Generating Plans in Concurrent, Probabilistic, Over-Subscribed Domains

Li Li and **Nilufer Onder**
Department of Computer Science
Michigan Technological University
1400 Townsend Drive
Houghton, MI 49931
{lili,nilufer}@mtu.edu

Abstract

Planning in realistic domains involves reasoning under uncertainty, operating under time and resource constraints, and finding the optimal set of goals to be achieved. In this paper, we provide an AO* based algorithm that can deal with durative actions, concurrent execution, over-subscribed goals, and probabilistic outcomes in a unified way. We explore plan optimization by introducing two novel aspects to the model. First, we introduce parallel steps that serve the same goal and increase the probability of success in addition to parallel steps that serve different goals and decrease execution time. Second, we introduce plan steps to terminate concurrent steps that are no longer useful so that resources can be conserved. Our algorithm called CPOAO* (Concurrent, Probabilistic, Oversubscription AO*) can deal with the aforementioned extensions and relies on the AO* framework to reduce the size of the search space using informative heuristic functions. We describe our framework, implementation, the heuristic functions we use, the experimental results, and potential research on heuristics that can further reduce the size of search space.

Introduction

Generating plans in realistic domains presents two main challenges. First, there is uncertainty about the action effects as well as the state of the world. Second, the resources for carrying out the tasks are limited. Research in *probabilistic planning* deals with issues involving uncertainty (Kushmerick, Hanks, and Weld 1995; Bonet and Geffner 2005). *MDP-based planners* (Boutilier, Dean, and Hanks 1999) deal with both uncertain actions and resource limitations. Heuristics have been developed for dealing with time and resources in deterministic domains (Haslum and Geffner 2001). *Oversubscription planners* must select a subset of the goals to plan for because resource limitations do not allow all the goals to be achieved (Smith 2004; Benton, Do, and Kambhampati 2005). *Concurrent planners* generate plans with shorter makespan by using parallel actions (Little and Thiebaux 2006; Mausam and Weld 2008). Our work contributes to recent research by providing a unified framework to consider durative, concurrent, probabilistic actions in over-subscribed domains; by exploring the use of parallel actions to increase expected rewards; and by studying interruptible actions to optimize resource usage.

To motivate our work, consider a simplified Mars rover domain (Bresina et al. 2002) where two pictures must be taken within 5 minutes, and actions have fixed known durations. The rover has two cameras: cam_0 succeeds with probability 0.6, and takes 5 minutes; and cam_1 succeeds with probability 0.5, and takes 4 minutes. In a case where both pictures have a value of 10, the best strategy is to use cam_0 for one picture and cam_1 for the other in parallel. Because all the actions need to finish to collect the rewards, we call this *all-finish parallelism*. The total expected reward will be $10 \times 60\% + 10 \times 50\% = 11$. In a different case where the picture values are 100 and 10, the best strategy is to use both cam_0 and cam_1 in parallel to achieve the larger reward. In this case, the success of the earliest finishing action is sufficient. We call this case *early-finish parallelism*. Cam_1 finishes earlier than cam_0 and the expected reward for using cam_1 is $100 \times 50\% = 50$. If cam_1 fails, the expected remaining unachieved reward will be $100 \times (1 - 50\%) = 50$. Then the expected reward for action cam_0 is $50 \times 60\% = 30$. Therefore, the total expected reward is $50 + 30 = 80$. This is larger than the expected reward of using the cameras for different pictures, which is, $100 \times 60\% + 10 \times 50\% = 65$.

When both cameras are used for the same picture, if cam_1 succeeds in achieving the target reward, we can abort cam_0 immediately unless it serves other goals. Such termination avoids unnecessary expenditure of resources. If plan steps are marked with termination conditions during plan generation, they can be monitored during plan execution for these conditions.

We have developed an algorithm called CPOAO* (Concurrent, Probabilistic, Oversubscription AO*) which extends the AO* framework to use concurrent actions of both kinds defined above. Other AO* based planners include LAO* (Hansen and Zilberstein 2001) which finds solutions with loops, and HAO* (Mausam et al. 2005) which deals with continuous resources and stochastic consumptions. Recent concurrent planners are CPTP (Mausam and Weld 2008), an MDP-based planner, and Paragraph (Little and Thiebaux 2006) which uses a Graphplan framework. These algorithms prevent actions that achieve the same goal

from executing in parallel ("restricted" concurrency as explained in (Little and Thiebaux 2006)). In domains where explicit rewards are assigned to goals, there is advantage in having parallel actions that serve the same goal, i.e., early-finish parallelism. In our approach, we provide the means for using such parallel actions to maximize expected rewards. In multi-agent domains or in parallel single-agent domains, there is advantage in terminating actions as soon as the expected result is obtained, or as soon as it becomes impossible to obtain the expected results. Our algorithm is capable of marking the actions to be terminated so that resources can be saved to achieve other goals. Another example of using redundant actions comes from ProPL, a process monitoring language (Pfeffer 2005) where processes might include redundant parallel actions such as seeking task approval from two managers when one approval is sufficient. The focus of ProPL is on expressing and monitoring such actions. Our focus is in generating plans that can use redundant actions, as well as marking the actions that need to be terminated.

The Planning Problem

We consider probabilistic, over-subscribed planning problems with durative actions. The solution plans have concurrent actions. Formally, a *planning problem* is a 5-tuple (S, A, s_0, R, T) where S is the state space, A is the set of actions, s_0 is the initial state, R is the reward set, and T is the time limit for plan execution.

A state $s \in S$ is a quadruple (s_p, s_r, s_a, t) where $s_p \subseteq P$ is a set of propositions, P is the set of all the domain propositions, s_r is a vector of numeric resource values, s_a is the set of currently executing concurrent actions, and t is the remaining available time. For each action in s_a, the remaining time to complete is recorded.

An action a consists of a precondition list, a resource consumption vector, and a set of possible outcomes. An outcome o_i is defined as a triple (add(o_i), del(o_i), prob(o_i)) denoting the add list, the delete list, and the probability that this outcome happens. The total probability of all the possible outcomes of an action should be 1. We adopt the common semantics for action execution. Before an action can be executed, the preconditions must hold, and the amount of each resource must be greater than or equal to the value specified for that resource. After an action is executed, the result is a set of states where for each outcome, the resources used are subtracted, the propositions in the add list are added, and the propositions in the delete list are deleted. We assume that resources are consumed in proportion to execution time regardless of whether the action succeeds or fails. For example, if an action which has a duration of 5 and consumes 4 power units is aborted after 3 time units, it consumes $4 \times (3/5) = 2.4$ power units. The probability of the new state resulting from outcome o_i in state s_j is the probability of s_j multiplied by prob(o_i). The process of finding the results of an executed step and a special *do-nothing* action are defined formally further below.

Each reward in R is a proposition-value pair. When a proposition is achieved, its value is added to the total rewards. When it becomes false, its value is subtracted from the total rewards. This way, the value of a proposition can contribute at most once to the total rewards collected. Our problem model does not include *hard goals* because a plan with a probability of 1.0 might not exist. Instead, important goals are represented by assigning large rewards to them. In this regard, our probabilistic actions are similar in spirit to PPDDL actions with the exception of rewards being associated directly with propositions rather than through state transitions (Younes et al. 2005, pp. 854-855). The current implementation supports a TGP-style (Smith and Weld 1999) subset of PPDDL. We assume that the preconditions hold at start and overall, the effects are only available after the action finishes successfully, and the action duration is fixed.

The solution to a planning problem is an *optimal contingent plan* that maximizes the expected reward of the initial state $s_0 = (s_{p0}, s_{r0}, s_{a0}, t_0)$. In s_0, the propositions in s_{p0} hold, the propositions in $P - s_{p0}$ do not hold, the resources have the levels given in the vector s_{r0}, no actions have been started ($s_{a0} = \emptyset$), and t_0 is equal to time limit T. We formally define a *plan* as a set of (state, <CAS,TS>) pairs where <CAS,TS> denotes the actions that will be taken at the given state. A CAS is a *concurrent action set* that includes both already executing actions and newly started actions. The actions in a CAS should be *compatible*. Two actions a_1 and a_2 in a CAS are said to be *incompatible* if (1) a proposition occurs in one of the add lists of a_1 and in one of the delete lists of a_2, (2) a_1 produces the opposite of a precondition of a_2, or (3) the sum of their resource requirements is greater than the current available resource levels. A TS is a *termination set* that denotes the set of executing steps that will be terminated.

For each action in a CAS, we specify the number of time units left to execute. The *duration* of CAS_i (dur(CAS_i)) is defined as the duration of the shortest action in it. This allows us to evaluate the states resulting from the termination of the earliest action. The *result* of executing a $<\text{CAS}_i,\text{TS}_i>$ in a state $s_i = (s_{pi}, s_{ri}, s_{ai}, t_i)$ is computed by first computing $s'_i = (s'_{pi}, s'_{ri}, s'_{ai}, t'_i)$ so that the effects of terminating the actions in TS_i are reflected. The *result* of executing the concurrent action set CAS_i in state $s'_i = (s'_{pi}, s'_{ri}, s'_{ai}, t'_i)$ is a probability distribution over the resulting states and is defined with respect to the action that requires the minimum time to execute. Each state in the probability distribution corresponds to a distinct action outcome and is obtained by (1) using the add and delete lists to find the new set of propositions that hold, (2) using the resource consumption vectors of the executing actions to update the resource levels, (3) updating the actions in the CAS set to reflect the execution time that has passed, and (4) updating the time t'_i to reflect the remaining available time. Suppose that dur(CAS_i) $= t_j$ and action a_j has this duration. This means that a_j will be the first action to complete in CAS_i. Further suppose that a_j has n possible outcomes $o_1, \ldots, o_n$. Then, *result* (s'_i, CAS_i) is defined as a probability distribution over n states where each state is obtained by applying an outcome to state s'_i. The result of applying outcome o_k to s'_i is a new state s_k defined as: $s_k = (s_{pk} = s'_{pi} - \text{del}(o_k) + \text{add}(o_k), s_{rk}, s_{ak}, t_k = t'_i - t_j)$. s_{ak} is obtained by subtracting the action duration t_j from

all the action durations in CAS_i, and removing the completed action a_j from CAS_i. s_{rk} is obtained by applying the resources consumed by executing and terminated actions. The probability of the new state s_k is defined as prob(s'_i) $\times$ prob(o_k). If there are multiple actions that require t_j time units, then the effects of all of these actions are applied.

The underlying theoretical model of our framework can be thought of as a concurrent MDP (Mausam and Weld 2008). In order to provide the ability to utilize redundant parallel actions and to abort useless actions, the concurrent MDP model must be extended in two ways. First, the concurrent action sets should include redundant actions that can increase the probability of achieving some goals. Second, the set of actions defined for the MDP should include termination actions in addition to startable actions.

Our objective is to find an optimal contingent plan that maximizes the expected reward of the initial state s_0. We use the following definition for the expected reward of a state s_i with respect to a plan π:

$$E_\pi(s_i) = \begin{cases} \sum_{s_j \in C(s_i)} P_{(i,j)} E_\pi(s_j) & \text{if } s_i \text{ is not a terminal state} \\ R_{s_i} & \text{if } s_i \text{ is a terminal state} \end{cases}$$

In the above equation, $C(s_i)$ is the set of resulting states under plan π, $P_{(i,j)}$ is the probability of entering state s_j from state s_i, and R_{s_i} is the sum of the rewards achieved when ending in state s_i. The *terminal states* are the leaves of the search graph and are explained in the next section. A discount factor is not needed because a reward associated with a proposition can be received only once, and the planning problem includes a time limit that bounds the horizon.

The CPOAO* Algorithm

AO* is a heuristic search algorithm that searches in an "and-or" graph (Nilsson 1980). The CPOAO* algorithm depicted in Figure 1 extends the AO* algorithm. The nodes represent the states, the hyperarcs represent probabilistic actions, and the "and sets" represent the possible results of actions. The input to the CPOAO* algorithm is a planning problem, the output is a function that maps the reachable states to concurrent action sets. The output function represents an acyclic finite automaton. The solution plan does not contain loops due to two reasons similar to the HAO* framework (Mausam et al. 2005). First, the time limit that is given as part of the planning problem bounds the horizon. Second, every action has a non-zero duration and time is part of the state.

The main data structure is a search graph called the *working graph (WORK-G)* (Hansen and Zilberstein 2001; Mausam et al. 2005). WORK-G is a *hypergraph* with nodes that represent states and *hyperarcs* that represent the alternative concurrent action sets (CASs) and termination sets (TSs) that can be executed in a state. Each ending node of a hyperarc represents one possible outcome of the <CAS,TS> pair. We represent a plan by a *solution graph* (SOLN-G). CHANGE-E is a set that is used to propagate the changes in the expected rewards upward in the solution graph.

The main loop of the search starts at line 3, and continues until the solution graph is "complete," i.e., has no non-terminal tip states. The leaves of the search graph are the *terminal states.* There are three types of terminal states. In the first type, there are no applicable actions due to lack of preconditions or resources. In the second type, all the rewards have been collected, therefore there is no need to execute any action. The third type includes the states entered after executing the special "do-nothing" action from a state s_j. In this case, s_j is a state where there are unachieved rewards and resources available, but leaving the current state might result in the loss of already achieved rewards. For example, s_j might be a state where the rover is at the base location, but the amount of resources will not be sufficient to take the rover back if it leaves. Thus, the best action to execute at s_j is the "do-nothing" action.

In line 5, all the non-terminal tip states in the solution graph are expanded. To expand a search state s, we find the set of all the applicable action pairs using the procedure in Figure 2. We first find all the single actions that are applicable and store it in the set I. An action is applicable in a state if its preconditions hold, there are sufficient resources,

Require: A planning problem (5-tuple).
Ensure: A solution plan that can contain concurrent actions.
1: WORK-G $\leftarrow$ MAKEROOTNODE(s_0).
2: SOLN-G $\leftarrow$ INSERT(SOLN-G, s_0)
3: **while** SOLN-G contains non-terminal tip states **do**
4: CHANGE-E $\leftarrow \emptyset$
5: **for each** s that is an unexpanded non-terminal tip state in SOLN-G **do**
6: APPLICABLE $\leftarrow$ COMPUTE-CAS-TS(s)
7: **for all** ac in APPLICABLE **do**
8: Apply ac on state s to generate the child states of s.
9: Calculate the heuristic values of the expected rewards for newly generated children states.
10: **end for**
11: Find BEST-CAS-TS, the best action set for s.
12: Expand SOLN-G to include BEST-CAS-TS.
13: Update the expected reward of state s based on BEST-CAS-TS.
14: Add the parent states of s into the set CHANGE-E if the expected reward of s has changed.
15: **end for**
16: **while** CHANGE-E is not empty **do**
17: Choose and remove a state $s' \in$ CHANGE-E that has no descendant in CHANGE-E.
18: Update the expected reward of s' by reselecting its best CAS..
19: **if** The expected reward of s' has changed **then**
20: Add the parent states of s' into CHANGE-E.
21: **end if**
22: **end while**
23: Recompute the best solution graph B by following the best CAS-TSs from the initial state s_0 to the tip states.
24: **end while**
25: **return** SOLN-G

Figure 1: The CPOAO* algorithm.

and it is not already executing. Next, we generate I^p, the power set of I. Each element of I^p represents a set of actions that are candidates for concurrent execution. Then, the cross product of I^p with all the possible sets of actions that can be terminated is taken. The sets containing incompatible actions are eliminated from the resulting set.

Require: A state.
Ensure: A set of applicable concurrent action sets.
1: Initialize I with all the applicable actions for state s.
2: Generate I^p which is the power set of set I.
3: Merge all the unfinished actions of state s to the sets in I^p.
4: For each CAS in I^p, add the possible termination actions.
5: If a <CAS,TS> contains incompatible actions, delete it from set I^p. The resulting set,denoted I^c, contains all applicable <CAS,TS> pairs for state s. For each applicable CAS, set its duration to the duration of the shortest action in it.
6: **return** I^c.

Figure 2: The COMPUTE-CAS-TS algorithm.

Terminating an action might have benefits that become obvious only further down in the search tree. For example, when an action finishes with success, all the parallel actions that serve only the succeeding goal should be terminated because they become unnecessary. When an action finishes with failure, all the parallel actions that serve only the failing goal should be terminated because they can't succeed individually. In the base algorithm, steps are terminated based on the expected rewards calculated from the children states. Heuristics based on causal link analysis can be incorporated for efficiency. Due to the power set computation, the set APPLICABLE contains an exponential number of subsets. Therefore, heuristics that can decrease the number of <CAS,TS> pairs are valuable. We describe such heuristics in the next section.

After finding the set of action pairs applicable, we apply each <CAS,TS> pair to state s using the *result* procedure defined in the previous section. The working graph is updated to include the new resulting states or to add the links to the already existing states. The heuristic value of each new state is computed. At the next step (line 11), the best <CAS,TS> pair for state s is found using the following formula: $\text{argmax}_{\text{cas-ts}\in\text{APPLICABLE}}\text{result}(s, \text{cas-ts})$. The best <CAS,TS> pair and its resulting states are marked to become part of the best solution graph (line 12). In lines 13 through 21, the expected value of the state s as well as its ancestors in the solution graph are updated using the best <CAS,TS> pair.

When every state in the current best solution graph is either expanded or a terminal state, the optimal plan is found. The algorithm returns the best solution graph (SOLN-G) that encodes this plan. Figure 3 shows the contingent plan found by CPOAO* for the example described in the first section. Taking pictures 1 and 2 have rewards of 15 and 20, respectively, and the time limit is 7. Each node represents

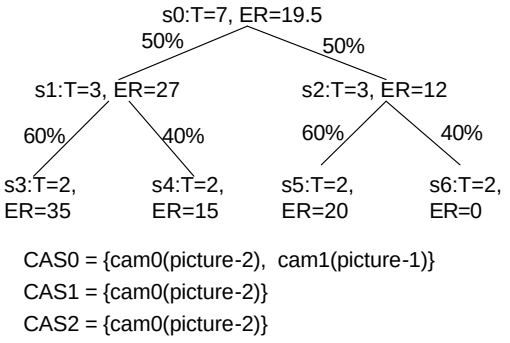

Figure 3: Example plan found by CPOAO*.

a state that can be reached under this plan. For each node, we show the time left (T) and the final expected reward (ER) at this state. For the CASs that have probabilistic outcomes, the probability of each outcome is marked on the link. The left branch corresponds to the successful outcome and the right branch is the failed one. CAS0, CAS1 and CAS2 are applied at s0, s1 and s2 respectively. Both of CAS1 and CAS2 only contains one unfinished action cam0(Picture-2). The remaining duration of cam0(picture-2) is 1. The final total expected reward of s0 is 19.5.

CPOAO* inherits the optimality property from AO* when an admissible heuristic is used. CPOAO* is guaranteed to terminate because a finite time limit for plan execution is given in the planning problem and each action takes a non-zero duration.

Heuristics and Empirical Evaluation

To evaluate our algorithm we designed problems based on the Mars rover domain introduced earlier. We varied problem complexity by using 5, 10, 12, and 15 locations. We created different problems by varying the connectivity of the locations and the number of rewards. The problem names are coded as "m-n-k" where m is the number of locations, n is the number of paths and k is the number of the rewards in the problem.

The base algorithm runs for only small problems due to parallelism and the large branching factor. We had similar observations when Paragraph (Little and Thiebaux 2006) was run with the Mars rover problems that were modified to remove the action durations and goal rewards. When started without a bounding horizon, Paragraph[1] was able to find cyclic solutions for problems 5-5-5 and 5-10-5 in 1.27 and 3.66 seconds, respectively. In Table 1, we show execution times of acyclic Paragraph with time bounds of 10 and 20, and base CPOAO* with a time bound of 20. CPOAO* was

[1]Paragraph was compiled using CMUCL version 19c. The experiments were run on a 2 GB computer with a 3.2GHz Pentium CPU. CPOAO* was compiled using g++ 4.1.2 on a Fedora Release 7 system. The experiments were run on a 1 GB computer with 4 Pentium 3GHz CPUs.

able to solve only the first problem with a time bound of 40. The results reveal that increasing the time limits make the problems too complex for both algorithms. Note that we provide execution times as a reference only and not as an indicator of the performance of the two algorithms. Because C++ is usually faster than Lisp, comparing the execution times would be misleading.

We observed that in parallel domains heuristics would be most effective in steps 6 and 9 of the CPOAO* algorithm shown in Figure 1. Step 6 computes all the applicable actions at a state and step 9 computes the expected rewards to direct the search algorithm. Therefore, we first looked at ways to prune branches of concurrent action sets (CASs). For ease of implementation, we restricted the domain to include only time as a resource, and implemented a technique to prune branches of concurrent actions sets. The technique is based on the fact that we do not need to have a branch for a concurrent action set if there is another concurrent action set that is always *better* than it. We say that the concurrent action set A is always better than the concurrent action set B if the expected total reward that could be collected by following the concurrent action set A is always greater than following the concurrent action set B. For example, when the actions do not consume any resources other than time and action preconditions are met, starting the actions earlier is always better than letting the operators be idle. We can abort any action at any time if it turns out that we should not wait for its completion. The following rules can be used to determine whether or not two concurrent action sets have the "better" relation. (1) If the shortest action a in CAS A does not consume any resource other than time or delete any propositions, then CAS A is always better than CAS B = A-{a}. (2) Let $A = B \cup \{b\}$. If b is not the shortest action in A and b does not consume any resource other than time, then A is always better than B. (3) Suppose CAS A and CAS B contain the same set of actions except action b. If action b is a member of both CASs but action b in A has a shorter remaining time than the action b in B, then CAS A is always better than CAS B.

The first rule states that if an action consumes only time, it does not harm to start it as early as possible. The second rule states that even if an action is deleting a proposition, it is still safe to start it as early as possible as long as it is not the shortest action in the action set. If it is not the shortest action, we still have a chance to abort it so that it will not delete any propositions. The third rule states that there is no need to abort the currently executing action and restart it immediately. The "better" relation is transitive. Therefore, these three rules can be applied more than once to find out if one CAS is better than another.

We tested CPOAO* with the pruning technique based on the better relation. We ran the problems with execution time limits of 20 and 40 given as part of the planning problem. The results are reported in the "pruning only" columns of Table 1. The results show that the current implementation of CPOAO* can return a plan for up to 15 locations. As expected, the execution time and the number of nodes generated increase exponentially as the time limit increases due to the large branching factor at each internal node.

In order to guide the search, we implemented a heuristic that uses a *reverse plan graph (rpgraph)* which is a relaxed plan graph constructed with the goal propositions at the first level. We use the rpgraph to estimate the expected rewards at a state. At the beginning of the planning process, we create an rpgraph for each goal by first placing the goal proposition at the initial level. We then create alternating levels of action and proposition nodes. For each proposition at the last level, an action that can achieve this proposition is inserted into the next level. For each action, the results are linked to the previous proposition level, and the preconditions are linked to the next proposition level. Each proposition node contains a resource list indicating the resources required to reach the goal from this proposition node. The resource levels are updated using the resource requirements of the actions. The expansion of the rpgraph stops when the level of any resource exceeds the value present in the initial state. After the rpgraph is constructed we delete the proposition nodes and the corresponding actions if the path from the goal contains other proposition nodes for the same propositions that require lower levels of resources. The idea behind the rpgraph can be contrasted to labelled uncertainty graphs (LUGs) (Bryce, Kambhampati, and Smith 2006). A LUG represents multiple possible worlds in a compact way whereas an rpgraph represents multiple possible rewards.

When a new state s is generated during planning, we find a reduced rpgraph for each goal by first finding the proposition nodes that correspond to the propositions in s. Each proposition node that has resource levels at or below the resource levels of s is marked as *enabled.* If all the preconditions of an action are enabled, then its parent propositions are also enabled. All the non-enabled nodes are deleted to obtain the reduced rpgraph.

We then calculate the upper bound of the probability of achieving goal g from state s. We start at the leaf nodes and propagate probabilities upward by using action probabilities and updating the resource lists to reflect the maximal resource levels at the proposition nodes. The expected reward at state s is computed by adding the expected rewards for each goal using the formula:

$$E(s) = \sum_{g \in \text{GOALS}} P(g) \times R(g),$$

where $P(g)$ is the probability of goal g computed from the corresponding rpgraph, and $R(g)$ is the reward of g as given in the planning problem. The results using the rpgraph heuristics are reported in last two columns of Table 1. The results show up to one order of magnitude decrease in the number of nodes generated as compared to CPOAO* with pruning only. In the test domain, this heuristic is admissible because time is the only resource and the rewards are never underestimated.

CPOAO* implicitly solves an oversubscription planning problem by maximizing the expected total reward of the initial state. Heuristics approaches for solving deterministic oversubscription problems (Smith 2004; Benton, Do, and Kambhampati 2005) can be studied in our framework.

Problem	Acyclic Paragraph		base CPOAO*		CPOAO* with pruning only				CPOAO* with pruning and rpgraph			
	T = 10	T = 20	T = 20		T = 20		T = 40		T = 20		T = 40	
	ET	ET	NG	ET	NG	ET	NG	ET	NG	ET	NG	ET
5-5-5	< 0.1	5	164	< 0.1	102	< 0.1	1886	< 0.1	36	< 0.1	809	< 0.1
5-5-10	3319	–	224	< 0.1	105	< 0.1	11141	5	45	< 0.1	2648	< 0.1
5-10-5	2	15	376	< 0.1	183	< 0.1	12081	5	52	< 0.1	1248	< 0.1
5-10-10	–	–	933	< 0.1	439	< 0.1	99441	172	74	< 0.1	11120	5
10-10-8	–	–	272	< 0.1	79	< 0.1	1269	< 0.1	45	< 0.1	375	< 0.1
10-10-21	–	–	914	< 0.1	252	< 0.1	21577	12	99	< 0.1	3153	1
10-17-8	–	–	293	< 0.1	152	< 0.1	6775	1	46	< 0.1	760	< 0.1
10-17-21	–	–	965	< 0.1	321	< 0.1	75727	87	1030	< 0.1	10940	4
12-12-12	–	–	530	< 0.1	120	< 0.1	2832	1	56	< 0.1	662	< 0.1
12-12-23	–	–	1170	< 0.1	287	< 0.1	27914	23	86	< 0.1	5269	2
12-21-12	–	–	501	< 0.1	204	< 0.1	11315	3	53	< 0.1	1391	< 0.1
12-21-23	–	–	1230	< 0.1	380	< 0.1	85203	99	116	< 0.1	11833	5
15-16-14	–	–	1067	< 0.1	180	< 0.1	6306	2	60	< 0.1	1232	1
15-16-31	–	–	1941	< 0.1	417	< 0.1	49954	61	93	< 0.1	7946	2
15-28-14	–	–	1127	< 0.1	347	< 0.1	29760	18	71	< 0.1	2460	< 0.1
15-28-31	–	–	2345	< 0.1	694	< 0.1	–	–	146	< 0.1	19815	8

Table 1: Experimental results for various problems and heuristics. T: Time limit. NG: The number of nodes generated. ET: Execution Time (sec.). A "–" indicates that the problem was not solvable within 5 minutes.

Conclusion

We have presented the design and evaluation of CPOAO*, an algorithm that can deal with durative actions, concurrent execution, over-subscribed goals, and probabilistic outcomes in a unified way. We describe a modular, heuristic-based plan generation framework that has two contributions: the use of concurrent actions to achieve the same goal, and the marking of actions that should be terminated. Our current implementation includes heuristics that show potential by greatly reducing the search space. The framework enables further research in (1) both admissible and nonadmissible search heuristics, (2) explicit, possibly domain dependent theories of how to detect useless actions in parallel, durative domains, (3) domain model extensions such as continuous resources, and (4) action model extensions such as actions that can terminate midway and might leave the system in unknown states.

References

Benton, J.; Do, M. B.; and Kambhampati, S. 2005. Over-subscription planning with numeric goals. *Proc. IJCAI-05*.

Bonet, B., and Geffner, H. 2005. mGPT: A probabilistic planner based on heuristic search. *J. of Artificial Intelligence Research* 24:933–944.

Boutilier, C.; Dean, T.; and Hanks, S. 1999. Decision theoretic planning: Structural assumptions and computational leverage. *J. of Artificial Intelligence Research* 11:1–94.

Bresina, J. L.; Dearden, R.; Meuleau, N.; Ramakrishnan, S.; Smith, D. E.; and Washington, R. 2002. Planning under continuous time and resource uncertainty: A challenge for AI. In *Proc. UAI-02*, 77–84.

Bryce, D.; Kambhampati, S.; and Smith, D. 2006. Planning graph heuristics for belief space search. *J. of Artificial Intelligence Research* 26:35–99.

Hansen, E. A., and Zilberstein, S. 2001. LAO*: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence* 129((1-2)):35–62.

Haslum, P., and Geffner, H. 2001. Heuristic planning with time and resources. In *Proc. 6th European Conference on Planning*.

Kushmerick, N.; Hanks, S.; and Weld, D. 1995. An algorithm for probabilistic planning. *Artificial Intelligence* 76:239–86.

Little, I., and Thiebaux, S. 2006. Concurrent probabilistic planning in the graphplan framework. In *Proc. ICAPS-06*, 263–272.

Mausam, and Weld, D. S. 2008. Planning with durative actions in stochastic domains. *J. of Artificial Intelligence Research* 31:33–82.

Mausam; Benazera, E.; Brafman, R.; Meuleau, N.; and Hansen, E. A. 2005. Planning with continuous resources in stochastic domains. *Proc. IJCAI-05*.

Nilsson, N. J. 1980. Principles of artificial intelligence. *Tioga Publishing*.

Pfeffer, A. 2005. Functional specification of probabilistic process models. In *Proc. AAAI-05*, 663–669.

Smith, D. E., and Weld, D. 1999. Temporal planning with mutual exclusion reasoning. In *Proc. IJCAI-99*, 326–333.

Smith, D. E. 2004. Choosing objectives in over-subscription planning. In *Proc. ICAPS-04*.

Younes, H. L.; Littman, M. L.; Weissman, D.; and Asmuth, J. 2005. The first probabilistic track of the international planning competition. *J. of Artificial Intelligence Research* 24:851–887.