Continual On-line Planning as Decision-Theoretic Incremental Heuristic Search

Seth Lemons[1] and **J. Benton**[2] and **Wheeler Ruml**[1] and **Minh Do**[3] and **Sungwook Yoon**[3]

[1]Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA
seth.lemons at cs.unh.edu
ruml at cs.unh.edu

[2]Dept. of Computer Science and Eng.
Arizona State University
Tempe, AZ 85287 USA
j.benton at asu.edu

[3]Embedded Reasoning Area
Palo Alto Research Center
Palo Alto, CA 94304 USA
minhdo at parc.com
sungwook.yoon at parc.com

Abstract

This paper presents an approach to integrating planning and execution in time-sensitive environments. We present a simple setting in which to consider the issue, that we call continual on-line planning. New goals arrive stochastically during execution, the agent issues actions for execution one at a time, and the environment is otherwise deterministic. We take the objective to be a form of time-dependent partial satisfaction planning reminiscent of discounted MDPs: goals offer reward that decays over time, actions incur fixed costs, and the agent attempts to maximize net utility. We argue that this setting highlights the central challenge of time-aware planning while excluding the complexity of non-deterministic actions. Our approach to this problem is based on real-time heuristic search. We view the two central issues as the decision of which partial plans to elaborate during search and the decision of when to issue an action for execution. We propose an extension of Russell and Wefald's decision-theoretic A* algorithm that can cope with our inadmissible heuristic. Our algorithm, DTOCS, handles the complexities of the online setting by balancing deliberative planning and real-time response.

Introduction

The goal of planning is to synthesize a set of actions that, when executed, will achieve the user's goals. Most academic research on general-purpose planning has concentrated on off-line planning, in which plan synthesis is considered separately from plan execution. This separation was originally motivated by the fact that even simplified off-line settings, such as sequential non-temporal planning, are intractable in the general case and it has helped focus research in the field on core algorithmic issues. In the last ten years, tremendous advances have been made in domain-independent plan synthesis and in many domains we now can find parallel temporal plans with hundreds of actions in a few seconds.

However, currently deployed planners for real-world applications are frequently run in an on-line setting in which plan synthesis and execution run concurrently. Such domains include manufacturing process control, supply chain management, power distribution network configuration, transportation logistics, mobile robotics, and spacecraft control. To take two prominent examples, in the CASPER system developed at JPL for the EO-1 satellite, observations of the Earth are analyzed on-board and can trigger recognition of important activity that immediately spawns requests for additional images (Gratch and Chien 1996). And in the Man-U-Plan system developed at PARC for parallel printing systems, new requests for printed sheets arrive in the system at a rate of several per second interleaving with the printing of other sheets (Ruml, Do, and Fromherz 2005).

Despite the importance of on-line planning domains, the issues they raise related to wall-clock time and plan execution have not been thoroughly explored. Indeed, during ICAPS community meetings and Festivus events, many prominent planning researchers have pointed out the weak connection of academic work to industrial applications as the most pressing issue facing the community today. One reason for this disconnection is that real-world applications tend to be complex. Another is that evaluating an on-line planner is more complex than running a planner off-line. In this paper, we first address these obstacles by proposing a simple formulation, called *continual on-line planning*, that extends current standard off-line planning into the online setting that takes into account wall-clock time. We propose a simple and crisp setting to study and evaluate alternative approaches to the problem of time-aware planning in on-line applications such as those outlined above. To solve problems in this setting, we present a real-time heuristic search approach that extends the decision-theoretic A^* algorithm (DTA*) (Russell and Wefald 1988). The new search algorithm is guided by a variation of the temporal planning heuristic and is used in a temporal forward decision-epoch based planner. This extends our previous work on building an online continual planning testbed and adapting popular offline planning algorithms to the online scenario (Benton, Do, and Ruml 2007). Our approach also utilizes our recent work on anticipatory planning algorithms for probabilistic planning (Hubbe et al. 2008) to anticipate uncertain online goals.

The paper is organized as follows: we start by formalizing the continual online planning problem. Next, we describe our search-based approach, Decision Theoretic On-line Continual Search (or DTOCS[1]) that extends DTA* to

[1]Pronounced "detox."

online planning, concentrating on key issues such as how to manage search time and when to issue an action to execute. We continue with discussion on a new temporal planning graph based heuristic adapting to the online problem constraints and objective function. We finish the paper with discussion on potential future work.

Continual On-line Planning

We propose a simple setting we call *continual on-line planning*. We assume that goals arrive over time according to some probability distribution that can be usefully modeled. Each goal has an associated reward and rate at which that reward decreases as time passes, starting at the moment when the goal is revealed. Each action the agent can issue has a known fixed cost. A continual on-line planning problem is defined by the tuple $\langle F, S, A, G, C, R \rangle$ where S is a set of possible states, each state consists of a set of facts which are a subset of F, and A is the set of actions that can be taken in the problem. G is a function for generating goals over time, and C is a cost function mapping actions to real values. For each $g \in G$, $a(g)$ is the arrival time of g, $v(g)$ is the initial reward value, and $d(g)$is a linear degradation factor for the reward given by achieving g. Each goal is defined by the usual partial state definition to be achieved. $R(g,t)$ is a reward function, giving a real value for goal g at time t. For a set of goals $\mathcal{G}$ and time t, we compute the reward obtained by achieving g as

$$R(g,t) = v(g) - (t - a(g)) * d(g)$$

The agent collects a goal's current reward at the first time that the goal's partial state specification holds between the arrival time of the goal and the time that the goal's reward reaches zero. The world state can be immediately changed—there is no requirement in our current formulation that the goal condition persist for a particular interval. The objective of the agent is to maximize its total *net utility* over its entire execution time, the sum of the goal rewards collected minus the sum of the costs of the actions executed. In other words, we want to maximize

$$O(T, \mathcal{G}) = \sum_{t=1}^{\infty} \sum_{g \in \mathcal{G}_t} R(g,t) - C(\alpha_t)$$

where T is a set of time points defined by action beginnings, ends, and goal generations. $\mathcal{G}$ is the set of all generated goals, $\mathcal{G}_t$ is the subset which have been achieved at time t but not achieved before t, and α_t is the set of actions begun at time t. $C(\alpha_t)$ returns the summed cost of any actions started at time t or 0 if none were started at that time.

While the new goal set changes the objective function over the agent's possible action sequences, we assume that the arrival of a goal does not affect the current state in any way. For example, it cannot affect the applicability of actions. Actions in our setting are deterministic, as opposed to settings in which executed actions may fail to achieve some or all of their post-conditions or cause additional effects beyond what was expected. To keep the problem simple, we assume that, apart from the stochastic arrival of goals and the actions issued by the agent, the world is deterministic. This means that there is a single agent and that the state is fully known at all points in time.

While one could simply synthesize a complete plan for the current set of goals, replanning whenever new goals arrive (Benton, Do, and Ruml 2007; Gratch and Chien 1996; Ruml, Do, and Fromherz 2005), the complexity of complete plan synthesis can delay the start of action execution. Because goal rewards decay as time passes, this strategy can result in lower net utility than a time-aware search strategy that interleaves further planning with the issuing of actions for execution. Interleaving planning with execution provides the possibility of achieving progress toward goals while planning for the future. It is important to note that time-aware search is different than anytime search which maintains an improving stream of complete incumbent solutions. While algorithms for this setting might find solutions and improve them, the focus of time-aware search is to be able to adapt to the arrival of new goals, rather than maintaining solutions for a static set of goals.

Furthermore, because we assume that the goal arrival distribution is known (or estimated on-line), we want to be able to plan for the future, acting in a way that anticipates any likely future goal arrivals. Because goal arrival does not change action applicability, there is no danger that we will issue an action whose execution depends on a goal that we expect to arrive in the future. If the goal fails to arrive, we simply receive less reward than expected, decreasing net utility.

As time passes, degradation in goal reward may lead to its reward no longer being greater than the cost of achieving it. Given that the objective in this setting is to maximize net utility, it may be detrimental to try to achieve all possible goals. Because of this, the setting can be seen as an extension of a partial satisfaction net utility problem (PSP net utility.)

Previous work in online planning has attempted to optimize average goal completion time and action costs up to some testing horizon, sometimes paying attention only to execution time for plans generated (Benton, Do, and Ruml 2007; Hubbe et al. 2008). While one might assume that planning time is negligible, if it must be done repeatedly (either at every execution step or whenever a new goal arrives) or is done in a large enough state space, the time spent planning may be just as important as the time spent executing the plan. A truly balanced algorithm would minimize cost and planning time, while maximizing reward for goals achieved.

A further complication relates to our dynamically changing reward structure. Approaches for handling dynamic cost changes (e.g., changing environments) have been proposed. For example, the D* Lite algorithm is made to re-use past search effort and recalculate heuristic values when an agent discovers a change in the environment (Koenig and Likhachev 2002). However, we have no such changes on rewards. Instead, we have a constant, measurable change dependent on time. For this reason we can give the exact reward (given by the current goals) at any time point and plan from there. Other environments have dealt with limited time for issuing actions (Korf 1990), but have used a static

```
1. while the utility of search $> 0$
2. $action \leftarrow$ select action to search
3. repeat $k$ times
4.     if a new goal has arrived, then
5.         clear the search tree and go to line 1
6.     if real time has reached the time stamp of the root, then
7.         prune everything except the subtree under the
               advance time top-level action
8.         go to line 1
9.     expand the best node under action
```

Figure 1: Pseudocode for DTOCS

time limit to decide when to issue each action, rather than adapting planning time to maximize reward. The Decision-Theoretic A* algorithm (DTA*) (Russell and Wefald 1988) makes some effort to balance search time with plan quality in a real-time setting, but did not handle decreasing goal reward, newly arriving goals, or partial satisfaction.

Our Approach

In this setting, we face a number of challenges that are not present in traditional planning. Because planning time influences the received reward, it may be advantageous to begin issuing actions before settling on a complete plan. In our setting, we are not strictly required to reach any particular goals, thus any action sequence is a legal plan and we do not risk incompleteness from issuing actions incrementally. Given the fact that the same plan executed at a different time gives different reward and our knowledge of the future is uncertain, it is also difficult to characterize optimal plans.

Given that goals arrive as we plan and execute, our algorithm, called Decision Theoretic On-line Continual Search (DTOCS), must use some method to predict and plan ahead for likely future goals. We have to decide somehow when to issue actions that we think are the best. As mentioned above, our objective is to maximize reward, minimize cost, and minimize actual time spent for both planning and executing to avoid goal degradation. Given our novel objective function, we must establish an appropriate expansion order for our search. Finally, we must handle the passage of real time as we search, handling additional goals, changes in which actions are executing, and degradation of goal reward. We will discuss each of these issues in detail below, and pseudocode is given in Figure 1.

Incremental Heuristic Search

We take inspiration from real-time search algorithms (Korf 1990) and simulation-based MDP solvers (Yoon et al. 2008). Unlike real-time search, we do not need a hard pre-specified limit on the amount of search performed before an action is issued. We use the term *incremental heuristic search* to differentiate this setting. Unlike hindsight optimization approaches, we need to carefully conserve computational effort when evaluating possible actions. We follow the approach taken by Russell and Wefald with DTA*, in which the planning agent periodically deliberates about whether further search is likely to cause a better action to be selected. In fact, we must be able to anticipate that the action selected after search will be sufficiently superior to the action that currently appears best that the delay caused by the search will be outweighed. Unfortunately, DTA* cannot be used directly because it assumes an admissible heuristic and assumes that action execution is instantaneous.

Planning is done continuously, updating and pruning the open list as time passes and actions are executed. The open list is cleared, however, when new goals arrive to allow more flexibility. The algorithm issues single actions (or "top-level" actions) intermittently, pruning nodes under all others but reusing nodes under the issued action by advancing the plans past that action. Confidence bounds are maintained on the top-level actions, representing the range within which the actual value the best plan underneath a top-level action is likely to take. We know that our heuristic has some amount of error and cannot be certain of any values for paths we have not fully explored. The DTA* framework does not directly apply, though, because it assumes a strict decrease in optimism along a path and the heuristic for the PSP net utility setting upon which our proposed heuristic is based is not optimistic. Because a top-level action's value can either increase or decrease, DTA*'s assumption that we need only search under the currently-best top-level action no longer applies.

Choosing when to issue actions is handled in DTOCS by determining whether the possible benefit of further search outweighs the loss in goal reward due to delaying action. Assume we have an action α which we currently believe is the best action, and an action β which we are not completely certain is worse. More search may reveal that β is, in fact, the best and α is not. However, the time spent searching will lead to decay in goal reward. We must issue actions when more search would likely cost more in lost goal reward than it would offer in plan improvement.

We attempt to expand nodes according to the degree to which they are likely to reduce our uncertainty about which top-level action is best. If goal reward did not decay, we could always search until we were confident that no action was better than α. However, we must instead prioritize nodes based on the probability that they will increase our certainty about the value of α or any other action with confidence bounds which overlap α's.

Handling the passage of time can be handled by the search using one of two methods: calculating heuristic values for some future point and re-calculating only when that point is reached, or generating heuristic profiles which are time aware. The first solution requires choosing the appropriate set of future points (or "happening points") for which to calculate heuristic values, and could lead to large amounts of overhead in calculating new values at these intervals. The heuristic profile solution gives a heuristic which is actually a set of values, each pertaining to particular times. It is more computationally expensive to compute up front but saves recalculation costs.

Anticipating Goal Arrival

To predict goal arrival in a computationally tractable fashion, we generate sample futures with anticipated goal arrivals in each. As in previous work on anticipatory planning and hindsight optimization, our algorithm samples possible futures using a goal arrival distribution and evaluates states according to the samples taken. This distribution could be provided as a part of the domain or learned from experience. These sampled futures look ahead into the future by a fixed horizon. Goals are added to a sample by using the goal generation function. These samples allow us to deal with concrete futures, rather than simply a goal arrival probability.

Yoon et al. (2008) evaluated single actions by calculating Q values for each action available from the current state according to each sampled future. DTOCS uses information from all sampled states to evaluate plans extending further into the state space than a single action. We generate plans that attempt not only to meet the goals of the current state but also goals from the sampled futures. When evaluating a state, we can easily calculate the heuristic for each sampled future by simulating execution of that plan in the sampled future. When we average these values, greater attention will automatically be paid to currently known goals that have already arrived because they will be present in all samples.

We will, however, have to modify or replace samples as time passes. First, our samples become invalid when their horizon becomes too close at hand to be useful for prediction and planning. We can avoid this happening at every time step by creating them with a longer horizon than we require and only updating them when they reach the actual desired horizon value. Likewise, samples will be less useful in evaluating plans which extend further into the future than the futures' current horizon. The occurrence of some events will also invalidate our samples, such as goals arriving which were not anticipated or anticipated goals not arriving. While replacing invalid samples is an obvious solution to this problem, it could cause drastic changes in heuristic values and lead to our open list needing to be completely re-ordered. There may be some benefit to extending, rather than replacing samples, because this may lead to less extreme change in heuristic values. In many cases, though, extending samples could lead to differences in heuristic values just as large as replacing or may lead us to depend too heavily on a limited sample set which may not be representative of the real goal arrival distribution and should, in fact, be replaced.

Search Space Structure

The state space is modeled after the one used by the Sapa planner (Benton, Do, and Kambhampati 2009). However, we split the open nodes between the top-level actions with which their plans begin. It is important to note that "advance time" is always an available top-level action, allowing the agent to wait for the next happening point to add new actions to the plan/execution. When advancing time, we move forward to the next happening point. These are traditionally the beginnings and ends of currently executing actions, but we face the complication that actions may not always be executing. Because of this, we include additional happening points for each goal, based on the predicted arrival time minus the makespan of the relaxed plan required to achieve it. We may, however, still not be acting at the right times, such as when goals arrive. We can add more happening points at regular intervals, but must take care not to make them too far apart or too close together.

Figure 2 provides an illustration of a Sapa-style search space where two actions, A and B, are applicable. Therefore, the available top-level actions are to advance time (not issue any action), issue action A (and immediately consider issuing an additional action or advancing time), or issue action B (thereby committing to not issuing action A, hence we must also advance time). After issuing action A, the current state would move to s_2 and we would consider either advancing time (and committing to not issuing B) or immediately issuing B to be executed along with A (and advancing time, because no more applicable actions are available to consider). In the figure, filled nodes are those at which time has advanced and hollow nodes represent intermediate decisions about additional actions to issue.

Confidence values for plans or top-level actions are based on an estimate of error in our heuristic. Much like the goal arrival distribution, this error measurement could be provided ahead of time or learned and could include features such as number of goals. As a plan is calculated further, more action costs are incurred and goals are achieved and we are, therefore, certain about the cost and reward for those goals, because goal sets in each sampled future are fixed. Confidence should, then, increase as search along a path progresses. The value of a top-level action and our confidence in it is based on the current plans (open nodes) under it. For an action α and set of nodes under it x with associated probability functions for f-value assignments, its value is calculated as

$$val(\alpha, x) = \mathop{\mathrm{E}}_{\vec{x} \in \mathbb{R}^n}[||\vec{x}||_\infty] = \int_{\vec{x} \in \mathbb{R}^n} (P(\vec{x}|x) * ||\vec{x}||_\infty) d\vec{x}$$

for $P(\vec{x}) = \prod P(x_i)$ where each $P(x_i)$ is our belief of the likelihood of the node at index i having the value x_i according to our current belief about that node's value and heuristic error. For the vector $\vec{x}$, $||\vec{x}||_\infty$ is the element-wise maximum, giving the highest value in the vector (i.e. the best node under a top-level action.)

We could simply use the value of the best plan under a top-level action, but the best may change to shorter plans (which we will likely be less confident in) as we discover that plans are not as good as the heuristic led us to believe. We would prefer to have a measure of uncertainty that constantly decreases with continued search, using information about some or all plans under the top-level action. Ideally, we would define $\vec{x}$ as the set of all nodes under α. However, we can reduce the size of computation by considering only a smaller set of nodes, either some constant number or selecting nodes within a certain range from the best node. One promising option is to pick nodes with expected values within the range of the confidence bound of the best node or the range we think its value is likely to change given a set amount of search.

Issuing Actions

The choice of when to issue an action is made by determining when the estimated benefit from further search would

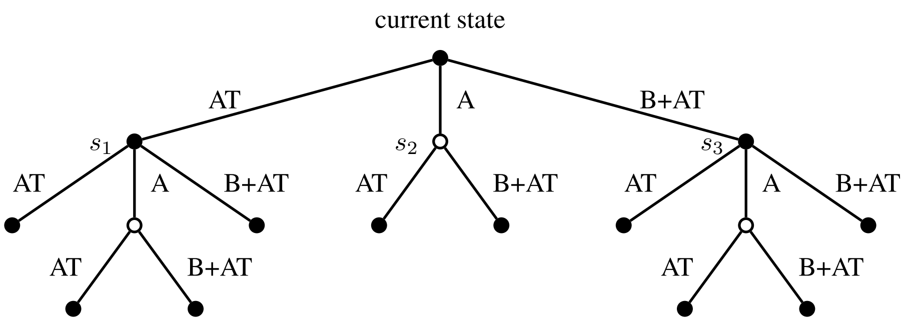

Figure 2: The structure of a Sapa-style incremental search space when two actions A and B are applicable.

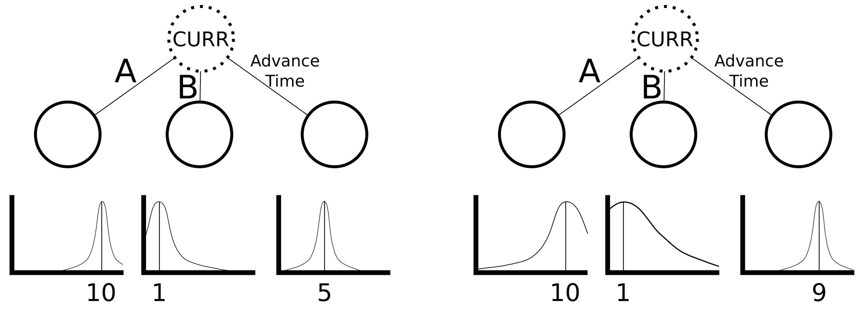

Figure 3: Left: The choice of action is clear. Right: Search may be justified.

outweigh the cost of doing more search. If search proves to be worthwhile, we continue expanding nodes. To reduce overhead, we assume some fixed 'quantum' of search (notated k on line 3 in Figure 1) between meta-level deliberation. occurs between evaluations of cost and benefit. More search along a path takes reward and cost that were previously estimates (due to the heuristic) and places them in the realm of certainty (because they have actually been experienced along the path.) Ultimately, we want to issue actions when we have explored far enough along paths to be sure that our estimate of the action's value is a good representation of reality.

In Figure 3 there are two examples of sets of actions under which some search has been performed, and further search may or may not be merited. On the left, action A has an expected net utility of 10 and the variance in the belief distribution is small, indicating that we are quite certain about that value. Action B has little overlap with A, and we are certain enough about the value of advancing time that it is unlikely that further search would reveal it to be better than A. Delaying execution for search would only cause goal reward degradation, so we choose to issue A. On the right in Figure 3, A still has the best expected value but we are less certain. We are also less certain about B's value, and advance time has substantial overlap with A. Depending on the rate of goal reward decay, further search may be beneficial to refine our beliefs about A and B. It is important to note that uncertainty about A would not be enough on its own to merit more search. Had we been uncertain about A but very certain that the values of all other actions were lower than A's, we would still be unlikely to change our preference for A after more search.

To compute the value of search, we need to estimate the effect of search and time spent searching on our beliefs about the values of top-level actions. We would like an estimate of how much we stand to gain if we discover a mistake in our belief about which action is best (benefit) and how much we stand to lose by possible wasted search effort (cost). Let α be the action currently judged best and β an action that may be better. We will use $v_t(\alpha)$ to represent the value of executing α at time t, taking into account how the goal rewards will decay. We can obtain the utility of search for line 1 of Figure 1 as

utility of search at time t =

$$\sum_{\beta \in actions, \beta \neq \alpha} \int_{y=0}^{\infty} p(\alpha = y)$$

$$\int_{z=0}^{y} p(\beta = z)(v_{t+\delta}(\alpha) - v_t(\alpha))dz +$$

$$\int_{z>y}^{\infty} p(\beta = z)(v_{t+\delta}(\beta) - v_t(\alpha))dz dy$$

The inner integral on the first line (over z from 0 to y) represents the loss of net utility that we experience if we search for time δ and still choose to execute α. Because $v(\alpha)$ is usually decreasing in time, this integral will usually be negative. The second line (for z greater than y) represents the situation where we discover that β is better than α and we execute it instead. This part of the equation could be either positive or negative, depending on how the delayed value of β compares to the original value of α. This formulation will likely be infeasible to evaluate directly, but will guide our development of an efficient approximation.

Focusing Search

Because our primary goal in the search is to become more certain about the values top-level actions, we want to direct our search to reduce uncertainty about them as quickly as possible. We may not want to search under the top-level action we currently think is best (α), because we may already be quite certain of its value, but not certain whether another action is better or worse because their confidence bounds overlap. Our strategy, then, is to focus search under the node with confidence bounds overlapping α (this includes α itself) and which our model leads us to believe will produce the greatest reduction in uncertainty if we were to search under it. We can either use a learned model for change in certainty given a set amount of search or use the confidence distribution to determine it. To use the confidence distribution, we can sample from that distribution and use the sampled probability that an action has the best value times the variance in values sampled. Using a learned model of change in uncertainty, the correct action under which to search could be determined as follows. If we assume a random, unknown variable I which represents the top-level action with the best value, we want to search in a way that reduces entropy in our knowledge about I. So, we select the action under which we will search as in line 2 of Figure 1 by

$$\underset{j}{\operatorname{argmin}}\, H(I \mid \text{beliefs under } x_j \text{ updated by search})$$

where $H(I)$ is defined as

$$\sum_i -(P(I=i) * log_2(P(I=i)))$$

The values for $P(I=i)$ will be calculated using our model to predict the change in node values under each action x_i assuming we were to search under it.

We could even select individual nodes from the global open list for expansion using similar criteria. This may be quite computationally intensive, however, for large open lists, so we choose to first focus search by top-level action and then select individual nodes under that.

Passage of Time

One issue that arises in our setting due to goal reward degradation is the need to re-evaluate the value of plans if too much time passes. A solution to this problem is to calculate values for the next happening point (traditionally when the next action in a plan begins or ends). In that case, we would plan as if that is when we would be issuing our next action. This approach has the advantage that it prevents reordering of nodes within the queues under each top level action. An alternative is to have a heuristic search method that can easily re-order search nodes. Unfortunately, this already expensive process would need to be over every top level action to reevaluate the their values, compounding the problem.

A Heuristic Evaluation Function

Online continual planning, while vastly different in many ways, maintains some similarities to temporal and partial satisfaction planning. This enables us to build on successful heuristic techniques from those areas. In particular, we modified temporal heuristic first defined in the partial satisfaction planner SapaPS (Benton, Do, and Kambhampati 2009) during our search.

SapaPS, like most modern satisficing temporal planners, is built for optimizing cost metrics that have a only cursory relationship to the makespan of the plan. Heuristics built around this idea tend to give an extremely optimistic view of the time point that a particular goal can be achieved. This poses some problems for planning in our scenario, where goal achievement reward declines as time increases. Despite this shortfall, we wish to leverage the practical gains these planners have shown. The challenge is how to augment state-of-the-art heuristic approaches such that they account for a time-dependent goal reward model.

The usual method for solving the relaxed problem in a temporal setting involves generating a temporal relaxed planning graph (TRPG). With this framework action costs can be propagated, allowing for local decisions on the least-cost supporting action for a given (sub)goal during relaxed plan extraction (Do and Kambhampati 2003). We follow this same methodology, additionally building a separate simple temporal problem (STP) to reason about goal achievement time. Specifically, the relaxed plan extraction process generates an STP that accounts for: (1) action durations; (2) action precedence and (3) static, binary mutual exclusions. While the original process naturally handles the first two points, addressing mutual exclusions is done only in the STP.

Finding Goal Achievement Time

A simple temporal problem can give estimates for goal achievement time, thereby providing a guess as to the potential reward that may be received at a particular state. We create an STP while extracting a relaxed plan, adding constraints inherent with the relaxed plan structure, as well as new constraints dealing with mutual exclusions between actions. Figure 4 shows the algorithm for relaxed plan extraction. The algorithm introduces additions to the relaxed plan extraction process found in the planner Sapa (Do and Kambhampati 2003), which first introduced the RTPG.

Our algorithm needs to distinguish between an action's start and end points to account for duration in the resulting STP. We use $a_{\vdash}$ and $a_{\dashv}$ to respectively denote the start and end points of a durative action..

The algorithm works as typical for relaxed plan extraction. For each goal, it finds a least-cost action supporter. At that time, a constraint is added to the STP that indicates the start and end points of the action must be separated by at least its duration (line 5). The supporter's conditions are then added to the goal queue (i.e., *OpenConditions*), and the next goal is processed. When a condition is added to the queue, it is associated with the action that achieved it. Then the supporting action is included into the relaxed plan and a constraint is added to the STP that includes the precedence

```
1. add all goals as initial set of open conditions
2. include dummy end action in relaxed plan (RP)
3. while there are no open conditions
4.     add least-cost achiever $a$ for next open condition to RP
5.     add constraint between $a_{\vdash}$ (start) and $a_{\dashv}$ (end) (duration length)
6.     add constraint between $a$ and RP action in chain (0 length)
7.     for all actions $b$ mutex with $a$
8.         add 0 length constraint between $a$ and $b$
9.     add all preconditions of $a$ to open conditions list
10. add dummy start action to RP
11. add constraint from $[0, \infty]$ between start and end
12. add 0 length constraints between start and actions at time 0
```

Figure 4: Relaxed plan extraction procedure with simultaneous construction of an STP

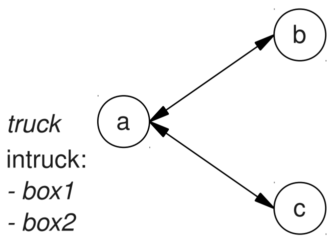

Figure 5: Problem set up for our example.

between supporting and supported action with respect to a particular subgoal (line 6).

These constraints are implicit in the structure of the relaxed plan, and so add very little with regard to estimates on goal reward. To perform meaningful action rescheduling, we add binary mutual exclusion constraints in the STP. Upon selecting an action a, we scan the current relaxed plan and add a constraint between a and any action a' that deletes the conditions of a (line 8). This allows us to gain better estimates on goal achievement time, thereby allowing our reward model to be evaluated.

Solving the STP Using the STP, we can gain an estimate on the reward for a particular goal. In other words, solving the STP gives us the time at which particular actions will begin and end (Dechter, Meiri, and Pearl 1991). The end point of actions that achieve a top level goal g will give us the earliest time point at which the heuristic believes g is achievable given the constraints in the STP. This allows us to find an estimate on the reward that we may achieve for the goal. For each goal g, an estimated achievement time point t, and an arrival time $\rho(g)$ we apply the following heuristic reward function:

$$R_h(g,t) = \begin{cases} v(g) & \text{if } t \leq \rho(g) \\ d(g) * t + v(g) & \text{if } \rho(g) < t \leq \frac{-v(g)}{d(g)} \\ 0 & \text{if } t > \frac{-v(g)}{d(g)} \end{cases}$$

This reward function differs from that of the original problem in that we explicitly assume that goals of a given future have already arrived (even if their scheduled arrival time, according to the future distribution, would be later). We do this so the heuristic can provide better estimates on possible reward, even when it is over-optimistic as to the goal achievement time.

Example: To see how the STP works (ignoring goal reward for the moment), consider a logistics problem where we must deliver two boxes (presently in the truck), *box1* and *box2*, to locations *b* and *c*, respectively (see Figure 5). Moving between locations takes 5 time units, while dropping a box at a location takes 1 time unit. A temporal relaxed plan for this problem would normally consider both goals achieved after 6 time units. That is, it would find a relaxed plan where the truck moves simultaneously from location *a* to location *b* and from location *a* to location *c*. It would similarly drop both boxes at the same time. Using the STP generated during relaxed plan extraction (as seen in Figure 6), we can re-calculate an estimate as to the makespan of the relaxed plan (and how long it takes to achieve each goal). In this case, solving the STP from the *start* action to the goal-achiving actions (i.e., *drop box1* and *drop box2*) gives us a time of 11 for both goals (and the entire plan). We can subsequently calculate reward for each goal given a reward function.

Selecting Goals

To select goals, we use a method similar to the one found in the planner SapaPS (Benton, Do, and Kambhampati 2009). The algorithm first calculates the new reward given by the time point at which a goal could be achieved according to the STN. Then, for each goal, it finds the difference between this value and the cost of reaching the goal (according to the cost propagation process). If this results in a value less than or equal to zero, the goal and all actions supporting only that goal are removed from the relaxed plan.

Possible Extensions and Conclusion

While our setting includes arrival of goals, this could be extended in two ways: goals whose arrival affects the problem state, and non-determinism in action results. In some domains it might make sense for arriving goals to affect the applicability of actions, but plans may not be applicable across multiple sampled futures. Likewise, uncertainty about the results of action execution could lead to plans not giving the reward we expected or even being possible in reality. Because we issue only single actions, this may be easier to handle in a naive manner, however, since we could assume ideal conditions in our search but only issue actions which are still applicable to the agent's real state. Both of these could be encompassed and further extended by asynchronous arrival of facts or sets of facts, which would, therefore, raise many of the same issues if it were used to extend our setting.

Given the conceptual work already done on DTOCS, one obvious next step is an implementation and comparison with previous work. An analysis of its performance would give us further insight into its strengths and weaknesses, and possibly indications of further room for optimization. One issue in implementation is which of the possible shortcuts and approximations to use for top-level action evaluation, calculating the worth of further search, and focusing search. The

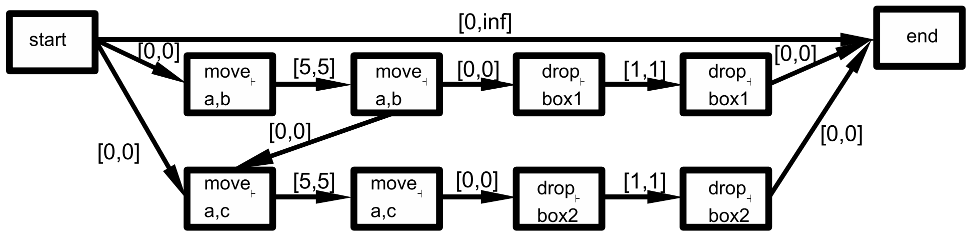

Figure 6: A graphical representation of the STP generated by the heuristic for our example.

correct balance of accuracy with efficiency will be vital to evaluation.

Additionally, future work should include exploration of the theoretical aspects of DTOCS. It would be useful to improve our understanding of the worst and best case performance, as well as the space required for the various calculations. Additionally, we should examine the space and time required to successfully learn the models required for our calculations of heuristic error and value change.

One extension to the algorithm would be the calculation of some heuristic value that represents more than a single point in time. We would prefer some sort of heuristic profile that gives us values for a state over some range of time. However, the issue of how to sort nodes with different values depending on the time is not trivial. Likewise, we would still have to re-calculate these profiles when time passes beyond the end of their current range.

Conclusion

Several practical planning problems involve on-line continual planning due to asynchronous goal arrival. However, most academic planners focus on fundamental, off-line scenarios and ignore the challenges associated with interleaving execution and planning. In particular, on-line continual planners must consider whether to continue to plan or to begin issuing actions while attempting to optimize wall-clock time and action cost. In our scenario we generalize this problem further to handle cases where goals have a decreasing reward relative to their arrival time and achievement time. This compounds the problem of balancing planning and execution time, because goals become irrelevant if their reward reaches zero. In these cases, failing to issue an action in a timely manner significantly impacts the quality of the resulting plan.

To handle these issues, we developed a search algorithm, DTOCS, based on DTA* The idea is to search under top-level actions for a given amount of time in order to reduce the uncertainty we have about their values. Additionally, we define a search *benefit* and *cost* under each top-level action such that we can decide which is the most promising in order to narrow down the action choice. An action is issued when the possible benefit of further search outweighs the loss of goal reward due to delaying action execution. During search, DTOCS applies a temporal heuristic that reschedules relaxed plans for estimating goal achievement time.

This search methodology offers a way of handling on-line continual planning problems where wall-clock goal achievement time is of paramount importance. Given that the structure of the framework is general enough to use any search under the top-level actions, the DTOCS algorithm will likely easily extend to handling scenarios that optimize over other quality measures dependent on wall-clock time. As such, we believe that the approach could potentially find applications in any area of robotics and artificial intelligence that involves such metrics.

References

Benton, J.; Do, M.; and Kambhampati, S. 2009. Anytime heuristic search for partial satisfaction planning. In *Artificial Intelligence*.

Benton, J.; Do, M. B.; and Ruml, W. 2007. A simple testbed for on-line planning. In *Proceedings of the ICAPS-07 Workshop on Moving Planning and Scheduling Systems into the Real World*.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–95.

Do, M. B., and Kambhampati, S. 2003. Sapa: a multi-objective metric temporal planer. *Journal of Artificial Intelligence Research* 20:155–194.

Gratch, J., and Chien, S. 1996. Adaptive problem-solving for large-scale scheduling problems: A case study. *Journal of Artificial Intelligence Research* 4:365–396.

Hubbe, A.; Ruml, W.; Yoon, S.; Benton, J.; and Do, M. B. 2008. On-line anticipatory planning. In *Proceedings of the ICAPS-08 Workshop on A Reality Check for Planning and Scheduling Under Uncertainty*.

Koenig, S., and Likhachev, M. 2002. D* lite. In *AAAI*, 476–483. American Association for Artificial Intelligence.

Korf, R. E. 1990. Real-time heuristic search. *Artificial Intelligence* 42:189–211.

Ruml, W.; Do, M. B.; and Fromherz, M. P. J. 2005. On-line planning and scheduling for high-speed manufacturing. In *Proceedings of ICAPS-05*, 30–39.

Russell, S., and Wefald, E. 1988. Multi-level decision-theoretic search. In *Proceedings of the AAAI Symposium on Computer Game-Playing*.

Yoon, S.; Fern, A.; Givan, R.; and Kambhampati, S. 2008. Probabilistic planning via determinization in hindsight. In *AAAI*.