Planning with Concurrency under Resources and Time Uncertainty

Éric Beaudry and **Froduald Kabanza** and **François Michaud**[1]

Abstract. Planning with actions concurrency under resources and time uncertainty has been recognized as a challenging and interesting problem. Most current approaches rely on a discrete model to represent resources and time, which contributes to the combinatorial explosion of the search space when dealing with both actions concurrency and resources and time uncertainty. A recent alternative approach uses continuous random variables to represent the uncertainty on time, thus avoiding the state-space explosion caused by the discretization of timestamps. We generalize this approach to consider uncertainty on both resources and time. Our planner is based on a forward chaining search in a state-space where the state representation is characterized by a set of object and numeric state variables. Object state variables are associated with random variables tracking the time at which the state variables' current value has been assigned. The search algorithm dynamically generates a Bayesian network that models the dependency between time and numeric random variables. The planning algorithm queries the Bayesian network to estimate the probability that the resources (numerical state variables) remain in a valid state, the probability of success and the expected cost of the generated plans. Experiments were performed on a transport domain in which we introduced uncertainty on the duration of actions and on the fuel consumption of trucks.

1 INTRODUCTION

Planning with actions concurrency under resources and time uncertainty has gained in interest over the last decade. One good illustration of such planning problems is the Mars rovers domain. The planning of daily activities for the rovers involves uncertainty on resources and time [3]. For instance, since the surface of Mars is partially known and locally uncertain, the duration of a navigation task and the energy required to fulfill it are probabilistic.

Up to this date, many different approaches have been proposed for planning with actions concurrency under uncertainty. MDP-based approaches include CoMDP and CPTP [10]. Contingency and simulation based paradigms include the Generate, Test, and Debug (GTD) framework [11, 6]. Others include the Factory Policy Gradient planner [4] and Prottle [9] which is based on a LRTDP algorithm.

Many approaches, including the MDP ones, rely on a discrete model for representing resources and time. Adopting a discrete model introduces a blow up of the state-space and the number of state transitions, which limits the size of planning problems that can be addressed in practice. For instance, in the search process, an action having an uncertain duration produces several successor states having different timestamps.

[1] Université de Sherbrooke, Canada, emails: eric.beaudry@usherbrooke.ca, kabanza@usherbrooke.ca, francois.michaud@usherbrooke.ca

Simulation-based approaches like GTD generate contingent plans to address uncertainty. It proceeds as follows: (1) a plan is generated using a deterministic planner which ignores uncertainty; (2) the generated plan is simulated to identify potential failure points; and (3) a conditional branch is added to robustify the plan. These three steps are repeated a number of times to create a contingent plan. However, this approach is not optimal because it uses simulation to identify failure points and employs heuristic methods to decide where conditional branches have to be inserted.

This paper presents another approach for planning with concurrency under numeric (resources and time) uncertainty. It generalizes a previous work which addresses action concurrency and action duration uncertainty [2]. The proposed approach uses a forward chaining search to generate non-conditional plans which are robust to resources and time uncertainty. Random variables are used to model the current belief of resources and time.

A Bayesian network is used to maintain the dependency relationships between the 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 estimate the probability that the resources (numerical state variables) remain in a valid state, the probability of success and the expected cost of the generated plans.

The remainder of this paper is organised as follows. First, the formalism of states and actions is presented. It is followed by the planning algorithm, experiments and a conclusion.

2 FORMALISM

2.1 State-Variable Representation

A state-variable representation is used to describe state features. There are two types of **state variables**: object variables and numeric variables. An **object state variable** $x \in X$ describes a particular feature of the world state which has a finite domain $Dom(x)$. For instance, the current location of a robot can be represented by an object variable whose domain is the set of all locations distributed over a map. A **numeric state variable** $y \in Y$ describes a numeric feature of the world state. A resource like the current energy level of a robot's battery is an example of a state numeric variable. Each numeric variable y has a valid domain of values $Dom(y) = [y_{min}, y_{max}]$ where $(y_{min}, y_{max}) \in \mathbb{R}^2$. It is assumed that no exogenous events take place; hence only planned actions cause state changes. A world state is an assignment of values to the state variables, while action effects (state updates) are changes of variable values. In this paper, we consider uncertainty on the value of numeric variables, but not on those of object variables. The set of all state variables is noted $Z = X \cup Y$.

2.2 Numeric and Time Random Variables

The uncertainty on the numeric state variables is modelled by random variables. A **numeric random variable** $n \in N$ thus models a belief of a numeric state variable. A numeric random variable is defined by an equation which specifies its relationship with other random variables. For instance, let y be a numeric state variable representing a particular resource. The current belief of y is modelled by the numeric random variable n_0. Let the random variable $cons_{a,y}$ represent the amount of resource y consumed by action a. So the execution of action a changes the current belief of y to a new random variable n_1 associated with the equation $n_1 = n_0 - cons_{a,y}$.

The uncertainty on time is also modelled by random variables. A **time random variable** $t \in T$ marks the occurrence of an event, corresponding to either the start or the end of an action. An event induces a change of the values of a subset of state variables. The time random variable $t_0 \in T$ is reserved for the initial time, i.e., the time associated to all state variables in the initial world state. Each action a has a duration represented by a random variable d_a. A time random variable $t \in T$ is defined by an equation specifying the time at which the associated event occurs. For instance, an action a which starts at time t_0 and ends at time t_1 is defined by the equation $t_1 = t_0 + d_a$.

2.3 Formal State Definition

Formally, a **state** s is defined by $s = (U, V, W, P)$ where:

- U is a total mapping function $U : X \rightarrow \cup_{x \in X} Dom(x)$ which retrieves the current assigned value for each object variable $x \in X$ such that $U(x) \in Dom(x)$.
- V is a total mapping function $V : X \rightarrow T$ which denotes the time at which the assignations variables X have become valid.
- W is a total mapping function $W : Y \rightarrow N$ which denotes the current belief of numeric variables Y. A belief of a numeric variable is represented by a numeric random variable $n \in N$.
- P a total mapping function $P : X \rightarrow 2^T$ which indicates the set of time random variables associated to persistence conditions on state object variables X.

Persistence conditions are used to track *over all* conditions of actions. Each time random variable $t \in P(x)$ imposes that object 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. An object state variable x always has an implicit persistence condition that must hold until x becomes valid, i.e., $V(x) \in P(x)$.

Hence a state is not associated with a fixed timestamp like other approaches for action concurrency [1]. A state rather describes the current world state using a set of state features, that is, a set of value assignations for all state variables Z. Contrary to numeric state variables, there is no uncertainty about the values being assigned to object state variables. The only uncertainty on object state variables is about **when** their assigned values become valid. The function $V(x)$ models this uncertainty by mapping each object state variable to a corresponding time random variable.

2.4 Actions

The specification of actions follows the extensions introduced in PDDL 2.1 [7] for expressing temporal planning domains. The set of all actions is denoted by A. Roughly, an **action** $a \in A$ is a tuple a=($name$, $cstart$, $coverall$, $estart$, $eend$, $enum$, 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;
- $estart$ and $eend$ are respectively the sets of effects *at start* and *at end* on the state object variables;
- $enum$ is the set of numeric effects on state numeric variables;
- and $d_a \in D$ is the random variable which models the duration of the action.

A condition c is a boolean expression over state variables. The function $vars(c) \rightarrow 2^X$ returns the set of all object state variables that are referenced by the condition c. An object effect $e = (x, exp)$ specifies the assignation of the value resulting from the evaluation of expression exp to the object state variable x. Expressions $conds(a)$ and $effects(a)$ return respectively all conditions and all effects of action a.

A numeric effect is either a change e_c or an assignation e_a. A numeric change $e_c = (y, numchange_{a,y})$ specifies that the action changes (increases or decreases) the numeric state variable y by the belief on the random variable $numchange_{a,y}$. A numeric assignation $e_a = (y, newvar_{a,y})$ specifies that the numeric state variable y takes as belief the random variable $newvar_{a,y}$.

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 $PDF_{d_a}(u) : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ is the probability density function of the duration u of an action a. We make the assumption that actions have independent durations.

An action a is applicable in a state s if all the following conditions are satisfied:

1. s satisfies all *at start* and *over all* conditions of a (denoted $s \models a$) are satisfied. 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.
2. All state numeric variable $y \in Y$ are in a valid state, i.e., $W(y) \in [y_{min}, y_{max}]$.

Since the belief of a numeric state variable is probabilistic, the validity of its value is also probabilistic. Hence the application of an action may cause a numeric state variable to become invalid. We note $Pb(W(y) \in Dom(y))$ the probability that a numeric state variable y is in a valid state when its belief is modelled by a numeric random variable $W(y) \in N$.

2.5 Example: Transport Domain

Consider the Transport planning domain in which trucks have to deliver packages to different locations distributed over a map. Let $R = \{r_1, ..., r_n\}$ be a set of n trucks, $P = \{p_1, ..., p_m\}$ be a set of m packages and $L = \{l_1, ..., l_k\}$ be a map of k locations. 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 $Goto(r, l_a, l_b)$ describes the movement of a truck r from location l_a to location l_b. The required fuel and the duration of a *Goto* action are modelled using normal distributions where both the mean and the standard deviation are proportional to the distance to be traveled. *Load* and *Unload* actions specify the loading and unloading of a package by a truck. The duration of these actions is defined by a uniform distribution. The set of object 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 object variables is defined as $Dom(C[r]) = L$ ($\forall r \in R$) and $Dom(C[p]) = L \cup R$ ($\forall p \in P$). The set of numeric state variables $Y = \{F[r] | r \in R\}$ specifies the current fuel level of trucks. 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 $dt_k \in \mathbb{R}^+$.

$Goto(r, l_a, l_b)$	
$cstart$	$CurrentLocation[r] = l_a$
$eend$	$CurrentLocation[r] = l_b$
$duration$	$Normal(dist/speed, 0.2 * dist/speed)$
$enum$	$Fuel[r]$-=$Normal(dist/rate, 0.3 * dist/rate)$
$Load(r_i, p_j, l_k)$	
$cstart$	$CurrentLocation[p_j] = l_k$
$coverall$	$CurrentLocation[r_i] = l_k$
$eend$	$CurrentLocation[p_j] = r_i$
$duration$	$Uniform(30, 60)$
$Unload(r_i, p_j, l_k)$	
$cstart$	$CurrentLocation[p_j] = r_i$
$coverall$	$CurrentLocation[r_i] = l_k$
$eend$	$CurrentLocation[p_j] = l_k$
$duration$	$Uniform(30, 60)$
$Refuel(r, l)$	
$coverall$	$CurrentLocation[r] = l_k$
$enum$	$Fuel[r] = Fuel_{max}[r]$
$duration$	$Uniform(30, 60)$

Table 1. Actions specification of the Transport domain

3 RTU Planner

The Resources and Time Uncertainty (RTU) Planner performs a forward chaining search in a space of states. The state-space explored at any given point is a graph, with nodes corresponding to states and transitions representing actions.

In addition to the search graph, the planner generates a Bayesian network to track the dependency relationships between the random variables. A Bayesian network is a directed acyclic graph $B = (M, E)$ where M is a set of random variables and E is a set of edges representing dependencies between random variables. Dependencies in the Bayesian network are derived from the equations of the numeric and time random variables, which in turn are obtained from the application of an action to the current state.

The planning algorithm handles concurrency and delayed effects differently from a traditional model for concurrency [1]. A delayed effect for an action takes place 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. There are transitions along which time freezes to interleave simultaneous actions and transitions updating the timestamp. 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 directly attached to states; it is rather attached to state features. Therefore there is no need for delayed effects and for the special advance time action. Time increments are rather tracked by the time variables attached to state 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 thus 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 a given metric formula. The choice of an action a at Line 5 is a backtrack point. A relaxed planning graph-based heuristic [8] is involved to guide this choice.

Algorithm 1 Plan

```
1. PLAN(s, G, A)
2.   if Pb(s ⊨ G) ≥ threshold
3.     π ← ExtractPlan(s)
4.     return π
5.   nondeterministically choose a ∈ {a ∈ A | a ⊨ s}
6.   s' ← Apply(s, a)
7.   return Plan(s', G, A)
```

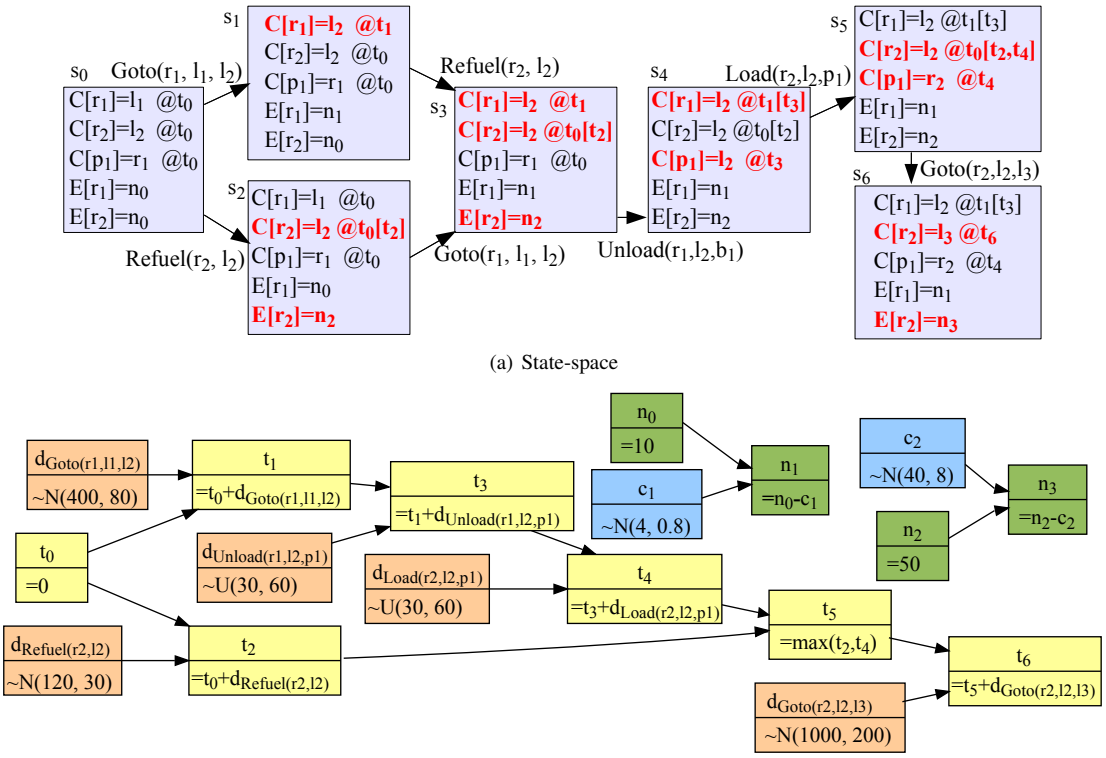
Algorithm 2 Apply Action Function

```
1. function APPLY(s, a)
2.   s' ← s
3.   t_conds ← max_{x∈vars(conds(a))} s.V(x)
4.   t_release ← max_{x∈vars(effects(a))} max(s.P(x))
5.   t_start ← max(t_conds, t_release)
6.   t_end ← t_start + d_a
7.   for each c ∈ a.coverall
8.     for each v ∈ vars(c)
9.       s'.P(x) ← s'.P(x) ∪ {t_end}
10.  for each e ∈ a.estart
11.    s'.U(e.x) ← eval(e.exp)
12.    s'.V(e.x) ← t_start
13.    s'.P(e.x) ← {t_start}
14.  for each e ∈ a.eend
15.    s'.U(e.x) ← eval(e.exp)
16.    s'.V(e.x) ← t_end
17.    s'.P(e.x) ← {t_end}
18.  for each e ∈ a.enum
19.    s'.W(e.x) ← eval(e.exp)
20.  returns s'
```

The Apply function of Algorithm 2 details how a resulting state s' is obtained from 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 object 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 object state variables modified by an effect. Then at Line 5, the time random variable t_{start} is created. Indeed, its equation corresponds to the max value 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 object state variable involved in an *over all* condition. Lines 10-17 process *at start* and *at end* effects. For each effect on an object 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. Line 18-19 process numeric effects.

3.1 Example

Figure 1 illustrates an example of a partial search carried out by Algorithm 1 on a problem instance of the Transport domain. Expanded



(a) State-space

(b) Bayesian network

Figure 1. Sample search with the Transport domain

states are shown in (a). Three object state variables are shown on each state: $C[r_1]$, $C[r_2]$ and $C[p_1]$ represent respectively the current location of trucks r_1 and r_2 and of package p_1. Each state has two numeric state variables, $E[r_1]$ and $E[r_2]$, representing the fuel level of each truck. All object state variables of the initial state s_0 are associated with the time random variable t_0 which represents the initial time. Trucks r_1 and r_2 are initially located at locations l_1 and l_2 respectively, and each one has 10 units of fuel, which is represented by the n_0 numeric variable. The package p_1 is initially loaded on truck r_1. Subfigure (b) shows the generated Bayesian network. Several elements are depicted, including the equations of time and numeric random variables, and the probability distributions followed by the action duration random variables and by the numeric change random variables.

State s_1 is obtained by applying the action $Goto(r_1, l_1, l_2)$ to state s_0. The Apply function (see Algorithm 2) works as follows. The action $Goto(r_1, l_1, l_2)$ 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 $C[r_1]$ object state variable, Line 4 computes the time $t_{release} = \max(t_0) = t_0$. At Line 5, the action's 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 $t_{end} = t_0 + d_{Goto(r_1,l_1,l_2)}$ is created and added to the Bayesian network with the label t_1. Next, Lines 13-16 apply effects by performing the assignation $C[r_1] = l_2$ and by setting time t_1 as the valid time for $C[r_1]$. The numeric effect creates a new numeric random variable n_1 which is associated to the belief of $E[r_1]$ in state s_1.

Applying action $Refuel(r_2, l_2)$ from state s_0 generates the state s_2. Since this action has an *over all* condition on the state variable $C[r_2]$, a persistence condition is added until t_2 (noted within [] in the figure). The belief of $E[r_1]$ is updated to the numeric random variable n_2, which increases the fuel capacity to 50. Because the actions $Refuel(r_2, l_2)$ and $Goto(r_1, l_1, l_2)$ can be executed simultaneously, the planner finds two paths to state s_3.

Applying action $Unload(r_1, l_2, p_1)$ in state s_3 creates state s_4. 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 $t_3 = t_1 + d_{Unload(r_1,l_2,p_1)}$. 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. The action $Load(r_2, l_2, p_1)$ has two *at start* conditions: $C[r_2] = l_2$ and $C[p_1] = l_2$. 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 $t_4 = \max(t_2, t_3)$.

3.2 Bayesian Network Inference Algorithm

A Bayesian network inference algorithm is required to estimate the probability of success and the expected cost of plans. The choice of an inference algorithm for Bayesian networks is guided by the structure of the Bayesian network and by the type of random variables it includes [5]. In our case, the Bayesian network is composed of continuous random variables. In this case, analytical inference methods are possible if one can impose some restrictions on the allowed probability distributions. In particular, normal distributions are often preferred because they are defined by two parameters (mean μ and

standard deviation σ), which makes the manipulation of such distributions possible with analytical approaches.

In our approach, the numeric random variables (N) and 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 follow a normal distribution in our approach because of the manipulations involved in the generation of the variables' equations.

Our approach rather 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 [5]. Let B be a Bayesian network defined as $B = (M, E)$ where M is a finite set of random variables and E is a finite set of dependencies. The direct sampling algorithm starts by finding a topological sort order $M' = (v_1, v_2, \ldots, v_n)$ of M where $n = \|M\|$ and $v_j \notin ancestors(v_i), \forall i, j \mid i < j \leq n$. Then it generates a sample for each random variable using M'. For random variables having no parent, the algorithm generates a random sample using their probability distribution. For the other random variables, it generates a sample according to the parents' generated samples. The estimation of the belief of random variables is done by repeating this procedure m times and by computing the average of all samples for each random variable. The number of samples m is set empirically. The estimation error is inversely proportional to the square root of m.

3.2.1 Incremental Belief Evaluation

The Bayesian network is constructed dynamically during the search process. Algorithm 2 is responsible for repeatedly extending the network. Once a new numeric random variable or 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 random variables is required by the heuristic function to guide the planning search, and to estimate the probability that a plan satisfies time constraints.

Because the Bayesian network is generated dynamically, 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 in memory, more precisely in arrays corresponding to each random variable. The i^{th} sample 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, it is impossible to do so when dealing with 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.

4 EMPIRICAL RESULTS

We experimented our algorithm on a planning domain inspired by the International Planning Competition (IPC). Since the previous editions of the uncertainty track did not consider both concurrency and uncertainty on resources and time, we created our own set of problems. The Transport domain we use introduces uncertainty on the consumption of resources and on the duration of actions. The definition of actions is presented in Table 1.

A direct comparison with other approaches like CPTP [10] and GTD [11] was not possible because these planners were not available at writing time. To give an idea of the cost of the required overhead for addressing uncertainty, we compared two versions of our planner: the first one considers uncertainty (RTU Planner) and the second one ignores it (Deterministic Planner). The RTU Planner generates plans which have a probability of success grater than or equal to 0.9. The Deterministic Planner ignores uncertainty by using the mean of the probabilistic distribution of the durations and the resource usage.

Table 2 reports the empirical results obtained on randomly generated problems. The first and second columns show the size of the problems, expressed in terms of the number of trucks and packages. Columns under RTU 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), the expected plan cost and the absolute error (ϵ) on the expected plan cost. The cost of plans is computed as a linear combination of the plan's makespan and the total amount of fuel consumed. The expected cost of plans is estimated by the Bayesian network and its error (ϵ) is given with a confidence level of 0.95. To estimate the belief of a random variable within the Bayesian network, 5000 samples has been generated. We keep arrays of samples in memory for at most 10000 random variables. A few problems were not successfully solved (marked by –) because the allocated time limit (10 minutes) was reached. The experiments were made on an Intel Core 2 Quad 2.4 GHz computer with 2 GB of RAM. Columns under Deterministic Planner give the number of states, the cost of the generated plan, and the CPU time (in seconds). These experiments validate our hypothesis that the overhead of managing random variables is largely compensated by the state-space reduction incurred. Indeed, we avoid the state-space blow up caused by having different timestamps for each duration unit.

4.1 Impact of the number of samples

The necessary use of an inference algorithm to evaluate random variables in the Bayesian network imposes a computational overhead. Direct sampling algorithms have a O(nm) runtime where n is the number of random variables and m is the number of samples. A higher number of generated samples produces a lower estimation error on the belief of random variables. Figure 2 presents the planning time and the estimation error of the plans' cost with respect to the number of samples, for a problem with 2 trucks and 3 packages. The planning time grows linearly with the number of samples while the estimation error is inversely proportional to the square root of the number of samples. For this problem, 5000 samples represents a good trade-off between planning speed and the estimation error.

5 CONCLUSION

We presented a new planning approach that extends the forward chaining search for dealing with action concurrency under resources and time uncertainty. Rather than representing resources and time with discrete numeric values, continuous random variables are used. Each object state variable is associated to a continuous time random variable representing the time at which the state feature has been

Problem Instance		Resources and Time Uncertainty (RTU) Planner					Deterministic Planner		
Trucks	Packages	States	RV	CPU (s)	Cost	ϵ	States	CPU (s)	Cost
1	2	94	179	0.088	1631.6	3.6	61	0.031	1631.7
1	3	1560	2065	1.08	1751.5	3.5	680	0.401	1746.7
1	4	6588	7330	4.17	2147.4	3.9	899	0.129	2137.7
2	2	50	98	0.036	1032.4	3.6	50	0.001	1029.6
2	3	24147	4343	6.87	1079.7	3.7	12660	1.285	1074.6
2	4	15829	6864	4.62	1470.2	3.3	9484	0.442	1465.6
2	5	8479	7111	3.55	2068.0	4.8	5728	0.173	2061.9
3	3	142	224	0.091	962.8	3.4	142	0.004	964.4
3	4	584349	31486	600	–	–	429009	80.7	1334.0
3	5	124435	13305	27.56	1478.7	3.5	77128	5.105	1472.0
3	6	200087	23570	54.05	1755.4	4.7	94942	6.162	1749.7
3	7	71332	15301	18.56	1755.1	4.7	40250	2.266	1750.0
4	3	143	208	0.086	721.8	2.2	143	0.003	716.9
4	4	273	337	0.141	984.2	3.4	273	0.006	982.2
4	5	524831	28025	600	–	–	739248	600	–
4	6	243664	23018	63.6	1667.0	4.8	128836	9.511	1666.6
4	7	450474	29740	600	–	–	683742	600	–

Table 2. Empirical results for Transport domain

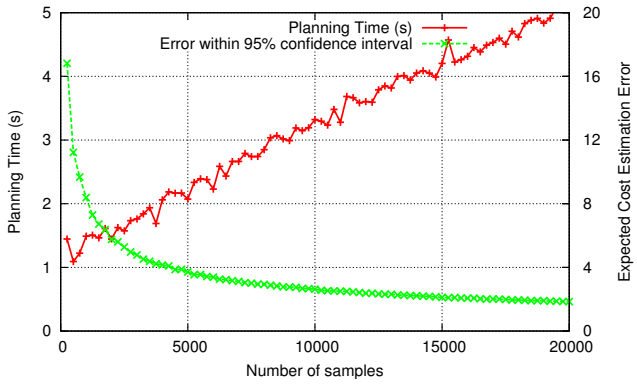


Figure 2. Impact of number of samples

assigned a value. Additionally, each numeric state variable (e.g., a resource) is associated to a numeric random variable representing its current belief. Random variables are organized into a Bayesian network. A direct sampling algorithm is used to estimate the probability of success and the expected quality of plans. Empirical experiments on probabilistic versions of the Transport and Rovers domains show that our planner is able to deal efficiently with actions concurrency under resources and time uncertainty.

As future work, we plan to introduce contingency in our approach. Conditional branches will be added to improve the quality of plans. Each branch will be associated to a test condition comparing the observed time to a determined time. The challenge is to find where to insert conditional branches and which predetermined time to use. We will analyze the probability distribution of the numeric and time random variables associated with states reaching the goal, but only those with a probability of success under a specific threshold. This could give valuable insight on how much a contingency branch could improve the plan's quality.

ACKNOWLEDGEMENTS

This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Fonds québécois de la recherche sur la nature et les technologies (FQRNT). We would like to thank the referees for their comments, which helped improve this paper.

REFERENCES

[1] F. Bacchus and M. Ady, 'Planning with resources and concurrency: a forward chaining approach', in *Proc. of International Joint Conference on Artificial Intelligence*, pp. 417–424, (2001).

[2] E. Beaudry, F. Kabanza, and F. Michaud, 'Planning for concurrent action executions under action duration uncertainty using dynamically generated bayesian networks', in *Proc. of International Conference on Automated Planning and Scheduling*, (2010).

[3] J. Bresina, R. Dearden, N. Meuleau, D. Smith, and R. Washington, 'Planning under continuous time and resource uncertainty: A challenge for AI', in *Proc. of 19th Conference on Uncertainty in AI*, pp. 77–84, (2002).

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

[5] A. Darwiche, *Modeling and Reasoning with Bayesian Networks*, Cambridge University Press, April 2009.

[6] R. Dearden, N. Meuleau, S. Ramakrishnan, D. Smith, and R. Washington, 'Incremental contingency planning', in *Proc. of ICAPS Workshop on Planning under Uncertainty*, (2003).

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

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

[9] I. Little, D. Aberdeen, and S. Thiébaux, 'Prottle: A probabilistic temporal planner', in *Proc. of National Conference on Artificial Intelligence*, (2005).

[10] Mausam and Daniel S. Weld, 'Concurrent probabilistic temporal planning', *Journal of Artificial Intelligence Research*, **31**, 33–82, (2008).

[11] H. Younes and R. Simmons, 'Policy generation for continuous-time stochastic domains with concurrency', in *Proc. of International Conference on Automated Planning and Scheduling*, (2004).