1 and t_2 specify the end time of two actions started at t_0. The time random variable $t_3 = max(t_1, t_2)$ defines the earliest time at which both actions will be completed.



d_Goto(r1, l1,l3) = N(20, 4)
t0 = 0
d_Goto(r2, l2, l3) = N(30, 6)
t1 = t0 + d_Goto(r1, l1,l3)
t2 = t0 + d_Goto(r2, l2, l3)
t3 = max(t1, t2)

Figure 1: Sample Bayesian Network

The planning algorithm handles concurrency and delayed effects differently from a traditional model for concurrency (Bacchus and Ady 2001). A delayed effect for an action is one specified to hold at a given point of time after the execution of an action. In a traditional implementation, time is associated to states in the state-space. We have transitions along which time freezes to interleave simultaneous actions; and transitions updating the time stamp. The search process manages delayed effects by registering them in an event queue attached to states. A special advance time action activates the delayed effects whenever appropriate.

In our approach, time is not attached to state. It is rather directly attached to state features; therefore there is no need of a delayed effect queue and the special advance time action. Time increment is tracked by the time variables attached to time features. A time variable for a feature is updated by the application of an action only if the effect of the action changes the feature; the update reflects the delayed effect on the feature.

Algorithm 1 shows the entry point of the planning algorithm. The planner searches for a plan which, when executed has a probability of success higher than a given threshold, and which optimizes the metric m. The choice of an action a at Line 5 is a backtrack choice point. Heuristics are involved to guide the choice.

Algorithm 1 Plan

```
1. PLAN($s$, $G$, $A$, $m$)
2.    if $Pb(s \models G) \geq threshold$
3.       $\pi \leftarrow$ ExtractPlan($s$)
4.       return $\pi$
5.    nondeterministically choose $a \in \{a \in A \mid a \models s\}$
6.    $s' \leftarrow$ Apply($s$, $a$)
7.    return Plan($s'$, $G$, $A$, $m$)
```

Algorithm 2 Apply Action Function

```
1. function APPLY($s$, $a$)
2.    $s' \leftarrow s$
3.    $t_{conds} \leftarrow \max_{x \in vars(conds(a))} s.V(x)$
4.    $t_{release} \leftarrow \max_{x \in vars(effects(a))} \max(s.P(x))$
5.    $t_{start} \leftarrow \max(t_{conds}, t_{release})$
6.    $t_{end} \leftarrow t_{start} + d_a$
7.    for each $c \in a.coverall$
8.       for each $v \in vars(c)$
9.          $s'.P(x) \leftarrow s'.P(x) \cup \{t_{end}\}$
10.   for each $e \in a.estart$
11.      $s'.U(e.x) \leftarrow eval(e.exp)$
12.      $s'.V(e.x) \leftarrow t_{start}$
13.      $s'.P(e.x) \leftarrow \{t_{start}\}$
14.   for each $e \in a.eend$
15.      $s'.U(e.x) \leftarrow eval(e.exp)$
16.      $s'.V(e.x) \leftarrow t_{end}$
17.      $s'.P(e.x) \leftarrow \{t_{end}\}$
18.   returns $s'$
```

The Apply function of Algorithm 2 details how a resulting state s' is obtained by the application of an action a in a state s. The start time of an action is defined as the earliest time at which its requirements are satisfied in the current state. Line 3 builds the time t_{conds} which is the earliest time at which all *at start* and *over all* conditions are satisfied. This time corresponds to the maximum of all time random variables associated to the state variables referenced in the action's conditions. Line 4 builds time $t_{release}$ which is the earliest time at which all persistence conditions are released on all state variables modified by an effect. Then at Line 5, the time random variable t_{start} is created. Indeed, its equation is the max of all time random variables collected in Lines 3-4. Line 6 creates the time random variable t_{end} with the equation $t_{end} = t_{start} + d_a$. Once created, the time random variables t_{start} and t_{end} are added to the Bayesian network if they do not already exist. Lines 7-9 add a persistence condition which expires at t_{end} for each state variable involved in an *over all* condition. Lines 10-17 process *at start* and *at end* effects. For each effect on a state variable, they assign this state variable a new value, set the valid time to t_{start} and add t_{end} to the set of persistence conditions.

Figure 2 illustrates an example of a partial search carried out by Algorithm 1. Expanded states are shown in (a). Each state has three state variables which represent respectively the current location of trucks r_1 and r_2 and of package p_1. All state variables of the initial state s_0 are associated with the time random variable t_0 which represents the initial time. Subfigure (b) shows the generated Bayesian network; it shows the equations of time random variables and the probability distributions followed by the action duration random variables.

The state s_1 is obtained by applying the action $\mathit{Goto}(r_1, l_1, l_3)$ to state s_0. The Apply function (see Algorithm 2) works as follows. The action $\mathit{Goto}(r_1, l_1, l_3)$ has the *at start* condition $C[r_1] = l_1$. Because $C[r_1]$ is associated to t_0, we have $t_{conds} = \max(t_0) = t_0$. Since the action modifies $C[r_1]$, which is associated to $t_{release}$, time $t_{release} = \max(t_0) = t_0$. At Line 5, the action start time is thus defined as $t_{start} = \max(t_{conds}, t_{release}) = t_0$ which already exists. Then at Line 6, the time random variable $t_{end} = t_0 + d_{Goto(r_1,l_1,l_3)}$ is created and added to the network with the label t_1. Next, Lines 13-16 apply effects by performing the assignation $C[r_1] = l_3$ and by setting time t_1 as the valid time for $C[r_1]$.

Applying action $\mathit{Unload}(r_1, l_3, p_1)$ in state s_3 creates state s_4. The action has two preconditions $C[r_1] = l_3$ and $C[p_1] = r_1$, which are respectively associated to t_1 and t_0. This, the start time of this action is $\max(t_0, t_1)$ which could be simplified to t_1 because t_0 is an ancestor of t_1. The end time is specified by $t_3 = t_1 + d_{Unload(r_1,l_3,p_1)}$. Since the action has an *over all* condition, a persistence condition is added on the state variable $C[r_1]$, which must hold until the end time t_3 (noted within [] in the figure). The action $\mathit{Load}(r_2, l_3, p_1)$ has two *at start* conditions: $C[r_2] = l_3$ and $C[p_1] = l_3$. Since state variables $C[r_2]$ and $C[p_1]$ are valid at times t_2 and t_3 respectively, the action start time is defined by a new time random variable $t_4 = \max(t_2, t_3)$.

Bayesian Network Inference Algorithm

A Bayesian network inference algorithm is required to estimate the probability of success and the expected makespan of plans. The choice of an inference algorithm for Bayesian networks is guided by the structure of the Bayesian network and on the type of the random variables it includes (Darwiche 2009). In our case, the Bayesian network has continuous random variables. In this case, analytical inference methods are possible if one can impose some restrictions on the allowed probability distributions. In particular, a normal distribution is often used because its definition by two parameters (mean μ and standard deviation σ) makes it suitable for analytical approaches.

In our approach, the time random variables (T) cannot be constrained to follow normal distributions since their equations may contain several instances of the maximum operator. Even if two random variables a and b were normally distributed, the resulting random variable would not be a normal distribution because of the manipulations involved in the equations defining variables in our approach.

Our approach leads to arbitrary forms of probabilistic distributions. Because there exists no exact and analytic method for Bayesian networks having arbitrary types of distribution, approximate inference algorithms have to be used. For this reason, we use a direct sampling algorithm for the Bayesian network inferences (Darwiche 2009).

Using a Bayesian network inference algorithm introduces an overhead. The direct sampling algorithm runs in $O(n \cdot m)$ where $n = |T| + |D|$ is the number of nodes in the Bayesian network, and m is the number of samples generated. The estimation error is inversely proportional to the squared root of the number of generated samples m.

Incremental Belief Evaluation The Bayesian network is dynamically built during the search process. Its extensions are made with Algorithm 2. Once a new time random variable is created (see t_{start} and t_{end} in the algorithm), it is added to the Bayesian network and its belief is immediately estimated. The belief on time random variables is required by the heuristic function and to estimate the probability that a plan satisfies time constraints.

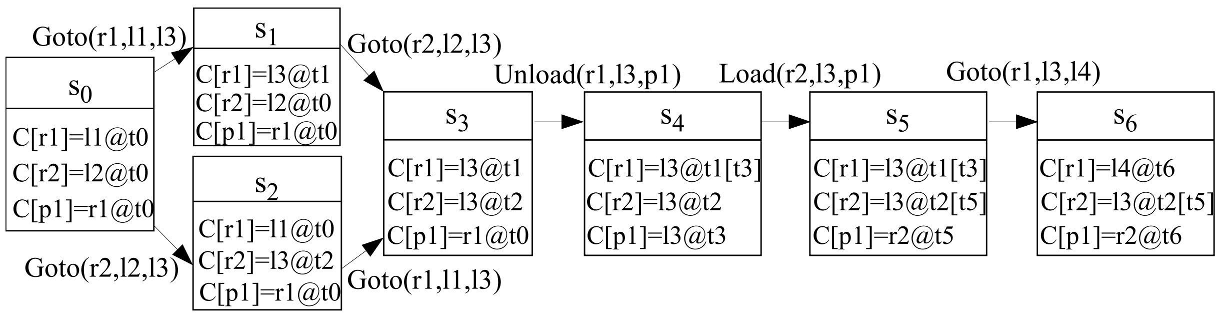
Because the Bayesian network is dynamically generated, we want to avoid evaluating the whole Bayesian network each time a new random variable is added. In the worst case, adopting this strategy would indeed require $n(n-1)/2$ evaluations for a network of n nodes. To reduce computation time, the generated samples are kept into memory into an array for each random variable. The i^{th} samples of all random variables correspond to a simulation of the whole Bayesian network. When adding a new random variable, new samples are generated by considering the samples of the parent variables. Thus the computation cost of incremental evaluation of a Bayesian network is equivalent to one evaluation of the entire network.

For small networks, keeping all samples in memory may not be a problem. However, this is not the case for larger networks. Therefore, we rather adopt a caching mechanism that keeps the generated samples of the most recently accessed random variables. This strategy offers a good trade-off between efficiency and memory requirement.

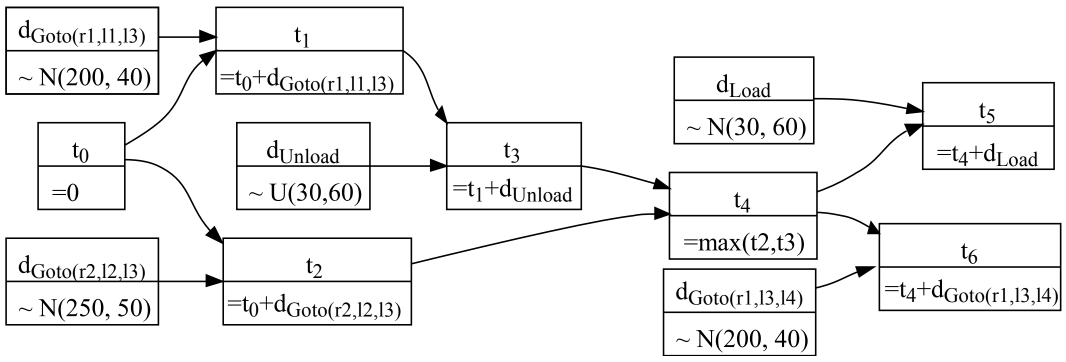
Heuristics

In order to be efficient, forward-chaining search based planners require heuristics. For instance, the Fast-Forward (FF) planner introduced a domain independent and admissible heuristic called Relaxed GraphPlan (RGP) (Hoffmann and Nebel 2001). Basically, RGP generates a relaxed plan using a simplified GraphPlan algorithm which ignores delete (negative) effects. The extracted plan is used to estimate a lower bound on the number of actions (or the remaining cost) required to reach the goal. This heuristic has been adapted for state-based temporal planners (Do and Kambhampati 2003).

We adapted the RGP heuristic to domains having action concurrency and time uncertainty. Because our planner is based on a state-variable representation, instead of tracking minimal appearing time (cost) of each literal, the minimal



(a) State space



(b) Bayesian network

Figure 2: Sample Search

valid time (cost) of each possible assignment of each state variable is tracked. This is stored into the mapping function $M_c(x, v) : (X, Dom(x)) \to \mathbb{R}$. Algorithm 3 describes the heuristic function which estimates a lower bound of the expected cost of a plan generated by a forward-search from a state s.

Line 2 initializes the M_c function to $+\infty$ over its domain. Then at Lines 3-4, the minimum cost for the current value of each state variable is set to its valid time. Line 6 loops on all possible actions $a \in A$. If an action can reduce the minimum cost of an assignment $x = v$ then $M_c(x, v)$ is updated. The loop of Line 5 performs updates until there is no minimum cost reduction or until the goal is satisfied. The SatisfiedCost function returns a lower bound on the cost required to satisfy G by considering M_c.

Algorithm 3 Evaluate Heuristic Function

```
1.  function EVALUATE_HEURISTIC($s$, $G$)
2.     $M_c(.,.) \leftarrow +\infty$
3.     for each $x \in X$
4.        $M_c(x, A(x)) \leftarrow T(x)$
5.     while SatisfiedCost($M_c$, $G$)$= +\infty$
6.        for each $a \in A$
7.           $s \leftarrow$ build a state $s$ from $M_c \Rightarrow a \models s$
8.           $s' \leftarrow$ Apply($s, a$)
9.           for each $x \in vars(effects(a))$
10.             $M_c(x, A(x)) \leftarrow min(M_c(x, A(x)), s'.T(x))$
11.    return SatisfiedCost($M_c$, $G$)
```

To speed up the evaluation of the heuristic function in Algorithm 3, random variables are not manipulated. All calculations are done using scalar values which are initialized to the expected value of their corresponding random variables in the Bayesian network. Because time is probabilistic, the makespan (or other cost criteria) of a plan is also probabilistic. Definition 1 revises the notion of admissibility of a heuristic defined in a domain with time uncertainty.

Definition 1 A heuristic is **probabilistically admissible** if and only if it does not overestimate the expected cost of the optimal plan to reach the goal.

Theorem 1 The heuristic function presented in Algorithm 3 is probabilistically admissible when random variables are replaced by their expected values.

Proof. The Algorithm 3 is an adaption of RGP heuristic for the state-variable representation. For fully deterministic planning domains, it has been proven that this kind of heuristic never overestimates the optimal cost (Hoffmann and Nebel 2001; Do and Kambhampati 2003). Because it ignores delete (negative) effects, the generated relaxed plan is a sub-set of the optimal plan.

Now consider time uncertainty. As previously said, the heuristic function is evaluated using scalar values rather than random variables. We have to make sure that this relaxation never overestimates the expected cost of an optimal plan. Basically, time random variables (T) are expressed using two operators: (1) the sum operator to compute the end time of an action, and (2) the $\max$ operator to compute the start time of an action. When a sequence of actions a_1, ..., a_n is planned, a time random variable $t_e(a_n)$ is created with an equation equivalent to $t_e(a_n) = D_{a_1} + ... + D_{a_n}$. The expected sum of the duration of the actions is given by $E[t_e(a_n)] = E[D_{a_1} + ... + D_{a_n}] = E[D_{a_1}] + ... + E[D_{a_n}]$. The evaluation of the heuristic function returns the exact same duration because it sums up the expected values of each duration: $Heuristic(t_e(a_n)) = E[D_{a_1}] + ... + E[D_{a_n}]$. Thus the evaluation of time random variables based on a sum is correct.

When an action b requires the completion of other actions a_1, ..., a_n, the start time of the action b is defined by a random variable $t_s(b) = \max(a_1, ..., a_n)$. Here the evaluation of the heuristic cannot overestimate the real expected value since $Heuristic(t_s(b)) = \max(E[D_{a_1}], ..., E[D_{a_n}]) \leq E[t_s(b)] = E[\max(D_{a_1}, ..., D_{a_n})]$ (Jensen's inequality). Because relaxing calculation does not overestimate the expected value, the heuristic is probabilistically admissible. □

Finding Equivalent Time Random Variables

A fundamental step in a forward-chaining search algorithm in a state-space is the ability to test whether a state s has already been visited. Recall that two states are equivalent if their mapping functions U, V, P are the same. The functions V and P involve time random variables. Two time random variables are equivalent if their equations are the same. However, two time random variables could also be equivalent even if their equations are not the same.

Consider a situation where two actions a_1 and a_2 cannot be executed concurrently, and thus have to be executed sequentially. Time t_0 is the initial time. Executing a_1 and a_2 successively leads to times t_1 and t_3. The reverse order leads to times t_2 and t_4. Figure 3 shows the Bayesian network for this situation.

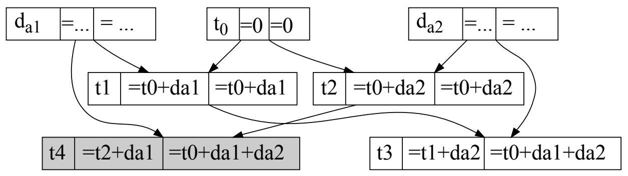


Figure 3: Sample Bayesian Network

To identify equivalent time random variables, equations could be rewritten into another form using only random variables having no dependencies to other time random variables. Examples of this alternate representation are displayed in the third block of each random variable rectangle in Figure 3. By normalizing the way equations are written (e.g. by sorting terms) equation comparison can be made more efficient. Because the rewritten equations of t_3 and t_4 are the same, the variable t_4 can be replaced by t_3.

Empirical Results

We experimented our planning algorithm on two planning domains inspired from the International Planning Competition (IPC). To this end, we adapted the Transport and Rovers domains by introducing uncertainty on the duration of actions. The Transport domain is presented in Table 1. The Rovers domain models problems where many rovers must acquire data using a special sensor at different sites of interests (SOI) within time-window constraints. Then, the sensed data has to be transmitted to the base station using a wireless link. The experiments were made on a computer with an Intel Core 2 Quad 2.4 GHz and 2 GB of RAM.

As previously said, to deal with time uncertainty, our planning approach is based on a continuous time model: random variables are used to represent time. This avoids the state-space explosion of discrete time model based approaches. However, our approach introduces an overhead because it uses a Bayesian network to estimate the belief of the random variables. Our hypothesis is that this overhead outweighs the state-space explosion caused by the use of a discrete time stamp in states.

To validate this hypothesis, a comparison has to be made with a discrete time-based planner. Since CPTP (Mausam and Weld 2006) planner is not available , we decided to implement our own discrete time-based planner. Because implementing a full featured CPTP planner would require too much effort, we have implemented a CoMDP-based planner which uses a discrete time model. The planning of simultaneous actions is achieved by combining several actions together (Boutilier 1996; Mausam and Weld 2004).

Table 2 reports the empirical results. The first and second columns show the size of problems, expressed in terms of the number of trucks and packages for the Transport domain and the number of rovers and SOI for the Rovers domain. The columns under our planner detail the number of states generated, the number of random variables added to the Bayesian Network, the number of evaluations of random variables, the CPU time (in seconds) and the estimated expected makespan of plans. The absolute error on the estimated expected makespan is bellow 3 units of time for a 95 % confidence level. For estimating the belief of a random variable of the Bayesian Network, 5000 samples are generated. We keep arrays of samples in memory for at most 1000 random variables. Columns under Concurrent MDP give the number of states, the expected makespan and the CPU time. These experiments validate our hypothesis that the overhead of managing random variables is largely compensated by the state-space reduction incurred.Indeed, our approach avoids the state-space blow up caused by having different time stamps for each duration unit. A few tests failed because they ran out of memory or reached the allowed CPU time limit (180 seconds).

Related Work

The framework of Markov Decision Process (MDP) (Bellman 1957) provides a basis for many probabilistic artificial intelligence (AI) planning frameworks. In its basic formulation, MDP naturally fits in with the process of making sequential decisions under uncertainty. The Concurrent MDP (CoMDP) (Mausam and Weld 2004) approach is an extension of the MDP framework designed to deal with concurrent actions, by using MDP transitions corresponding to sets

Transport Problem		Our Planner					Discrete Time Planner		
Trucks	Packages	States	RanVars	Evals	CPU	E[Makespan]	States	CPU	E[Makespan]
1	2	71	118	118	0.150	1435	21,349	3.43	1,435
1	3	220	320	320	0.257	1871	127,964	42.2	1,871
2	2	216	179	179	0.255	986	21,459	2.11	986
2	3	237	170	170	0.238	987	451,363	220	1,021
2	4	24,772	7,810	11,276	9.06	1,666	Fail		
3	3	673	254	254	0.399	984	266,617	162	984
3	6	61,665	6,582	7,818	12.66	1,340	Fail		
4	6	Fail					Fail		
4	8	76,106	6253	7197	8.3	1,255	Fail		

Rover Problem		Our Planner					Discrete Time Planner		
Rovers	Sites	States	RanVars	Evals	CPU	E[Makespan]	States	CPU	E[Makespan]
1	2	50	49	49	0.073	1,653	18,297	3.42	1,653
1	3	109	80	80	0.111	1,966	142,835	50.3	1,966
1	4	330	154	154	0.300	2,666	51,765	239	2,666
1	5	406	156	156	0.353	2,892	Fail		
2	4	6,018	431	431	1.55	1,963	341,937	186	2,013
2	6	12,445	423	423	9.72	2,515	Fail		
2	8	85,830	1,312	1,363	224	4,571	Fail		

Table 2: Empirical results for Transport and Rovers domains

of combined (simultaneous) actions. The combined effect of a simultaneous set of actions is obtained based upon a conflict analysis of the actions. This results in a potential exponential explosion, given that the planner has to consider all possible action combination subsets in each state. One of the main contributions of CoMDP consists in pruning strategies to avoid exploring action combinations naively.

Another limitation of CoMDP is that a set of simultaneous actions is considered as a macro-action having unique time duration. Consequently, model actions with interleaved effects cannot be modeled unless the smallest possible time unit that can separate any two effects in the domain is adopted. This, however, would exacerbate the state explosion.

The problems caused by considering simultaneous sets of actions as macro-actions are solved by adopting an interwoven epoch search space, similar to (Bacchus and Ady 2001). Actions have delayed effects which are stored in the states' event queue. Introduced in the CoMDP framework, this yielded the Concurrent Probabilistic Temporal Planning (CPTP) (Mausam and Weld 2006). Although the CPTP approach produces plans with smaller makespan than CoMDP, it still leaves a huge state-space because of the discrete time representation.

Planning with concurrent actions under time uncertainty can be done using the Generate, Test, and Debug (GTD) framework (Younes and Simmons 2004). The idea is to invoke a deterministic planner to obtain a deterministic plan. The generated plan is then simulated to identify potential failure points before execution. The plan is then robustified by generating contingencies (once again using the deterministic planner) to patch failure points [1].

The plans simulation meant to identify failures points involves Monte Carlo simulations. Once the failure points have been identified, many strategies can be used to robustify the plan. In general, finding the best insertion point for plan robustification is an NP-hard decision, so heuristic strategies must be used. A naive approach consists of creating a conditional branch at each identified failure point. This, however, has little benefit, given that the failure may only be a reflection of a decision made earlier on the current execution path. The insertion points can be randomly chosen (Younes and Simmons 2004) or selected using planning graph analysis (Dearden et al. 2003).

One advantage of these approaches over CoMDP and CPTP is that they can model time and resources as continuous quantities and are thus not limited to discrete ones. This reduces the number of states and requires significantly less memory. The disadvantages are that these approaches do not guarantee completeness (Mausam and Weld 2006).

The Factored Policy Gradient (FPG) (Buffet and Aberdeen 2009) is another planning approach that deals with both concurrency and uncertainty. It is based on policy-gradient methods borrowed from reinforcement learning (Sutton et al. 1999). To reduce computation efforts for solving a MDP, policy-gradient methods introduce an approximation function to estimate states' value during policy generation. Gradients are evaluated using Monte Carlo simulations. To enable concurrency, FPG generates a global factorized policy composed by local policies, each one controlling a specific action.

There exist other approaches including Prottle (Lit-

[1]The expression "plan robustification" is from (Drummond and Bresina 1990). They introduced a planning approach for incrementally making a plan more robust by adding branches to deal with contingencies. Their approach is reminiscent of the approach outlined here, albeit they relied on backtracking search to identify points worth robustifying rather than simulating plan executions.

tle, Aberdeen, and Thiebaux 2005) which is based Labeled Real-Time Dynamic Programming (LRTDP) and GSMDP (Rachelson et al. 2008).

Conclusion and Future Work

This paper presents a new planning approach that extends the forward-chaining search for dealing with action concurrency under time uncertainty. Our main contribution is the introduction of a new state representation based on a continuous time model instead of a discrete one. Rather than representing time with a numeric time stamp attached to each state, we associate each state variable to a continuous time random variable. Random variables define the start and the end times of actions, and their duration. The random variables are organized into a Bayesian network. A direct sampling algorithm is used to estimate the probability of success and the expected makespan of plans. We also adapted the Relaxed Graph Plan heuristic for domains with time uncertainty. Empirical experiments on probabilistic versions of the Transport and Rovers domains show that our planner is able to deal efficiently with actions concurrency under time uncertainty.

Our approach is orthogonal to GTD in the sense that our focus is not the generation of contingencies to improve plans. While GTD relies on a deterministic planner which ignores uncertainty, our planner considers uncertainty during the search process. As future work, it could be interesting to combine both approaches. The GTD framework first finds a weak plan (with low probability of success) and then inserts conditional branches to robustify it. We believe that our planner could be used in a GTD approach instead of a deterministic planner. The framework would first compute a strong plan (high probability of success) and add conditional branches to reduce the expected makespan. Each conditional branch would capture opportunities arising if tasks are executed faster than expected. This extension will enable direct comparisons with other approaches having contingency.

Although we dealt with time uncertainty, our approach may be generalized to resources uncertainty by adding other random variables to represent resources. For instance, it could be possible to add uncertainty on the amount of fuel consumed by the trucks and rovers. Future work will also include dynamic programming and over MDP techniques to evaluate the quality of plans.

Acknowledgements

The authors gratefully acknowledge the contribution of the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Fonds qubcois de la recherche sur la nature et les technologies (FQRNT).

References

Bacchus, F., and Ady, M. 2001. Planning with resources and concurrency: a forward chaining approach. In *Proc. International Joint Conference on Artificial Intelligence*, 417–424.

Bellman, R. E. 1957. *Dynamic Programming*. Princeton University Press, Princeton, NJ.

Boutilier, C. 1996. Planning, learning and coordination in multiagent decision processes. In *Proc. Conference on Theoretical Aspects of Rationality and Knowledge*, 195–201.

Bresina, J.; Dearden, R.; Meuleau, N.; Smith, D.; and Washington, R. 2002. Planning under continuous time and resource uncertainty: A challenge for AI. In *Proc. Conference on Uncertainty in AI*, 77–84.

Buffet, O., and Aberdeen, D. 2009. The factored policy-gradient planner. *Artificial Intelligence* 173(5-6):722–747.

Darwiche, A. 2009. *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press.

Dearden, R.; Meuleau, N.; Ramakrishnan, S.; Smith, D.; and Washington, R. 2003. Incremental contingency planning. In *Proc. ICAPS Workshop on Planning under Uncertainty*.

Do, M., and Kambhampati, S. 2003. SAPA: A scalable multi-objective metric temporal planner. *Journal of Artificial Intelligence Research* 20:155–194.

Drummond, M., and Bresina, J. 1990. Anytime synthetic projection: Maximizing the probability of goal satisfaction. In *Proc. National Conference on Artificial Intelligence*, 138–144.

Fox, M., and Long, D. 2003. PDDL 2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research* 20:61–124.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:253–302.

Little, I.; Aberdeen, D.; and Thiebaux, S. 2005. Prottle: A probabilistic temporal planner. In *Proc. National Conference on Artificial Intelligence*.

Mausam, and Weld, D. S. 2004. Solving concurrent markov decision processes. In *Proc. National Conference on Artificial intelligence*, 716–722.

Mausam, and Weld, D. S. 2006. Probabilistic temporal planning with uncertain durations. In *Proc. National Conference on Artificial Intelligence*.

Nebel, B. 2000. On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research* 12:271–315.

Rachelson, E.; Fabiani, P.; Garcia, F.; and Quesnel, G. 2008. A simulation-based approach for solving temporal markov problems. In *Proc. European Conference on Artificial Intelligence*.

Sutton, R. S.; Mcallester, D.; Singh, S.; and Mansour, Y. 1999. Policy gradient methods for reinforcement learning with function approximation. In *Proc. of Advances in Neural Information Processing Systems*, 1057–1063. MIT Press.

Younes, H., and Simmons, R. 2004. Policy generation for continuous-time stochastic domains with concurrency. In *Proc. International Conference on Automated Planning and Scheduling*.