A Simple Testbed for On-line Planning

J. Benton
Dept. of Computer Science and Engineering
Arizona State University
Tempe, AZ 85287 USA
j.benton@asu.edu

Minh B. Do and **Wheeler Ruml**
Embedded Reasoning Area
Palo Alto Research Center
Palo Alto, CA 94304 USA
{minhdo,ruml}@parc.com

Abstract

In contrast to most academic work on AI planning, many potential applications have an on-line character. For example, the true objective may not be to find the fastest plan, but to have solving plus plan execution end as soon as possible. Or additional goals may arrive while previous plans are still being executed. While specific planning systems have been built to address real-world problems in such settings, the fundamental algorithmic issues have often been obscured by the myriad complexities inherent in any deployed system. Basic questions regarding the strengths and weakness of different planning approaches remain unresolved in the on-line setting. To enable the systematic study of this important area, we introduce a general simulation testbed and demonstrate its power by highlighting the critical sensitivities of two popular planning approaches when they are adapted to an on-line setting.

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. Despite the importance of these on-line domains, the issues they raise related to wall clock time and plan execution have not been thoroughly explored. Indeed, during the ICAPS 2005 Festivus, many prominent planning researchers proposed 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 and messy. Another is that evaluating an on-line planner is more complex than running a planner off-line. In this paper, we address both these obstacles by introducing a domain-independent simulator that takes a PDDL-like description of a planning domain and then conducts an interaction with a planner in real time. This allows easy testing of an on-line planner and straightforward definition of simple domains to probe the sensitivities of different approaches. We describe a sample domain that captures key elements of a real industrial application, on-line manufacturing, and show how it provides insight into two popular planning approaches, state-space progression and partial-order planning, that we adapt to the on-line setting. Our preliminary results show that the online progression planner returns a higher number of better quality solutions in terms of wall-clock goal achievement time or the total duration (planning time + makespan) to achieve the goal.

On-line Planning

In the off-line setting, wall-clock time is not important and a planner can assume the world stops during the planning process. No distinction is drawn between the world state at the start of planning and at the start of execution. This assumption may be acceptable when the world is static during planning or the planning time is insignificant compared to the duration of actions in a plan. However, in many domains, those assumptions fail to hold. While an on-line setting raises many issues, the ones we will directly address in this paper are:

Optimizing wall clock end time: In many applications, the objective is to achieve the goal as quickly as possible, meaning that the figure of merit is the sum of the planner's computation time and the plan's execution time. Few existing planners optimize this metric. Rather, the field is divided between optimal and suboptimal planners, with no clear way to compare them. Imagine that on a particular problem an optimal planner takes 10 seconds to find a plan with a makespan of 2 seconds and a suboptimal planner takes 0.1s to find a plan with a makespan of 4s. The end time metric provides a clear way to compare these results. There is also an algorithmic aspect to this problem. For example, imagine that the same optimal planner on a different problem takes 10s to find a plan with makespan 100s, while the same suboptimal planner takes 0.1s to find a plan with makespan 200s. Because the suboptimal planner ignores the concept of wall time, it can perform worse than the optimal planner.

Deadlines: In addition to merely achieving a goal, it can sometimes be important to achieve it at or before a specified time. In a situation with multiple goals, the order the planner considers them in may be important. This leads to the question of bounds on computation time and intelligent use of time. That issue has already arisen in the International Planning Competition. For all planners that terminate within a specified time bound, the plan with the lower makespan wins, even if its competitors take only a fraction of the time. Yet few planning algorithms are cognizant of deadlines.

Asynchronous goals: In many real-world settings, goals are not neatly divided into separate episodes. Rather, new goals may arrive while plans for previous goals are still executing. A planner may need to replan or work around the existing commitments it has made. Relevant issues are: should we keep the previous plans or cancel them and replan from scratch? What if there are action retraction costs (Cushing and Kambhampati 2005)? One might view this as just a special case of exogenous events, but note that current work in planning does not address it. For example, the timed initial literals of PDDL2.2 are timed relative to the start of plan execution, not wall clock time.

Execution failure: There has been much work on planning under different forms of uncertainty, but even after two iterations of the probabilistic planning track of the international planning competition there remains controversy over evaluation methodology. On-line evaluation measuring wall clock time for achieving the goal provides a clear and useful way to compare approaches for handling execution uncertainty, putting policy pre-computation on level ground with replanning.

Duration uncertainty: In many applications, the exact duration of an action is not known until it executes. This issue has received some treatment by those involved in applications, but has not yet found its way into planner evaluation.

There have, of course, been many planning systems designed and deployed to handle variations of these issues in specific on-line applications. However, these systems invariably reflect the size and complexity of the real-world domains they address, making it difficult for academic researchers to study, compare, and modify the systems. For example, such systems often use formalisms quite different from PDDL, the standard among academic researchers. And sometimes the systems cannot be distributed. Furthermore, such individual point solutions do not give us an understanding of the design space, the underlying principles, or the inherent trade-offs between various approaches to the same on-line problem. We propose studying simple domains that stay close to PDDL and do not require complete reimplementation of existing off-line planners, but that incorporate carefully chosen features of the on-line setting. This will allow the emergence of common benchmarks on which multiple approaches can be studied and compared.

Related Work

There has been work on approximating on-line planning issues in the off-line setting. For example, Fox *et al.* (2006) discuss the problem of plan stability when the arrival of new goals necessitate the repairing of the previously found plan. While the setting is on-line in the sense that new goals arrive continually, wall-clock time is not considered and the plan generation and plan execution processes are still completely separate. As noted above, planning with timed initial literals does not suffice to capture existing commitments, both because the literals are not timed relative to wall time and they preclude the option of changing previous choices when warranted. Ruml and Do (2007) present preliminary work on optimizing for wall clock end time, although their planner has not been publicly distributed.

There is work on on-line stochastic scheduling problem (Bent and Hentenryck 2004; Bidot *et al.* 2007) in which scheduling and execution are interleaved. Scheduling is the closest field to planning. However, planning problems investigate selecting actions that make up the plan to achieve the goals and this is not tackled in scheduling. In our work, we want to create the on-line environment where selecting a consistent plan satisfying real-time constraints takes center stage.

The European Network of Excellence in Planning Research (PLANET) have established the Technical Coordination Unit (TCU) for On-Line Planning and Scheduling (Verfaillie 2002). In addition to raising awareness of important on-line applications, the TCU organized an On-Line Planning and Scheduling Workshop at the AIPS-2002 conference. There are many important on-line planning/scheduling work, mostly on real-world system, published in this workshop. However, none of them is in the setting that is easily accessible to off-line planners that take standard PDDL planning language. Our work tries to capture some of the most important on-line planning constraints and provide a simulation environment that do not require substantial changes to the off-line planners input language to handle the new problem.

desJardins *et al.* (1999) surveys "distributed continual planning" in which planning and execution are interleaved and can be distributed across multiple agents. We focus here on just the continual part, providing an environment allowing for easy migration for researchers of off-line planning.

There is extensive work on on-line planning for space applications at the NASA's Ames and JPL research centers (Chien *et al.* 2000), (Dorais and Gawdiak 2002), and (Pell *et al.* 1997); the planning approaches involve both plan repairing techniques and constraint-based planning. While discussing the mixed initiative on-line planner for NASA mobile robot, Dorais and Gawdiak (2002) were willing to provide the simulation environment that can help research in on-line planning. However, it's application and simulator are complicated and its input language differs from STRIPS planning and PDDL-like languages.

Pollack and Ringuette (1990) introduced Tileworld, a simulation testbed for research on planning and plan management in dynamic environments. However, critiques of Tileworld (Al-Badr and Hanks 2001; Hanks *et al.* 1993) have noted that its handling of time can be awkward, as the simulation is not real-time, and that the simulation and the planner must be integrated together in the same Lisp process with access to a shared data structure.

SimPlanner is perhaps the closest system to our work (Sapena and Onaindia 2003). It provides both a PDDL planner and an executor to interact with the planner. However, the simulation is not with respect to wall clock time, actions are not durative, and new plans cannot execute in parallel with previous commitments.

A Simulation Testbed

We have developed an on-line planning simulation environment to facilitate research on on-line planning. Figure 1 shows

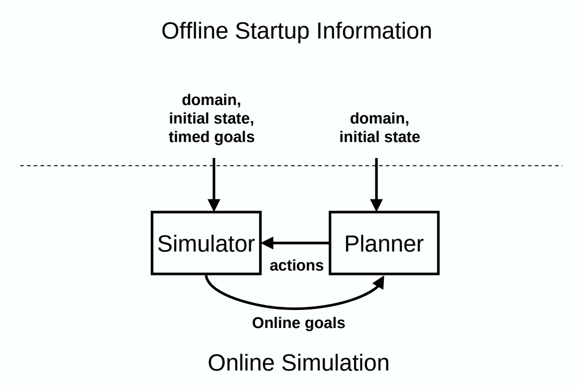


Figure 1: Our simulation architecture.

the simulation architecture in which the simulator and planner communicate in real-time. In the implementation, the planner is run as a child process of the simulator, communicating with it using plain text messages via standard I/O. This is very simple, as long as care is taken to flush output buffers. It is also portable across platforms, and language independent. As the simulator requires very little memory and CPU time, it is reasonable to run it on the same machine as the planner. (Using a stub planner that communicates through sockets to reach a real planner on a different machine is certainly a possibility.)

Simulator: Upon starting, the simulator takes as input: (1) a domain specification; (2) an initial state specification (problem file); (3) a timed goal specification file. Simple examples are discussed below. The domain and problem files are similar to the domain and problem files given to conventional off-line planners, following the same syntax as PDDL2.1 STRIPS-style durative actions (Fox and Long 2003). Each new goal g specified in the goal file has an associated time point t_g at which it is to be sent. Goals can also specify deadlines by which they must be achieved or utility functions that define their value based on when they are achieved. When they are sent to the planner, goals have a syntax similar to a PDDL2.1 problem file, as they specify new fluents, the initial values of the fluents and the goals to be achieved.

The simulation begins with the initial state specification. When a plan p is received from the planner, the simulator will simulate the execution of p and check for plan validity by detecting action precondition satisfaction and interferences with concurrent plans of other goals that are currently executing at the same wall-clock time. This is done by checking at the scheduled starting time of each action if any of its preconditions will not be satisfied due to previous action failure. If an action fails to execute, its effects will not be activated and can thus affect the executability of other actions scheduled later. The simulator notes the wall-clock time at which each goal is achieved in order to measure plan quality. The simulator stops executing at the wall-clock time when all goals are satisfied (or after a time-out if the planner has not provided any plan).

Although we do not exercise it in this paper, the simulator is also capable of simulating random action execution failure. In this case, it notifies the planner and removes all unexecuted actions affected by this failure from future execution. Command line options are provided for such tasks as random seed setting (to provide some degree of repeatability), and debugging output. Note that because the simulator and planner are running in real time, subject to OS scheduling and multiprogramming, there is no absolute guarantee of repeatability. We like to regard this as a benefit, in that one is encouraged to verify the statistical significance of any observed effect.

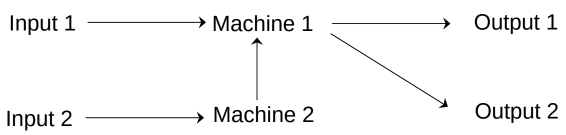


Figure 2: Example of a Manufacturing Plant.

On-Line planner: Upon receiving the new goal g at time t_g, a planner needs to produce a (complete or partial) plan P_g intended to satisfy g and send it back to the simulator. If the planner takes t_p to find P_g, and sends the plan to the simulator at time $t_s \geq t_g + t_p$, then the starting time t_a of each action $a \in P_g$ needs to satisfy $t_a > t_s$. Note that t_p is an elapsed time while all the other timing instances t_g, t_s, and t_a are wall-clock times.

This simulation testbed clearly addresses the issues of optimizing wall clock end time, goal deadlines, asynchronous goals, and execution failures. We intend to extend the simulator in the near future to handle execution duration uncertainty. Given that the inputted information provided to the planners are of the standard PDDL2.1 format that is used many academic off-line planners, we expect the framework will require minimal overhead when extending off-line planners to handle on-line planning constraints.

The Manufacturing Plant Domain

The simulator as described above can simulate any on-line planning domain given three input files: domain, initial problem, and goal written in the correct format. To test the framework and provide an example of a simple yet realistic on-line planning problem, we have written the domain file and the random problem and goal generators for the manufacturing plant domain described by Ruml *et al.* (2005).

The manufacturing plant domain consists of a set of machines linked together in a directed planar graph G (see Figure 2). Additionally, the domain includes a set of input and output devices. Input devices must have only out-links and output devices must have only in-links. Input and output devices must only link to machines on the convex hull of G. This is done to approximate the layout of the devices on a typical manufacturing floor. Material is routed from input devices to output devices. When material exists inside a machine, that machine may give some attribute to the material. Only one material may exist at a given time in most machines (with the only exception being that more than one may exist in input and output devices). New goals given to the planner consist of (1) the initial location of the material (an input device) (2) the final destination of the material (an output device) and (3) the set of attributes which the material must possess. A version of this domain was originally introduced in Ruml *et al.* (2005) and it shares some commonalities with scheduling problems (Pinedo 2002) where machines are discrete resources. In fact, it can be viewed as a combination of a planning and scheduling problem.

```
(define (domain manufacture)
 (:requirements :typing ...)
 (:types material attr machine - object)
 (:predicates (at ?mat - material
                  ?mach - machine)
   ....<snip>....
(:durative-action add_attribute
   :parameters (?mat - material
                ?mach - machine
                ?att - attr)
   :duration (= ?duration 2)
   :condition (and (over all (at ?mat ?mach))
                   (at start (gives ?mach ?att)))
   :effect (and (at end (has_attr ?mat ?att))))
```

Figure 3: Example domain file specifying an action.

```
(define (problem manufacturing-1)
(:domain manufacture)
(:objects input1 input2 machine1 machine2
          output1 output2 - machine)
(:init (is_output output1)
       (connected input1 machine1)
   ....<snip>....
(:goals (and))
```

Figure 4: Example problem file specifying initial state.

Figure 2 shows one example of the manufacturing plant with two inputs, two outputs, two machines and the connections between them. Machine 1 can give attribute 1 and machine 2 can give attribute 2 to any material routed through it. We will use this as a leading example throughout the rest of this paper. Figures 3, 4, and 5 shows the syntax for part of the domain file, the problem file specifying the initial state, and an example of the timed new goal. The domain and problem files are exactly like the normal domain and problem files given to STRIPS off-line planning, with exception that the goal set is empty. The timed new goal is similar to a (partial) problem file with new objects, their initial and desired goal condition. The time at which the goal need to be sent to the planner (e.g. 5.5 in Figure 5) is only known to the simulator, but not to the on-line planner.

On-Line Planning Algorithms

To illustrate the use of our testbed, we present two on-line planners derived by extending popular off-line planning algorithms: forward-state space (FSS) and partial order planning (POP). We also test different search algorithms and heuristics with those planning algorithms. The search algorithms and heuristics are geared toward either producing a plan fast (shorter planning time) or a shorter makespan (shorter plan execution time). As discussed above, those are the two components adding up to our objective function of minimizing wall-clock goal achievement time.

The input to both algorithms from the simulator are:

- The domain specification and initial world state I when the planner starts at time t_I.
- The set of goals G that arrive continuously at an unpredictable time t_G.

Recall that the abstract of the communication between the planner and the simulator is given in Figure 1; examples of the

```
(:goal 5.5
 (:newgoal (:domain manufacture)
 (:problem manufacturing-1)
 (:objects mat2 - material)
 (:init (at mat86 input2))
 (:goal (and (has_attribute mat2 attribute1)
             (has_attribute mat2 attribute2)
             (at mat86 output2)))))
```

Figure 5: An example timed goal for the simulator.

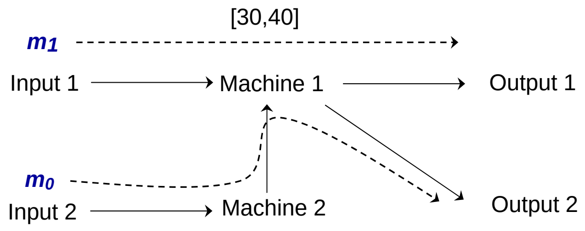


Figure 6: Manufacturing Plant with: new material m_1 and previous material m_0 for which route was planned.

domain, problem, and new goal files are given in Figure 3 and 5. The objective of the planner is to satisfy all goals in G by outputting a plan P_j that can finish executing as early as possible, according to the wall-clock time. Given that the simulator will not give the planner information about other actions from previous plans that were/are/will execute, the planner needs to keep track of how the world evolves according to its previously found plans[1]. In our implementation, the planner maintains a *global plan* $\mathcal{P}$, which is a union of all plans found. The world state at any given moment t can then be calculated by simulating $\mathcal{P}$. Within the planner, action interaction within $\mathcal{P}$ follows the semantics of TGP (Smith and Weld 1999): (1) all preconditions must hold during the execution of an action; (2) delete effects happen at the beginning and add effects happen at the end of action duration. The plan P found for each new goal g and the global plan $\mathcal{P}$ are in the same format used in the International Planning Competition, which is $P = \{(a, t_a)\}$ where a are grounded actions, and t_a are fixed starting time of a. The difference is that t_a is wall-clock time, not reference time.

We will also use the following notations for the rest of this section: P_S is the plan leading from the root search node to the state S (we will use "search node" and generated "plan state" interchangeably); t^s_a, t^e_a are the starting and ending time points of a given action a. We assume that the plans for previous goals (already committed in the execution engine) cannot be changed/retracted. At the moment, our planners cannot handle the execution failure feature of the simulator.

To illustrate the search steps in different on-line planning algorithm, we use the simple configuration of the manufacturing plant depicted in Figure 2. Figure 6 shows in dotted lines the route/plan for two units of material. Assume that moving a material from one location to another or putting an attribute on material both takes 10 time units. At time $t = 5$, the planner receives a new goal g which specifies the routing of material

[1]We plan to extend the simulator so that it accepts queries from the planner asking for current world state and future commitments. With this capability, the planner does not need to maintain the evolution of world state internally.

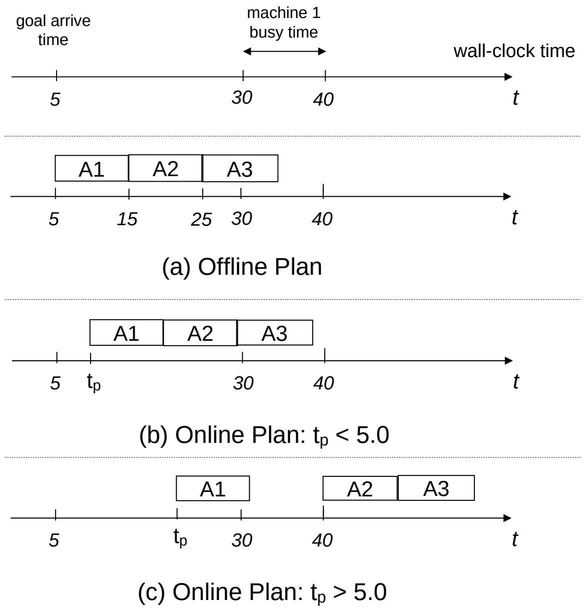


Figure 7: Plans by progression planners.

m_1, originally at *input 1* to *output 1* and m_1 needs to have attribute 1 (produced by *machine 1*). There is also a previous plan for another unit for material m_0 starting from *input 2* that occupies *machine 1* for the duration $[30, 40]$.

Forward State Space On-Line Planning

This is the approach used by many of the most effective off-line planners, such as FF (Hoffmann and Nebel 2001) and Fast Downward (Helmert 2006). Here the planner progresses from the initial state, and moves forward toward the goals, at each step, one action is added to the current state. An added action can be any grounded action applicable in the current state, or a special "advance time" action to move forward to the next relevant time point. The starting time t_a^s of all actions in the partial/final plan are all fixed. In classical planning, where all actions have unit duration, then the state is advanced one "step" (unit duration) at a time. In temporal planning, where actions may have different durations and can interleave arbitrarily, then the state advances forward during search by moving from the current time to the next "decision epoch". Each "decision epoch" is either the start or end time point of an action in the partial plan leading to the current state[2]. In our ongoing example, the initial set of decision epochs are $t = 5$, $t = 30$ and $t = 40$.

In our leading example depicted in Figure 6, an off-line planner will find a plan $P_1 = \{(5.0 : A_1), (15.0 : A_2), (25.0 : A_3)\}$ (i.e. A_1 start at 5.0) with $A_1 = Move(Input_1, Machine_1)$, $A_2 = AddAttr(Machine_1, m_1)$ and $A_3 = Move(Machine_1, Output_1)$. However, given that any planner will take some duration $t_p > 0$ to search for a plan, any action in the found plan P can only start as early as

[2]Theoretically, any time point in the future can be a relevant time point to "advance" to. However, to bound the search space, forward state-space planner only advance time to the next decision epoch.

Algorithm 1: Forward state-space on-line planning

```
1  Wall-clock time: $t_c$;
2  Estimated planning time: $\tau_p$;
3  World state at time $t_c + \tau_p$: $I$ ;
4  State Queue: $SQ = \{I\}$;
5  while $SQ \neq \emptyset$ do
6      S:= Dequeue(SQ) ;
7      if $\exists a \in P_S : t_a \geq t_c$ then
8          Return to Line 5;
9      end
10     if S satisfies G then
11         Output $P_S$;
12     else
13         Expand $S$;
14     end
15 end
```

$5.0 + t_p$. Thus, P_1 may not be a valid plan in the on-line setting.

Algorithm 1 shows the on-line version of the forward state-space (FSS) planning algorithm. To find a plan for a set of new goals g, a FSS planner needs an initial state I_g to progress from. Ideally, I_g should be the initial state at time $t = t_c + t_p$ (with t_c is the wall-clock time at which the planner first knows about g and t_p is the time taken to find the plan). However, because t_p is unknown, the on-line FSS planner needs to approximate it using the estimated planning time τ_p (Line 2) and thus the new initial state to progress from is the world state at time $t_1 = t_c + \tau_p$. If the planner finds a valid plan P within $t_p < \tau_p$, P is executable at t_1. Given this world state, the planner searches like an off-line planner. However, whenever a state S is selected for expansion, S will be discarded if there is an action in P_S that starts after the wall-clock time t_c at which the expansion occurs (Line 7). Note that algorithm 1 may find a solution even if the actual planning time t_p is larger than the estimated time τ_p. For example, a plan with the first action A_1 starts at the decision epoch $t = 30$. This can be found with $\tau_p = 5$ and the actual planning time $t_p = 20$. This is because the expand function (Line 13) can generate a child state just by advancing time to the next decision epoch. $t = 30$ is the next one in the root state. Figure 7 shows the plans found by the off-line and on-line FSS planning algorithms when $\tau_p \leq 5.0$ and when $\tau_p > 5.0$.

After the new initial state at $t + \tau_p$ is set, the planner basically uses the off-line version to find the final plan, except the checking of action starting time with the wall-clock, and thus it can employ all the techniques that make the FSS off-line version a fast algorithm and is likely to be able to find a plan quickly. The disadvantage of this approach is having to approximate the planning time t_p with the τ_p value. We are not aware of any work on predicting planner running time for arbitrary problems. If we underestimate t_p (i.e. $\tau_p < t_p$), we may need to restart the search. If $\tau_p > t_p$, the plan found may have to wait for a period $\tau_p - t_p$ before it can start executing. To partially overcome this "waiting" period, when the plan is found (Line 11) we post-process the plan before sending it to the simulator. The purpose of this post-processing phase is to try to shift the starting time of actions in P as close as possible to the wall-clock time at which P is found. In our implementation, we use the greedy post-processing approach implemented in the *SAPA* planner (Do and Kambhampati 2003) by: (1) first, greedily building the causal structure of actions in P and $\mathcal{P}$ (build a causal-link plan out of the fixed-time plan); (2) representing the causal relations by temporal constraints in a Simple Temporal Network (STN); (3) use the STN to find the earliest starting time for all actions with constraints so that all actions start after the current wall-clock time.

For off-line planning, different search algorithms can trade-off between plan quality (e.g. makespan) and planning time. Given that planning time contributes to the objective function of minimizing goal achievement time in on-line planning, we have also used different search algorithms, A^* and weighted A^*, to control search in the planner. With the objective function of minimizing plan makespan, then A^* is likely to take longer time to finish, but find plan with shorter execution time (makespan) than weighted A^*. We note that, unlike in an off-line setting where a complete search algorithm will find a plan if one exists given an infinite amount of memory and there is no bound on running time, this is not true for on-line planning. Because of goal deadlines and previously committed actions, a true guarantee in online planning may require an infinitely fast planner.

For the same purpose of investigating the tradeoff between different options that can lead to either: (1) shorter planning time; or (2) shorter plan execution time (makespan), besides using different search algorithms, we have also implemented two different heuristics to guide the planner's search. The first heuristic is the planning-steps measure heuristic, which normally leads to shorter planning time t_p, and the second one is a makespan-measure heuristic, which likely leads to a shorter makespan plan, but takes a longer time to solve. Both heuristics are derived by first solving the shortest-path problem using the connections in the manufacturing plant. Thus, at start-up time, the planner calculates the number of steps or makespan to reach each output node from any input node by solving the single-source all-destination shortest path problem using dynamic programming for all input nodes. This heuristic relaxes the real-time planning problem by ignoring the interactions with previous plans (thus, assuming that the manufacturing plant is always empty).

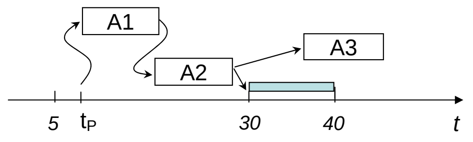
The advantages of the FSS planner are: (1) the state representation is simple; and (2) interactions between newly added actions and previously committed action are minimal. However, the FSS algorithm shown above needs to estimate the planning time t_p to set the initial fixed-time state to progress from and to estimate t_p correctly which is not an easy task.

Partial Order Planning

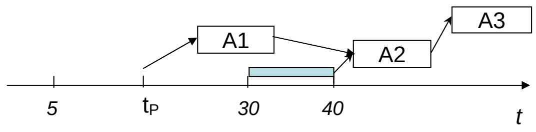
In temporal planning, partial order planners (POP) are a good alternative due to their flexible framework that is particularly versatile for temporal constraints. Given that on-line planning adds real-time temporal constraints to off-line planning, we believe that POP is a great option and have implemented a POP on-line planning algorithm. Algorithm 2 shows the on-line-POP algorithm. Like the off-line version, each search node $S = (C, A, L, O)$ consists of *C: open conditions*, *A: actions*, *L: causal links*, *O: partial orders*; the planner picks an open condition, selects the supporting action, adds the appropriate causal link and resolves any threat caused by adding temporal ordering constraints between newly added actions and the interfering actions in the current partial plan. Weld (1994) provides an excellent tutorial on the POP algorithm.

Algorithm 2: On-Line Partial Order Planning Algorithm

```
1  Wall-clock time: $t_c$;
2  Initial search node: $S_0 = (\langle G \rangle, \emptyset, \emptyset, \emptyset)$;
3  Search Queue: $SQ = \{S_0\}$;
4  while $SQ \neq \emptyset$ do
5      $S = (C, A, L, O) = Dequeue(SQ)$ ;
6      $\forall a \in A$ : add temporal constraint $t_c < t_a^s$;
7      if Temporal Inconsistent then
8          Return to Line 4;
9      end
10     if $C = \emptyset$ then
11         Output $P_S$;
12     else
13         $p \leftarrow$ remove an open condition from $C$;
14         Select actions $a$ supporting $p$;
15         $Enqueue(Apply(a, S), SQ)$;
16     end
17 end
```



(a) Online POP plan: tP < 5.0



(a) Online POP plan: tP > 5.0

Figure 8: Plan found by an on-line POP planner.

The main difference between the off-line and on-line versions are in Line 6-8 where whenever a search state is picked from the queue, we add temporal constraints to ensure that all actions in the partial plan found so far can start executing after the current wall-clock time. If there is a temporal inconsistency, then the selected node is discarded and the next best one is picked from the search queue for expansion. Figure 8 shows the plan found by Algorithm 2 (for two cases where the planner takes $t_p < 5$ and $t_p > 5$ to find this plan). In our implementation, all temporal constraints are managed by a complete Simple Temporal Network (STN) implementation. The same search algorithms and heuristic evaluation functions are used as in the state-space planner discussed above (i.e. A^* or weighted A^* algorithms with makespan or step measure heuristics).

For STRIPS off-line planning, FSS planners have been generally faster than POP-based planners. However, compared to the on-line FSS algorithm shown in Algorithm 1, the POP planner is more flexible and does not need to predict

	FSS				POP			
	Makespan		Steps		Makespan		Steps	
	A^*	wA^*	A^*	wA^*	A^*	wA^*	A^*	wA^*
Goal-achievement Time	164	8	22	117	20	7	6	3
Goal-achievement Duration	74	22	5	108	43	28	10	51

Table 1: Comparing different planning algorithms in terms of number of goals solved with the best solution using the metric: (1) wall-clock goal achievement time; and (2) duration from the time when the planner start searching for the plan and the time when the goal can be achieved. Ties are not counted.

and approximate the planning time t_p using the τ_p value. If both planners take the same amount of planning time t_p and find the same plan, then the plan returned by the POP planner will likely be able to start earlier and finish earlier because it suffers no waiting time or post-processing.

Besides FSS and POP algorithms, there are other frequently used off-line algorithms such as regression planner, local search or compilation. We do not feel simple adaptation of any of them would give a definite advantage over the FSS or POP algorithms. A regression planner needs an approximate time for when the plan will finish execution (to regress from), this involves approximating both the planning time t_p and the plan makespan. The compilation approach (e.g., compiling the problem into a linear program) has not shown to work well for off-line temporal planning and thus would scale poorly in both off-line planning and real time on-line planning. Cushing *et al.* (2007) discuss an algorithm that combines FSS and POP, which can potentially combine the speed of a FSS planner with the flexibility of a POP planner. However, the algorithm was not implemented and its relative performance is unknown.

Empirical Evaluation

We have implemented the online plan simulator discussed in this paper in Objective Caml.[3] The simulator can: (1) continuously output new goals to the planner; (2) receive plans and validate them; (3) simulate execution failures. The random problem and goal generator for the manufacturing domains are written in Java. When generating problems, the machine connections are produced through delaunay triangulation (Delaunay 1934). Each machine is randomly placed on a grid, then the coordinates are triangulated to create a planar graph between the machines. After this, the input and output nodes are placed on the convex hull of the graph. This is done to approximate the layout of a physical manufacturing plant. The goals are generated by randomly picking the start location, end location, the set of attributes that the material needs to carry at the end, and the time at which the goals need to be sent to the planner. The two online Progression and POP planners discussed in this paper are fully implemented in Objective Caml.

Upon starting both the simulator and the online planner (see Figure 1), they first exchange their basetimes to synchronize their internal clocks. Then the simulator will send out goals continuously to the planner through standard output and wait for the plans produced by the planner through standard input. Communication between the two modules are through the OCaml piping facility. Upon receiving the plans, the simulator will check for plan validity given the possible interactions between concurrently executing actions in the same plan, or between actions in plans for different goals. The simulator also simulates the plan execution and logs the goal achievement time. For the analysis purposes, the planner will also send to the simulator the wall-clock time at which it starts searching for a particular goal and this information is also logged. For the online progression planner, the τ_p value (see Algorithm 1) is initialized to 1 second and doubled every time the planner fails to find a plan. The planner will give up on a goal if search time is more than 100 seconds and move to the next goal that it has achieved (or continue to wait for the new goal to arrive). After sending out the last goal to the planner, the simulator will wait for 1000 seconds before it terminates. It is possible at that time, the planner has not finished planning for all goals. The online planner cannot handle execution failure messages from the simulator at the moment, so we disabled that feature from the simulator.

We ran the simulator with the two online planners on a Linux OS Intel Xeon 2.66GHz machine with 8GB of RAM, compiling the OCaml programs to optimized native code. We want to compare different plan settings (i.e., the planning algorithm, search algorithm, and heuristics) on the following two values: (1) the wall-clock time each goal is achieved; (2) the durations from the wall-clock time the planner start working on a goal to the wall-clock time that the goal is achieved. We created 10 random manufacturing plant configurations, each one is tested along with 20 randomly generated sets of online goals. Each set can contain anywhere from 2 to 10 goals.

Table 1 summarizes the results of the two online progression (FSS) and partial order (POP) planners with two different search algorithm: (1) A^* geared toward shorter solutions; and (2) weighted A^* geared toward faster solving time (we use $w = 3$ in the implementation). The two heuristics used: makespan-measure and steps measure are discussed in the earlier part of the paper. For a given goal g, let t_g be the wall-clock time at which the simulator starts sending g to the planner, and let $t_s > t_g$ be the time when the planner starts to work on g (due to previous goals, the planner may not be able to work on g instantaneously at t_g); upon receiving the plan returned by the planner and simulate its execution, the simulator found that g is achieved at $t_a > t_s$. We compare the performance of different planners using two quality measure: (1) the actual wall-clock goal achievement time t_a; (2) total time from t_s to t_a (thus: search time + makespan). Table 1 reports the number of goals for which each planner returns the *best* solution according to each quality measurement. The results indicate that in general, the progression planner finds better quality plans over the POP planner. Close investigation shows that the POP planner tends to take more time per node due to all the temporal reasoning inside the STN at each node.

[3]Executables are available at the URL `http://www.public.asu.edu/~vidar/op`.

The branching factor is also higher given the large number of possible partial orders not only within the current partial plan but also between actions belong to different plans. This can be quite time consuming when there are many concurrent plans. It is interesting to see that the best option for both planners are at the extreme case: (1) A^* search algorithm with makespan-measure heuristic; and (2) weighted A^* search algorithm with the step-measure heuristic. Among all options, the first one will likely return the shortest makespan solution while the second one will likely return the solution in the least amount of time. Online goal achievement time is the summation of those two factors. However, other combinations (e.g. A^* with the step-measure heuristic) that are not geared toward minimizing one of those two factors fall short of those two cases.

Conclusions & Future Work

In this paper, we introduced a simple testbed for aiding research in on-line planning. The testbed consists of a domain generator, a simulated on-line planning execution environment, and two on-line planners that work with the simulator. The system is fully implemented and can be used to investigate different on-line planning algorithms and strategies across different domains. It has already been chosen for use in the International Competition on Knowledge Engineering for Planning and Scheduling (ICKEPS 2007). The simulator communicates with the planners by exchanging messages in the standard PDDL planning language that should be easily parsed by the current state-of-the-art off-line planning systems. Therefore, we hope that it can help foster interest from the academic planning research community, which has hitherto concentrated on off-line planning, in the kind of on-line planning issues commonly found in applications. Given the ubiquity and importance of these problems, any advancement in generic on-line planning techniques could have great benefit in many applications.

While we want to concentrate on simulating the continual aspect of on-line planning, there are many things that can be done to extend our current work. For the simulator, we want to extend beyond temporal planning to the full PDDL3.0 specification that involves metric quantities, action costs and preferences. Another direction is to extend it to support multi-agent planning. This can be done by extending the communication channel from one to multiple online planners working in the shared simulation environment. For the planners, we want to investigate other planning algorithms, incorporating different search techniques (some of them directly take into account search time – planning time), and better search heuristics targeting different plan qualities that are relevant in the on-line planning setting such as real-time goal achievement time, stability, or commitment retraction cost. Right now, we believe that the heuristic used in our online planner is still rather weak because it does not fully address the interactions between previously found plans and the new plan in the online setting where wall-clock time keeps passing during the planning search.

References

Badr Al-Badr and Steve Hanks. Critiquing the tileworld: Agent architectures, planning benchmarks, and experimental methodology. In *http://citeseer.ist.psu.edu/al-badr91critiquing.html*, 2001.

Russell Bent and Pascal Van Hentenryck. Regrets only! online stochastic optimization under time constraints. In *Proc. of AAAI-04*, pages 501–506, 2004.

Julien Bidot, Thierry Vidal, Philippe Laborie, and Christopher Beck. A general framework for scheduling in a stochastic environment. In *Proc. of IJCAI-07*, 2007.

Steve A. Chien, Russell Knight, Andre Stechert, Rob Sherwood, and Gregg Rabideau. Using iterative repair to improve the responsiveness of planning and scheduling. In *Proc. of AIPS*, 2000.

Wiiliam Cushing and Subbarao Kambhampati. Replanning: A new perspective? In *ICAPS-05 Poster Program*, 2005.

Wiiliam Cushing, Subbarao Kambhampati, Mausam, and Daniel Weld. When is temporal planning really temporal? In *Proc. of IJCAI-07*, 2007.

B. Delaunay. Sur la sphere vide. In *Izvestia Akademii Nauk SSSR*, 1934.

Marie E. desJardins, Edmund H. Durfee, Charles L. Ortiz, Jr., and Michael J. Wolverton. A survey of research in distributed, continual planning. *AI Magazine*, 20(4):13–22, 1999.

Minh Do and Subbarao Kambhampati. Improving the temporal flexibility of position constrained metric temporal plans. In *Proc. of ICAPS-03*, 2003.

Gregory A. Dorais and Yuri Gawdiak. The challenge of planning and execution for spacecraft mobile robots. In *Proc. of On-line Planning and Scheduling Workshop, AIPS02*, pages 11–18, 2002.

Maria Fox and Derek Long. PDDL2.1: An extension of PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20, 2003.

Maria Fox, Alfonso Gerevini, Derek Long, and Ivan Serina. Plan stability: Replanning versus plan repair. In *Proc. of ICAPS-06*, pages 212–221, 2006.

Steve Hanks, Martha Pollack, and Paul Cohen. Benchmarks, testbeds, controlled experimentation, and the design of agent architectures. In *AI Magaznine*, pages 14(4):17–42, 1993.

Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, pages 191–246, 2006.

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

Barney Pell, Erann Gat, Ron Keesing, Nicola Muscettola, and Ben Smith. Robust periodic planning and execution for autonomous spacecraft. In *Proc. of IJCAI-97*, 1997.

Michael Pinedo. *Scheduling: Theory, Algorithms and Systems*. Prentice Hall, 2nd edition, 2002.

Martha E. Pollack and Marc Ringuette. Introducing the tileworld: Experimentally evaluating agent architectures. In *AAAI*, pages 183–189, 1990.

Wheeler Ruml and Minh Do. Best-first utility-guided search. In *Proc. of IJCAI-07*, 2007.

Wheeler Ruml, Minh Binh Do, and Markus Fromherz. On-line planning and scheduling for high-speed manufacturing. In *Proc. of ICAPS-05*, pages 30–39, 2005.

Oscar Sapena and Eva Onaindia. An architecture to integrate planning and execution in dynamic environments. In *Proc. of PlanSig-03*, 2003.

David E. Smith and Daniel S. Weld. Temporal planning with mutual exclusion reasoning. In *Proc. of IJCAI-99*, pages 326–333, 1999.

Gerard Verfaillie. Technical coordination unit for online planning and scheduling. In *http://www.informatik.uni-ulm.de/ki/Planet/TCU-olps*, 2002.

Daniel Weld. An introduction to least commitment planning. *AI Magazine*, 15(4):27–61, 1994.