Decision-Theoretic Planning in the Graphplan Framework

Gilbert Peterson and Diane J. Cook

University of Texas at Arlington
Computer Science and Engineering
Planning and Learning Laboratory
Box 19015, Arlington, TX 76019-015
{gpeterso, cook}@cse.uta.edu

Abstract

We have developed a decision-theoretic planner based upon the Graphplan planning algorithm, DT-Graphplan. DT-Graphplan reasons about probabilities, costs, and rewards at a propositional level, reconstructing limited state information. We are applying the planner to our robot task architecture to function on a miniature golf domain. By incorporating decision theory into planning, we seek to reduce the representational gap between behavior-based robotic controllers and constraint-based symbolic planners.

Introduction

This paper discusses DT-Graphplan, a decision-theoretic planner that we use as the planning and sequencing layers for a layered robotic architecture. By using a planner at both the symbolic planner level and the sequencer level we hope for a reduction of the work needed to reconfigure a robot for a new task. We will verify this by creating one set of behavior controllers for our robots and demonstrating the effectiveness of the controllers on multiple diverse plans.

The method of developing task control software for robots we address is a layering approach. This approach generally consists of a symbolic planner, a task sequencer, and a behavioral robotic controller. The task sequencer is responsible for breaking a command from an abstract plan into select robot-level actions and behaviors to execute. This representation leads to a robust functioning software control for a robot on a single task [Bonnasso and Kortenkamp, 1996]. When it is necessary to reconfigure the robot for a new task, the sequencer receives new sequences and the behavior controller gets additional behaviors.

The use of decision theory to guide the behavior of a robot is a familiar concept. We expand on this idea, using decision theory at a higher reasoning level to generate a plan. The use of planning over other methods stems from the desire to generate a quick workable set of actions, comprehendible to both the robot and user for domains with dynamic operating conditions. We hope to show that this is effective in our miniature golf domain in comparison to a strictly behavioral approach.

Our planner, DT-Graphplan, adds decision-theoretic reasoning to the framework of the Graphplan algorithm, extending the planner to handle probabilistic actions as well as utility driven search [Blum and Furst, 1995]. DT-Graphplan returns the first acceptable plan found meeting a user-defined threshold, for a user specified goal set. This extends recent work conducted on Graphplan to handle probabilities [Blum and Langford, 1999].

The development of DT-Graphplan was undertaken because decision-theoretic methods represent certain elements in robotic domains better than symbolic methods. For instance, in the miniature golf domain, that is the application for our robot architecture, the robot has the choice between picking up the ball and dropping it in the cup, or attempting to push the ball into the cup, Figure 1. By picking up and dropping the ball, the robot suffers a two-stroke penalty but has a large probability of achieving the task of getting the ball into the cup. If the robot attempts to push the ball into the hole, the stroke penalty is not as high, but the probability of success is not as great. Based on the cost of pushing with its probability of success and the cost of depositing the ball with its higher probability of success, the planner makes a choice of which plan to pursue.

Alternatively, another general application of decision-theoretic comparison trades off between risk and reward, comparing the risk of an action with its chance of success to another. The agent must choose between an a risky move with a low probability of success but great potential rewards compared with the utility of a conservative move with a higher probability of success and moderate potential rewards.

Our approach differs in that we developed a decision-theoretic planner to replace the symbolic planner. We use a decision-theoretic planner to formally reason about the uncertainty that is inherent in robot tasks, while retaining the ability to reason at a high level about the current world and goals.

This paper discusses the DT-Graphplan algorithm. The next section briefly covers the Graphplan algorithm. Section 3, discusses the method DT-Graphplan uses to perform graph building and search. Following this a subset of the miniature golf domain used as an example to

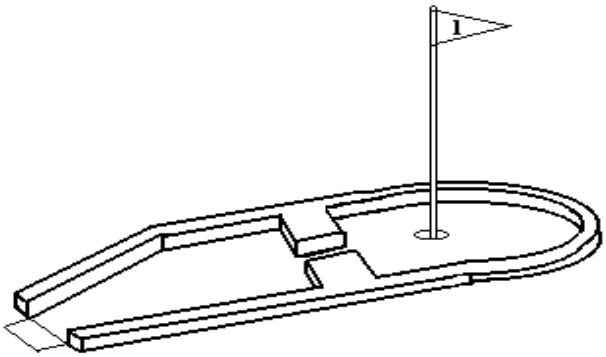


Figure 1: Miniature Golf.

illustrate DT-Graphplan is described. The final sections compare related work to DT-Graphplan and discuss possible future work.

Graphplan Background

The Graphplan algorithm written by Blum and Furst, plans by alternately expanding a planning graph and extracting a plan solution [Blum and Furst, 1995]. The planning graph is a series of layers alternating between proposition nodes and action nodes. The initial layer consists of proposition nodes that represent the initial plan condition. For each action node, directed edges lead from the proposition nodes that are the preconditions of the action to the action node, and then from the action to the proposition nodes that are the effects of the action.

During graph building, the graph retains binary mutual exclusion information (a mutex relation). This information speeds search by tracking the propositions that interfere with each other and can not exist simultaneously. The mutex relation also serves to preserve state information, two propositions which are mutex can not exist simultaneously. Two action instances at a level are mutex if they interfere - one action deletes a precondition or effect of another, or show competing needs – the actions have preconditions that are mutually exclusive at the previous level. Two propositions at a level are mutex if all ways of achieving the propositions (actions on the previous level) are mutex.

Graph expansion halts on a proposition layer when each element of the goal condition is present and none are pairwise mutex. Graphplan then searches the planning graph for the plan solution using a backward chaining search. The search results in a path from the goals to the initial condition consisting of only non-mutex actions. If search finds no plan, then graph expansion and search continues in an iterative fashion.

We chose to write our planner on top of the Graphplan algorithm due to the efficiency with which it locates a plan. In generating the new planner, the graph framework of propositions and actions tailored our calculation of probability and utility over states and actions.

Decision-theoretic Graphplan (DT-Graphplan)

The concept behind DT-Graphplan is similar to Graphplan. Create a graph with all of the possible combinations of actions from an initial condition until reaching a goal condition and then backward-chain search the graph for an acceptable plan. The DT-Graphplan algorithm accommodates decision theory, in allowing for probabilistic propositions and uncertain action effects. DT-Graphplan's objective is the generation of a plan meeting an atemporal threshold utility.

One approach to incorporating decision theory with the Graphplan algorithm creates state and action layers instead of centering on propositions and actions. An investigation of this representation conducted by Boutillier uses the Graphplan algorithm to reduce the solution space for an MDP solver [Boutillier, et. al., 1997].

One of the advantages of the Graphplan algorithm is that it relieves the planner of the burden of processing entire states at every node and instead reasons about individual propositions. DT-Graphplan extends this approach with decision-theoretic planning, and maintains the proposition and action layers from the original algorithm. In order to obtain possible future states for calculations, we rely on the generated mutex relation information.

Traditionally, rewards are determined based on a set of features of a state. Instead of assigning rewards to a state, singled out propositions in the plan graph receive the reward.

Since DT-Graphplan does not explicitly reason about a world state, the utility of a state is reconstructed. A state's utility depends on the rewards of the propositions in the state multiplied by probabilities of the propositions minus costs of the actions that brings us to the state.

In addition, DT-Graphplan plans under the assumption that all of the propositions for a state are independent. The current version of DT-Graphplan does not incorporate joint probability distributions. We are in the process of incorporating this into the algorithm.

DT-Graphplan Propositions

In Graphplan, each proposition's presence in the graph signifies its validity at that point. To accommodate negative propositions, the domain adds a negative version of the proposition and declares it mutex with the positive version.

DT-Graphplan represents each proposition as a probability value [0..1] and a utility value. The probability value represents the probability the proposition is currently true. One minus this value represents the probability the proposition is currently false. The utility value is the sum of the rewards earned to this point in the plan and the costs of previous actions.

Initial World Conditions

DT-Graphplan's initial world definition consists of a set of probabilistic propositions. The propositions in the initial set represent the probability of the existence of the propositions in an initial world state. Although not part of the initial world conditions, the domain also consists of a utility threshold, which represents the minimum acceptable utility for a generated plan. The domain includes a set of reward statements. The reward statements stipulate the amount of utility earned for the existence of a proposition in the graph.

Because of the assumption of proposition independence, DT-Graphplan does not distinguish between possible worlds. An example of this is the "bomb in the toilet" domain with two packages [McDermott, 1987]. In this domain, there is a fifty percent chance that package 1 is unsafe and package 2 is safe, and a fifty percent chance of the opposite. The condition of each package depends upon the other. To execute this domain in DT-Graphplan, the probability that each of the packages is safe is set at fifty percent. This represents the overall initial state of the world. However, after dunking one of the packages in the toilet, the probability independence assumption leads to the inability of DT-Graphplan to generate the 50% probability that the bomb is defused

Planning Graph Expansion

DT-Graphplan generates only one plan graph comprising all of the possible propositions with their various probabilities and utilities. Plan graph expansion occurs as in Graphplan, adding all of the actions possible given the propositions available at the current time step.

DT-Graphplan makes use of the addition of partially-factored expansion to handle conditional action effects [Koehler, et al., 1997]. This allows for context dependent action effects. Partially-factored expansion expands the conditional outcomes as the action is applied to the graph, expanding only the branches valid at the time.

Execution of an action in a state occurs when all of its preconditions are met. Since DT-Graphplan does not maintain strict state information, the insertion of an action depends on the presence of the precondition propositions. The propositions of the precondition must all be non-mutex; it must be possible for them to all exist at the same time, and therefore in the same state. The exact state information remains unknown at the time but is determinable. The important point is that all actions are applied to every possible state.

During graph expansion, action effects lead to additional, possibly new propositions. Each action inserted into the graph generates new propositions added to the graph at the next time step. In addition, a noop action caries each existing proposition at a given time step are to the next time step.

At any given time step, multiple occurrences of a given proposition may exist, with different probabilities and/or utilities. Each of these different propositions represents one possible condition of the world after executing a series of actions. All possible probability and utility proposition combinations exist because the graph represents all of the possible action applications from the initial state.

For each proposition in existence, if an action updates the probability or utility, the graph will generate a duplicate proposition with new probability and utility values. If a proposition exists with the same probability and utility, the action adding the proposition just references the existing proposition. Each action that affects the proposition at the next time step may also affect the additional propositions.

Because of the possible multiple similar propositions, an additional mutex rule exists to maintain state information and speed searching. The additional mutex rule specifies that each of the propositions with the same name and different probabilities and utilities are mutex with each other. This mutex rule behaves similarly to the one added to the Graphplan algorithm to accommodate negative propositions. Two propositions with the same name should not exist concurrently. DT-Graphplan retains the action mutex relations of interference of effects, and competing needs from the original Graphplan algorithm. DT-Graphplan also maintains the proposition mutex relation whereby two propositions are mutex if all ways of reaching the propositions are mutex.

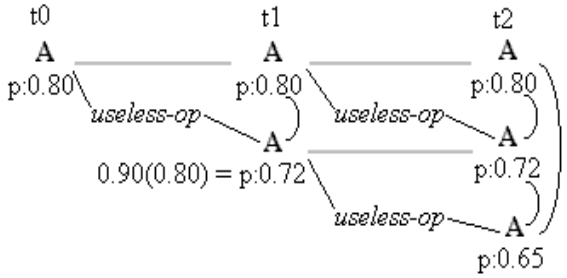


Figure 2: Example graph expansion.

This algorithm results in a graph with a greater branching factor and many more nodes per layer than a graph for a classical symbolic domain. Figure 2 shows an example of how proposition fanning occurs, at each successive proposition level there are more versions of the proposition A. In the example, there is an action 'useless-op' which given the precondition of proposition A has an outcome effect of 0.90 proposition A. From the figure, the outcome of 'useless-op' at time step 0 is the precondition probability of A (0.80) combined with the effect probability (0.90) resulting in proposition A at time step 1 having a probability of 0.90(0.80) = 0.72. At time step 1 there are two possible versions of proposition A, each possessing different probabilities and are mutex with each other. The light gray lines are the noop actions and the arcs between the propositions are mutex conditions. The figure shows no action mutexes.

Calculating Utilities

The probabilistic calculation of each action depends on the probability of each precondition and the action effects. The evidence of an action is a conjunction of the probability of each of its preconditions. Because of the independence assumption, this is just the product. Each action effect has a probabilistic adjustment multiplied with the evidence generates the outcome probability for the effect proposition.

The utility calculation of an action begins with determining the pre-existing utility value. The pre-existing utility is the sum of the precondition utilities. If more than one precondition received reward for the same condition, then the pre-existing utility only includes the largest of these rewards.

The pre-existing utility decreases by the execution cost of the action. The execution cost of an action is the estimated expense of executing the action. The utility of each of the action effects is set to the pre-existing utility.

Reward assignment follows the calculation of the pre-existing utility. As discussed earlier, the propositions instead of specific states propagated rewards. Reward assignment first verifies that there is a reward condition with the same proposition name as the new proposition. Reward calculation increases the utility of a proposition by the reward amount scaled by the proposition's probability.

After calculating the utility and probability values for an action effect, the proposition is added to the graph. If a version of the proposition does not exist with the same probability and utility values, graph expansion inserts a proposition with the new values. If an identically valued proposition exists, the graph updates the action effect to point to this preexisting proposition.

An action not only generates new propositions but also updates the decision-theoretic values of existing propositions. By repeating an action, the probability of a proposition may increase. For example, in the "moat and castle" domain, each successive execution of the 'build-castle' action increases the chance of the castles successful construction (Majercik and Littman, 1987). DT-Graphplan conducts probability propagation for each of the executable actions to achieve this affect.

When the effect of an action is not a precondition of the same action, and an instantiation of this proposition exists before action execution, the proposition's probability propagates. Propagation consists of inserting an additional action node, identical to the previous. This new action node includes an additional precondition leading from the previous instantiation of the effect proposition. Additionally, propagation replaces the effect proposition with an updated one. The new effect's probability is scaled by the probability of the proposition before the action executes. The example in the Example Domain section illustrates this process in detail.

After constructing each graph layer, there is the opportunity for graph expansion to halt. Graph expansion halts when the number of layers reaches a cutoff point. The conditions for cutoff occur when the last two layers are identical, or the goal conditions exist with a decision value greater than the given threshold.

Once the graph contains propositions meeting the goal conditions a number of goal sets are generated, each with a number of elements equal to the goal condition. DT-Graphplan sometimes generates multiple goal sets with a utility greater than the threshold at the stopping depth. These goal sets ordered largest to smallest are searched individually. The user can set a desired percentage of these goal lists to search before resuming the iterative deepening phase of the algorithm.

During iterative deepening, the algorithm alternates between adding an additional layer to the graph and searching the extended graph for a plan. This continues until the discovery of a plan or the graph reaches the maximum depth allowed. At the maximum graph depth, all of the generated goal lists meeting the threshold are searched. If there is still no plan discovered, then planning stops and the algorithm notifies the user that no plan exists.

The search phase of DT-Graphplan differs from Graphplan only in the action selection phase. In Graphplan, the last action adding a proposition is the first selected. In DT-Graphplan, the algorithm instead selects the action with the highest utility.

Example Domain

The example uses a small variation of the miniature golf domain we are applied to the robot architecture. In this domain, the initial condition is that the golf ball is on the tee. There is one reward of 2.0 for getting the ball in the cup. The actions are push to green, push-to-hole-in-one, and putt-to-cup. The push-to-green action's precondition is ball-on-tee, with the effects being to remove ball-on-tee, and add ball-on-green. The push-to-hole-in-one action's precondition is ball-on-tee with the effects of removing ball-on-tee, adding ball-on-green, and adding a possibility for ball-in-cup. The putt-to-cup action's precondition is ball-on-green and results in a chance to get the ball-in-the cup. The three action definitions in full are in Figure 3.

```
push-to-hole-in-one 0.70    ; op-name op-cost
 :p 0.70 > ball-on(tee)     ; preconditions prob. of ball-on-tee > 0.70
 :e + 1.00 ball-on(green),  ; effects incr. prob. of ball-on-green by 1.0
   - 0.00 ball-on(tee),     ;   decr. prob. of ball-on-tee by 0.00
   + 0.30 ball-in-cup().    ;   incr. prob. of ball-in-cup by 0.30

push-to-green 0.65          ; op-name op-cost
 :p 0.70 > ball-on(tee)     ; preconditions prob. of ball-on-tee > 0.70
 :e + 1.00 ball-on(green),  ; effects incr. prob. of ball-on-green by 1.0
   - 0.00 ball-on(tee).     ;   decr. prob. of ball-on-tee by 0.00

putt-to-cup 0.40            ; op-name op-cost
 :p 0.70 > ball-on(green)   ; preconditions prob. ball-on-green > 0.70
 :e + 0.85 ball-in-cup().   ; effects incr. prob. of ball-in-cup by 0.85
```

Figure 3: Miniature Golf Actions.

Figure 4 shows the planning graph generated for this small miniature golf domain for a utility threshold of 0.69. The arcs on the first layer show the existing mutex constraints. The dark outlined facts and actions represent

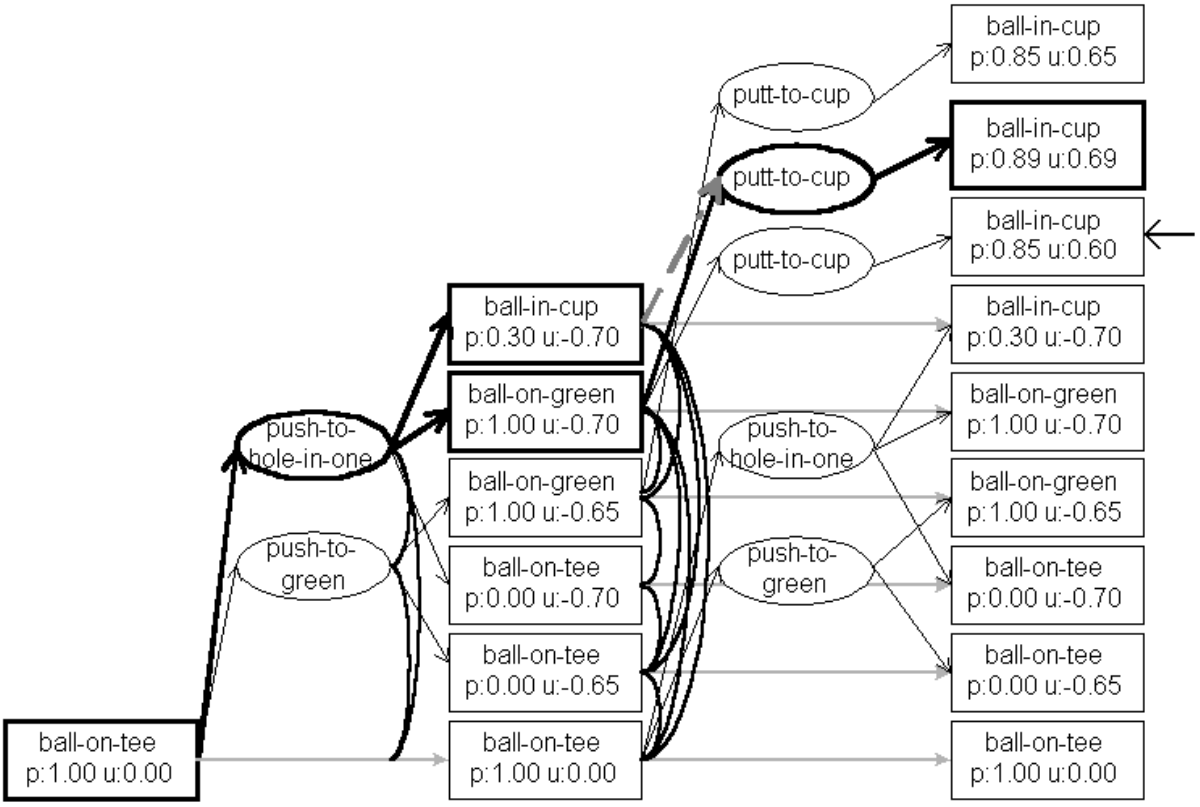


Figure 4: Miniature Golf Domain with threshold of 0.69.

the final plan elements. The final plan consists of push-to-hole-in-one and then putt-to-cup. Increasing the utility threshold beyond 0.69 causes the planning graph to extend an additional layer and results in a three-step plan of push-to-green, putt-to-cup, and putt-to-cup with a utility of 0.90.

To analyze how DT-Graphplan propagates probabilities and utilities, let's look at the putt-to-cup action in the final plan. The precondition for putt-to-cup is ball-on-green, the one used in this action has a probability of 1.00 and a utility of –0.70. Since this is the only precondition for the action, the evidence for the action is 1.00. Note that the existing utility of –0.70 is the cost of the putt-to-hole-in-one action that generated the ball-on-green proposition.

The putt-to-cup action has a single effect, that of establishing the ball-in-cup proposition with an outcome probability of 0.85. This leads to the generation of the proposition ball-in-cup with a probability of 0.85. Since there is a reward of 2.0 for getting the ball in the cup the utility is 0.85*2.0 – the accrued action costs of 1.10. This results in a utility of 0.85*2.00–1.10= 0.60.

However, 0.60 is not the utility of the proposition in the final plan, it is pointed at by the arrow. The proposition included in the plan makes use of the existing probability of the ball-in-cup from the previous time, 0.30. The gray dashed line in the figure shows this probability propagation operation. The probability of the ball-in-cup proposition is the existing probability 0.30+ the remaining probability scaled by the action's chance of success, (1.0–0.30)*0.85, yielding 0.30+0.595 = 0.895. The reward is calculated based on this probability resulting in the utility of 0.895*2.00-1.10= 0.69.

Related Work

One method of incorporating decision theory in planning is to represent an action based upon a transition matrix, representing possible changes between all of the world states. This is the method used in traditional Markov Decision Processes (MDP). An MDP models a dynamic system, where the state of the system is represented in terms of a joint probability distribution over the state of the system given the system state at the previous time slice. Additionally, an MDP system may observe a noisy function of the state variables, and have incomplete or imprecise state information. This condition generates a Partially-Observable MDP (POMDP). Partial observability is a similar objective as in DT-Graphplan; however, POMDP uses a different language by representing probability over the state. DT-Graphplan reasons with a proposition representation, representing probabilistic steps as a conditional probability distribution over a set of outcomes based upon the preconditions.

Buridan, one of the first probabilistic planners plans under conditions of partial observability [Draper, et al., 1994]. C-Buridan extends Buridan to generate contingent plans. DT-Graphplan differs from Buridan by not

incorporating initial world state probabilities. A world state is a collection of propositions in the initial state. In Buridan, it is possible to have multiple possible initial states where the chance that the initial state is one of these possible worlds is included in the domain description.

MAXPLAN, an additional probabilistic planner, compiles the planning problem into an E-MAJSAT problem and solves for a contingent plan [Majercik and Littman, 1998]. MAXPLAN converts the planning problem into a single step plan where variables represent each probability. Solving the generated satisfiability matrix also solves for the unknown probabilities. The developers of MAXPLAN in comparing it to Graphplan show that although not as fast compares favorably. The MAXPLAN planner is not as fast as Graphplan due to the use of mutex rules in Graphplan prunes a great deal of the search space that the satisfiability solution does not.

Conformant Graphplan (CGP) extends Graphplan, with the addition of multiple planning graphs, one for each possible world [Anderson, et al., 1998]. CGP builds a separate graph for each of the possible worlds and applies each action to all the worlds [Anderson, et al., 1998]. Sensory Graphplan (SGP) extends CGP by adding sensory actions [Weld, et al., 1998]. SGP tries to find a solution that works in all of the possible worlds. If a solution does not work then it uses the sensory actions to try to distinguish between the worlds. Once the worlds are distinguishable then each world gets a separate subgoal. The separate subgoals serve to select an alternative course of action for each world.

The PGraphplan planner, similarly to C-Buridan, produces a contingent probabilistic plan [Blum and Langford, 1999]. Both PGraphplan and DT-Graphplan originate with the Graphplan algorithm. PGraphplan's methodology differs from Graphplan in that instead of performing a backward-chaining search, search is a forward-chaining process to find an optimal contingent plan. PGraphplan uses the forward-chaining search phase to both propagate probabilities and find a plan.

All of these planners use a probabilistic model similar to MDP's. Each state has a probability associated with it. Actions result in new states with a set probability. Each proposition in the state represents the existence of an element in the state. DT-Graphplan differs from these approaches in that it reasons using propositions. Each proposition has its own probability signifying the probability it may exist at that point in time.

If two propositions are not mutex with each other and search shows that a series of actions leads to both then they may exist in the same future state. Each proposition has a maximum probability that it exist. The product of the proposition's probabilities represents the maximum probability they both exist at the same time. We are in the process of completing the addition of joint probability distributions to include other methods of calculating these state probabilities for a set of propositions.

Empirical Results

In this section, we compare DT-Graphplan with two existing probabilistic planners. The comparison with Buridan gives a general measure of the performance of DT-Graphplan to compare with other planners. Since DT-Graphplan and PGraphplan both use the Graphplan algorithm, testing of the two algorithms outlines the differences in the alternative approaches. We then demonstrate how DT-Graphplan goes beyond both planners' capabilities by planning with utilities.

One popular probability domain is the "moats and castles" domain [Majercik and Littman, 1998]. In this domain, the objective is to build a castle on the beach. During construction, a wave may come and destroy the castle. In order for a wave not to wash the castle away, it is best if the agent digs a moat first. There are two actions in this domain. One action is 'dig-moat' which has a 0.5 probability of actually creating a moat. Additionally, there is the 'build-castle' action. If a moat exists, the 'build-castle' action succeeds with a probability of 0.67. If no moat exists, the build action only succeeds 0.25 percent of the time. If the 'build-castle' action fails then the moat is destroyed.

We alter the domain definition to increase the number of possible solutions. Instead of having a single moat depth, there are four. Each 'dig-moat' action increases the depth of the moat by one, from no-moat to moat-depth-of-four. Once the depth of the moat reaches four, the 'dig-moat' action does not alter the moat's depth. As the moat's depth increases the probability of the 'build-castle' action succeeding increases. The probability distribution for 'build-castle' becomes 0.25 for no-moat, 0.46 for moat-depth-of-one, 0.60 for moat-depth-of-two, 0.70 for moat-depth-of-three, and 0.75 for moat-depth-of-four.

To run the "moat and castle" domain and other probabilistic domains on DT-Graphplan, the domain description must set all action costs to 0.00, and no reward conditions represented. We compared the performance of the "moat and castle" domain on PGraphplan, Buridan, and DT-Graphplan. We varied the number of castles in the domain from 1 to 5; Table 1 represents the run time results comparing Buridan and DT-Graphplan. Figures 5 and 6 graph the results of DT-Graphplan and PGraphplan in the execution time and number of nodes produced.

Table1: 'moat and castles' Results.

	1 castle	2 castles	3 castles	bomb and toilet
DT-Graphplan	0.000 s	0.005 s	0.005 s	0.000 s
Buridan	1.050 s	79.712 s	*	*

*: for 3 castles and more and the "bomb and toilet /w clogging" domain, Buridan took over 5 minutes to find a solution.

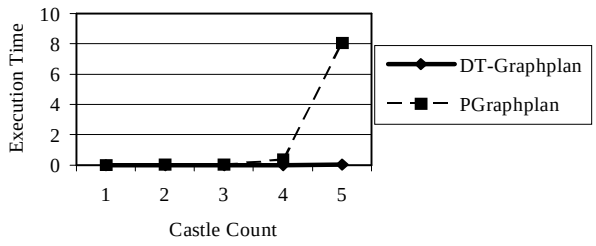
As Table 1 shows, the Buridan planner took over five minutes to solve the castle problems with three or more castles. Buridan also took longer than 5 minutes to solve the "bomb and the toilet" domain with clogging and two packages. As discussed earlier, the "bomb and toilet" domain representation in DT-Graphplan does not include world state information, instead representing each

package as having a fifty percent probability of containing a bomb. The comparison does show that DT-Graphplan quickly finds solutions to all of these small problems.

Figure 5 and 6 compare the execution time and node generation of DT-Graphplan with PGraphplan. Figure 5 shows the execution times of the two planners. The figure demonstrates that of the two planners DT-Graphplan locates a plan in the least amount of time and does not suffer as badly as the domain size increases. For each larger domain, the graph generated grows larger; generating more states to search.

One reason for the speed difference is the method used to search the generated graph. DT-Graphplan performs backward-chaining search and ignores a greater percentage of states than the forward-chaining search of PGraphplan.

Figure 5: Execution Time Comparison for 1-5 Castles.



In order to perform backward-chaining search, DT-Graphplan generates significantly more nodes. The additional nodes serve to propagate probability information from the initial conditions to the goals, as explained with the example domain. PGraphplan instead of generating additional nodes waits until the forward search phase to conduct this propagation. Figure 6 shows the comparison of the number of nodes generated between DT-Graphplan and PGraphplan.

One reason PGraphplan performs forward-chaining search is to generate a contingent plan. This differs from DT-Graphplan, which generates a blind plan.

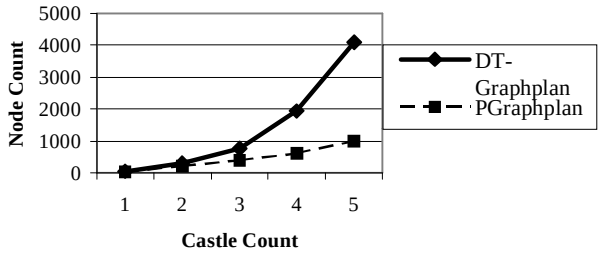


Figure 6: Node Generation for each Castle Domain.

Both C-Buridan and PGraphplan produce contingent plans while DT-Graphplan does not. The probabilistic planners create a contingent plan that for all action success/fail branches has a probability that meets or exceeds a desired threshold for a specific goal condition. Whereas the plan generated by DT-Graphplan is a single series of actions that leads to the greatest reward. We chose to tradeoff at the cost of possible contingent planning for the efficiency of Graphplan's backward-chaining search. Removing contingent planning does not interfere with the purpose of DT-Graphplan, which incorporates uncertainty in the form of probability and utility. We performed comparative testing with the probabilistic planners to include a comparison to existing planning systems.

One reason for developing DT-Graphplan was to incorporate utility into the description of the domain. One of the capabilities in this respect that we developed into DT-Graphplan is the ability to locate a plan that meets a given threshold for a goal set. This means that the user can define a domain with a set of rewards and a desired utility threshold and the planner will find the first plan meeting the threshold. In these domains, rewards represent a general set of desirable events, some of which may not be concurrently possible.

We have a larger version of the miniature golf domain from Figure 3 that has two levels. The golf course consists of a starting layer, and the ending layer that has the cup. A gutter and a chute connect the starting layer to the ending layer. The difference between using the chute versus the gutter is the chute is harder to get the ball into but gets the ball closer to and possibly into the cup. The initial domain consists of the ball on the starting layer and the goal is 'ball-in-cup'. One reward condition exists for getting the ball into the cup. Figure 7 describes most of the actions, the actions to pickup and drop the ball, and move between levels are omitted because of their lack of uncertainty.

```
push-to-gutter 0.30
:p 0.70 > ball-on(l1) 0.70 > robot-on(l1) 0.10 < robot-holds-ball()
:e + 1.00 ball-on(l2),
  - 0.00 ball-on(l1),
  + 0.20 ball-in-cup().

push-to-chute 1.15
:p 0.70 > ball-on(l1) 0.70 > robot-on(l1) 0.10 < robot-holds-ball()
:e + 1.00 ball-on(l2),
  - 0.00 ball-on(l1),
  + 0.90 ball-in-cup().

putt-to-cup 0.30
:v ?l level
:p 0.70 > ball-on(?l) 0.70 > robot-on(?l) 0.90 > cup-on(?l) 0.10 < robot-holds-ball()
:e + 0.90 ball-in-cup(?l).
```

Figure 7: Larger Miniature Golf Domain.

Setting the threshold for this larger miniature golf domain at 0.65, the resulting one step plan is to push the ball to the chute, which results in the ball being in the cup. By increasing the utility threshold to 0.70, the resulting plan extends to three steps with an overall utility of 1.23. This plan pushes the ball to the gutter, getting the ball to the second layer. The robot then moves to the second layer and putts the ball into the cup. By increasing the threshold even more, the plan generated has a utility of 1.36 and is the previous plan with an additional putt-to-

cup action, this is the highest utility plan possible.

Future Work

The additions we plan to make to DT-Graphplan consist of joint probability distributions and graph pruning. We initially postponed the handling of joint probabilities because of the increase in complexity they incur. In order to calculate joint probabilities, each proposition's probability calculation depends on all of the other propositions. One method of handling this is to add propositions for all possible joined propositions. This method increases the number of propositions at each level exponentially. Because there are more nodes, the time required to build and search the graph will also increase.

A second method to incorporate joint probabilities is to have the user predeclare any of the joins required by the domain. During graph building, the addition of joined propositions occurs only for those declared in the domain description. This reduces the number of propositions to add and search but relies on the user's insight into the domain. We are in the process of incorporating this method into DT-Graphplan. The user decides how densely interconnected the propositions are by including a network in the domain description which dictates how the propositions are interconnected and the manner the probabilities affect each other.

During graph construction, many nodes added to the planning graph play no role in the final plan. As shown in the largest 'moat and castle' domain, where at the solution depth there are 4095 nodes. The planner also makes limited use of the utility information available. This occurs in the plans for the larger miniature golf domain, which makes no use of the pickup/drop actions due to their high costs. All of the propositions produced by these actions only serve to waste space. We propose to add a system to reduce the number of propositions generated. This system would expand a set best percentage of propositions based on their utility, marking the remaining for later expansion. If search finds no plan, then the planner revisits and expands the marked nodes.

Conclusion

We have developed a time efficient decision-theoretic planner. Our planner based on the fast Graphplan algorithm finds the first plan meeting a user-defined threshold. The plan domains are searchable based on a user-defined goal or utility only. The actions used by the planner handle uncertain initial conditions and incorporate conditional outcome effects. Sets of rewards and action costs dictate the assessment of utility.

We have compared DT-Graphplan to existing probabilistic planners, showing the difference in the approaches and generated plans. Our results show an additional expressiveness of incorporating utility over these probabilistic planners. In the interest of increasing this expressiveness, we are in the process of extending the DT-Graphplan algorithm to incorporate joint probability distributions. This inclusion would remove the strong assumption of proposition independence.

We plan to apply our system in a layered robot control architecture to act as both the planner and sequencer. The result will be a system that can switch tasks with less programming, and can generate plans as dictated by resources.

References

Blum, A. L., and Furst, M. L., 1995, Fast Planning Through Planning Graph Analysis. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1636-1642.

Blum, A. L., and Langford, J. C., 1999, Probabilistic Planning in the Graphplan Framework. In *the 5th European Conference on Planning* (ECP'99).

Bonasso, R. P., and Kortenkamp, D., 1996, Using a layered control architecture to alleviate planning with incomplete information. In *Proceedings of the AAAI Spring Symposium*, "Planning with Incomplete Information for Robot Problems". 1-4.

Boutillier, C., Brafman, R. I., and Geib, C., 1997, Prioritized Goal Decomposition of Markov Decision Processes: Toward a Synthesis of Classical and Decision Theoretic Planning. In *Proceeding of the Fifteenth International Joint Conference on Artificial Intelligence.*

Draper, D., Hanks, S., and Weld, D., 1994, Probabilistic Planning with Information Gathering and Contingent Execution, *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems.*

Koehler, J., Nebel, B., Hoffmann, J., and Dimopoulos, Y., 1997, Extending Planning Graphs to an ADL Subset, *Technical Report No. 99*, Institute for Computer Science Albert Ludwigs University.

Majercik, S. M., and Littman, M. L., 1998, MAXPLAN: A New Approach to Probabilistic Planning. In *Proceedings of the Fourth International Conference on Artificial Intelligence and Planning Systems*, 86-93.

Smith, D. E., and Weld, D. S., 1998, Conformant Graphplan, *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 889-896.

Weld, D. S., Anderson, C. R., and Smith, D. E., 1998, Extending Graphplan to Handle Uncertainty and Sensing Actions, *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 896-904.