Integrating Reactive and Deliberative Planning in a Household Robot

Jim Blythe and W. Scott Reilly
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
jblythe+@cs.cmu.edu and wsr+@cs.cmu.edu

November 4, 1993

Abstract

Autonomous agents that respond intelligently in dynamic, complex environments need to be both reactive and deliberative. Reactive systems have traditionally fared better than deliberative planers in such environments, but are often hard to code and inflexible. To fill in some of these gaps, we propose a hybrid system that exploits the strengths of both reactive and deliberative systems. We demonstrate how our system controls a simulated household robot and compare our system to a purely reactive one in this domain.

1. Introduction

The typical household is a challenging environment for a robot, partly because of the presence of other agents. The robot must sense and respond to the other agents as it completes its task. In addition, other agents may move furniture and create new cleaning tasks; this leads to a less structured environment, for example, than a nuclear power plant or a road-following task. Thus it will be impossible to pre-program a vacuuming and cleaning robot to perform "blind", without sensing the environment and monitoring the progress of tasks. Some of the changes in the environment will demand reactive responses from the robot, such as interrupting a less important task to pick up a piece of trash. Thus a vacuuming robot must be flexible enough in its design to respond intelligently to a changing set of goals as well as a changing environment.

Such a robot needs to display a range of capabilities not typically found in a single system. *Deliberative systems* that embody powerful techniques for reasoning about actions and their consequences often fail to guarantee a timely response in time-critical situations [6]. Also, *reactive systems* that respond well in time-critical situations typically do not provide a reasonable response in situations unforeseen by the designer [5].

Reactive systems have traditionally been more successful than deliberative ones in controlling agents in dynamic domains, like a typical household. Building a reactive system, however, can be a complex and time-consuming endeavor because of the need to pre-code all of the behaviors of the system for all foreseeable circumstances. An efficient reactive system is also likely to have a narrow area of applicability, for instance a specific house and task set. Deliberative systems are best suited to long-term, off-line planning — effective for static environments, but not for controlling an autonomous agent operating in a dynamic environment. However, deliberative systems are often more robust to varying domains and sets of interacting goals because they employ some form of forward projection.

We have designed a hybrid reactive-deliberative system in an attempt to combine these complementary sets of capabilities, and use it to control a simulated vacuuming robot. In this paper, we compare the performance of the hybrid system with a purely reactive one hand-coded for the same domain. Although the performance of the hybrid system is inferior, we demonstrate that it is close enough to be an attractive option because of its flexibility.

2. The Architecture

2.1. The Component Systems: Hap & Prodigy

The Hap system is designed to execute plans for achieving multiple, prioritized goals. It is related to Firby's RAPs [8]. Hap starts with a set of pre-defined goals and hand-coded plans for achieving those goals. The hand-coded plans are designed to allow Hap to respond to its environment in a timely manner. Hap also allows for demons that dynamically create new goals in appropriate situations. If such a goal has a higher priority than the other active goals, execution of the current plan is interrupted in favor of a plan to achieve the new goal. Hap will attempt to resume its previous plan once it has handled this unexpected event. More details about Hap can be found in [13].

Prodigy 4.0 is a classical deliberative planner that uses means-ends search to create plans from descriptions of operators, given initial and goal state descriptions. Goal statements as well as the preconditions of operators may be arbitrary expressions of first-order logic, involving existential and universal quantification. Prodigy's planning involves a number of choice-points, such as choosing a goal to work on or an operator to use, and Prodigy uses *control rules* to represent information about which choices to make. More details about Prodigy can be found in [2].

A classical planner may take an arbitrary amount of time to complete its task; this is unacceptable for an agent that may require a rapid response. For this reason, we have modified Prodigy to be an instance of an *anytime planner* [4]. This means that the planner can be interrupted at any moment before it completes its task, and will return a portion of a complete plan that will allow the agent to take some actions. This has been done such that the planner is changed only minimally, allowing us to make use of the large body of work on classical planning systems, such as abstraction [12], machine learning to improve planning performance [14, 7, 10] and derivational analogy [16, 11]. More details about the anytime planner can be found in [3].

2.2. Integration

Hap is designed to react quickly and intelligently in a dynamic environment by using stored behaviors when possible. Prodigy is designed to plan for sets of goals that may interact, and to learn to plan more effectively. We integrate these two systems so as to retain the strengths of each, giving primary control of the agent to Hap.

Hap keeps the tree of goals and plans that the agent is pursuing up to date. When there are stored plans available or there is a strict time constraint, Hap will usually act in a pre-programmed way. When Hap has extra time to act or there is no stored plan to handle the current situation, Hap may call Prodigy. Planning is not distinguished from anything else Hap does, so the conditions under which Prodigy is called are contained in the pre-defined Hap productions. This allows the agent builder to create agents that vary along the spectrum between deliberation and reaction as desired. The call to Prodigy includes a predefined subset of the agent's goals for Prodigy to work on and a time bound, so that the agent does not lose reactivity even though planning would otherwise take an arbitrary amount of time to complete.

Since Prodigy reasons about the effects of its actions in the future, it can produce plans that will achieve a number of goals simultaneously, even if they interact in unexpected ways, and contend for the same resources. A pre-programmed approach may fail or produce less efficient behaviour because of these interactions. Prodigy is given a time bound and uses anytime planning techniques to produce the best plan it can within the allowed time. The goals passed to Prodigy are prioritized so that if it cannot solve all the goals in time it will solve the ones with higher priorities.

The integration works, in part, because the communication between Hap and Prodigy takes place at an appropriate level of abstraction, both of state and operators. Like other researchers [9, 15], we have used an abstraction boundary between the reactive and deliberative components of our architecture.

Deliberative planners typically assume a static or near-static world. While this allows the planner to construct a plan, in a dynamic domain this plan will frequently fail because some of its underlying assumptions have become false since the plan was constructed. This is one of the primary reasons deliberative planners have been unsuccessful as agent architectures. We can improve the quality of the plans by making the planner plan in a more static state. We accomplish this by providing the planner with an abstraction of the state that includes the more static elements and allowing it to plan in that state.

Prodigy uses the abstract state to generate an abstract plan, that is, a plan that is a sequence of Hap subgoals rather than a set of concrete actions. Hap uses its set of stored reactive plans to execute these subgoals in the dynamic world. Hap is able to fill in the dynamic details that Prodigy did not plan for. Also, Hap will often have numerous alternative plans for achieving a subgoal, so a single Prodigy plan can generate very different behavior depending on the current state of the world.

3. Mr. Fixit, a Household Robot

We have designed and built a simulated household robot agent, Mr. Fixit, using this architecture. Mr. Fixit lives in a simulated environment built with the Oz system [1]. Figure 1 gives a rough layout of the environment that Mr. Fixit inhabits. Many of the details of the world, which includes over 80 objects, have been left out as they

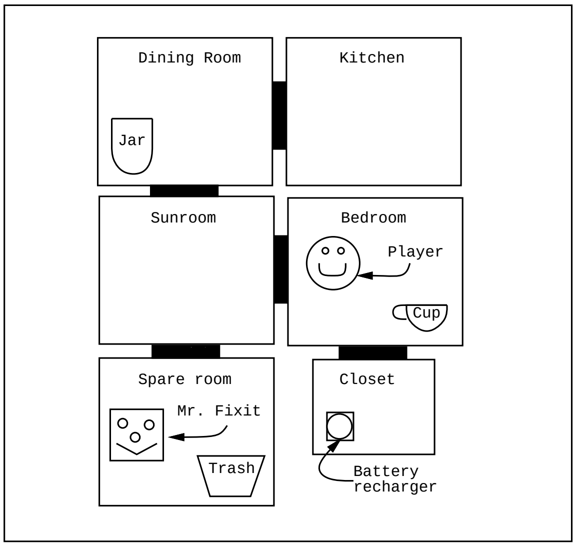

Figure 1: Rough layout of the simulated household robot environment

```
Player:  *BREAK  China-Cup
Fixit:     *GO-TO  Sunroom

[Plan: Goal: (and (recharge)
                          (thrown-out-trash))
           Time limit: 1 second
           Plan: ((goto recharger)
                     (recharge)
                     (goto cup)
                     (get cup)
                     (goto trash-can)
                     (put cup trash-can))]

Fixit:     *GO-TO  Bedroom
             *GO-TO  Closet
             *RECHARGE
             *GO-TO  Bedroom
             *TAKE  China-Cup
             *GO-TO  Sunroom
             *GO-TO  Spare-Room
             *PUT  China-Cup in Trash-Can

[Plan: Goal: (clean-dining-room)
            Time limit: 1 second
            Plan: ((goto dining-room)
                      (vacuum dining-room))]

Fixit:     *GO-TO  Sunroom
             *GO-TO  Dining-Room
Player:  *GO-TO  Sunroom
Fixit:     *VACUUM  Dining-Room
Player:  *GO-TO  Dining-Room
Fixit:     *SAY  "Hello."  to  Player
Player:  *BREAK  Jar

[Plan: Goal: (and (clean-dining-room)
                          (thrown-out-trash))
           Time limit: 1 second
           Plan: ((goto jar)
                     (get jar)
                     (goto trash-can)
                     (put jar in trash-can)
                     (goto dining-room)
                     (vacuum dining-room))]

Fixit:     *TAKE  Jar

[Plan: Goal: (and (clean-dining-room)
                          (thrown-out-trash)
                          (recharge))
           Time limit: 0.5 seconds
           Plan: ((goto recharger)
                     (recharge))]

              *GO-TO  Sunroom
              *GO-TO  Bedroom
Player:  *GO-TO  Sunroom
Fixit:     *GO-TO  Closet
              *RECHARGE

[Plan: Goal: (and (clean-dining-room)
                          (thrown-out-trash))
          Time limit: 1 second
          Plan: ((goto trash-can)
                    (put jar trash-can)
                    (goto dining-room)
                    (vacuum dining-room))]

Fixit:     *GO-TO  Bedroom
              *GO-TO  Spare-Room
              *PUT  Jar  in  Trash-Can
              *GO-TO  Sunroom
              *SAY  "Hello."  to  Player
              *GO-TO  Dining-Room
              *VACUUM  Dining-Room
```

Figure 2: Sample trace of the household robot

		Time bound (seconds)			
		0.25	0.50	1.00	2.00
Number of goals	1	0.90	0.98	1.00	1.00
	2	0.80	0.96	1.00	1.00
	4	0.38	0.52	1.00	1.00
	8	0.13	0.16	0.62	1.00

Table 1: Average proportion of goals solved.

are unimportant for the behaviors described here. The *player* is a human user that is also interacting with the simulation.

The robot has a number of important goals that define its behavior. They are, from most to least important: recharge battery when low, clean up broken objects, greet the player, vacuum dirty rooms, and roam the house looking for tasks to perform. Mr. Fixit keeps a simple model of the world, but given his limited sensing abilities and the existence of an unpredictable user in the world, his model will often be incomplete or incorrect.

Figure 2 shows a trace of a run of this simulation, which has been edited for brevity and clarity. The trace begins with Mr. Fixit in the spare room performing rounds while the Player is in the bedroom. The player breaks the cup. Fixit notices the broken cup at the same time that his battery needs recharging. Prodigy is notified of the two goals and is able to find a plan to solve them both within one second. Fixit executes the plan without any problems. During the execution of the plan Fixit goes by the Player twice without stopping for the traditional greeting. This is because the current plan being executed has a higher priority than greeting the user.

Once the two goals are accomplished, Fixit notifies Prodigy of a goal to clean the dining room, which is planned for successfully. During the execution of this plan, the player enters the dining room and is greeted because vacuuming is a low priority goal. Despite the warm greeting, the Player decides to break the jar. Fixit notices this and plans to clean up the mess and then return to vacuuming. While cleaning up the broken jar, Fixit's battery again gets low. The three pending goals are sent to Prodigy along with a half second time limit. Prodigy is able to solve only the goal to recharge in this amount of time and Fixit executes that plan successfully. Once that is done, Fixit calls Prodigy to replan for the other two goals and executes them.

3.1. Discussion

One of our main concerns in building an agent like Mr. Fixit is how quickly Prodigy is able to plan for goals in the domain. If Prodigy isn't given enough time to generate plans for even single goals, then the agent is going to be stuck. On the other hand, if we know that Prodigy is going to generally be given more than enough time to achieve some subset of goals, then there are tradeoffs in speed vs. plan interleaving. For example, say Fixit knows about 2 broken objects that need to be thrown out. If Prodigy only has time to solve one of the goals, it will return a plan that may be sub-optimal with respect to solving both goals.

We tested Prodigy's ability to solve goals quickly in this domain by giving it problems to solve with varying numbers of goals and varying time bounds. The results of the test are reproduced in table 1. When Prodigy is given a 2 second time bound, it can always solve the goal conjunct in this relatively simple domain, making the anytime planning redundant. With smaller time bounds, Prodigy is only able to solve for a proper subset of the goals. If Prodigy only returned complete solutions Hap wouldn't get any useful information in these time-critical cases.

This information can be used to guide the design of the domain specification given to Prodigy. If we know that Prodigy will often only have 0.25 or 0.5 seconds with which to work, we will want to generate simple serial plans. If the domain allows Prodigy to take 2 or more seconds, we can write control rules for Prodigy that will produce more efficient plans but that will usually take longer before completely planning for any single goal.

4. Experiments: Deliberative+Reactive vs. Reactive

We used the household robot domain as a testing ground for our architecture. We have already discussed some of the obvious benefits to using a deliberative planner as part of an agent architecture, but if the architecture doesn't perform well, these benefits may be overshadowed. To evaluate our architecture we decided to test it against a purely reactive agent. This agent was written entirely in Hap and was designed specifically for this domain. In general, the hand-coded plans were similar to the ones that Prodigy generated, but we also added specific interleaved plans for throwing out multiple pieces of garbage.

We did not expect the hybrid agent to do quite as well as the reactive agent, but if our architecture could come close, then we can reap the benefits of using a deliberative architecture without concern for losing reactivity or performance efficiency.

The domain described in section 3 was modified as follows: (1) We changed the procedure for throwing out garbage so that the robot first had to get a bag out of a cabinet in the kitchen, then put the trash in the bag, then put the bag in a trash bin in the sunroom; (2) A second robot, the Destructo2000 was added to the environment. This robot would generate cups and break them on the floor with some probability that we could control; (3) A (slightly unrealistic) phone was placed in the bedroom. With a 10% chance the phone would ring during any turn it was off the hook. If the robot hadn't answered the previous call, it was lost. Until another call came in, the previous caller would keep ringing; (4) All the rooms started in need of vacuuming and didn't become dirty again once vacuumed. (5) The player was removed from the simulation; (6) The new goal priority ordering was (from most to least important): recharge battery, answer phone, throw out trash, vacuum rooms.

The phone and the battery created occasional interrupts in behavior that were kept constant over every run. We were able to control how dynamic the environment was by changing the chance that the Destructo2000 would drop a cup. We ran both the hybrid and pure reactive robots with this chance at 5%, 8%, and 10%. We also ran the hybrid system at 3%.

The planner in the hybrid system had a 3 second time bound. We feel this was reasonable because, although a household robot might stand a longer delay without losing reactivity, its world model is also likely to be more complicated. Although we didn't do as complete an analysis of this version of the domain as was described in section 3.1, some informal experiments showed that the plans required to handle the new trash procedure required enough time that 3 seconds was not enough to consistently create interleaved plans for throwing out garbage. Because of this, the hybrid agent always generated serialized plans for throwing out trash. These were sub-optimal plans, but Prodigy was always able to solve at least one goal in the allotted time.

Figure 3 graphs how well the hybrid robot did in keeping up with the Destructo2000. It's fairly clear that at 5%, 8%, and 10%, Mr. Fixit is falling further and further behind but at 3% he is able to keep up. The purely reactive agent that we created had hand-coded plans for throwing out multiple pieces of trash simultaneously, and tuned to the layout of the building. Because of this we expected the reactive agent to perform better on this task. Our expectations were realized and figure 4 shows how at 10% the reactive agent is falling behind, but at both 8% and 5% the robot seems able to keep up.

The relative performances of the two systems in picking up cups was expected. That the hybrid system fared as well as it did on this task is reassuring. Admittedly, this is still a relatively simple domain so differences in performance will tend to be small, but at the same time moving to more complex domains will make it even harder for builders of reactive agents to create a complete and efficient set of behaviors for all situations.

A somewhat surprising result was how well the hybrid system did on the other tasks it was given. First, both agents performed at 100% on battery recharging, which was the most important goal the agent was given. Second, the hybrid agent was able to answer the phone at a rate of 78% as compared to 61% for the reactive agent. Third, at cleaning rooms, the hybrid agent was able to do almost as well as the reactive agent in the 10% domain and even slightly better in the 5% domain. This is shown in figure 5. We expect that the hybrid system outperforming the reactive system on these tasks is either a matter of a limited data set (10 runs for each robot-environment pair) or a product of the logistics of this domain and not attributable to our architecture.

Each time cycle in the simulations with the reactive agent took 9.2 seconds. This includes both agents, the physical world simulation, and the data gathering. When we changed to a hybrid system, this increased to 9.7 seconds. So on average, the hybrid agent only took 0.5 seconds longer to choose an action than the reactive agent.

5. Conclusions

This preliminary version of the architecture can be improved in several obvious places. For example, our current model of anytime planning only gives credit to states that solve one or more top-level goals. If no goal can be completely solved within the time bound, the planner will be unable to suggest an action. The planner is unable to make use of partial plans built during previous calls from Hap, and we have not yet investigated the impact of Prodigy's machine learning capabilities on this task.

The hybrid architecture we built provides power to agent builders which will be useful in designing agents for a variety of domain types. In highly dynamic domains where quick action is vital, the agent builder can put reactive behavior to deal with most situations into Hap and design the Prodigy system to return quickly. This

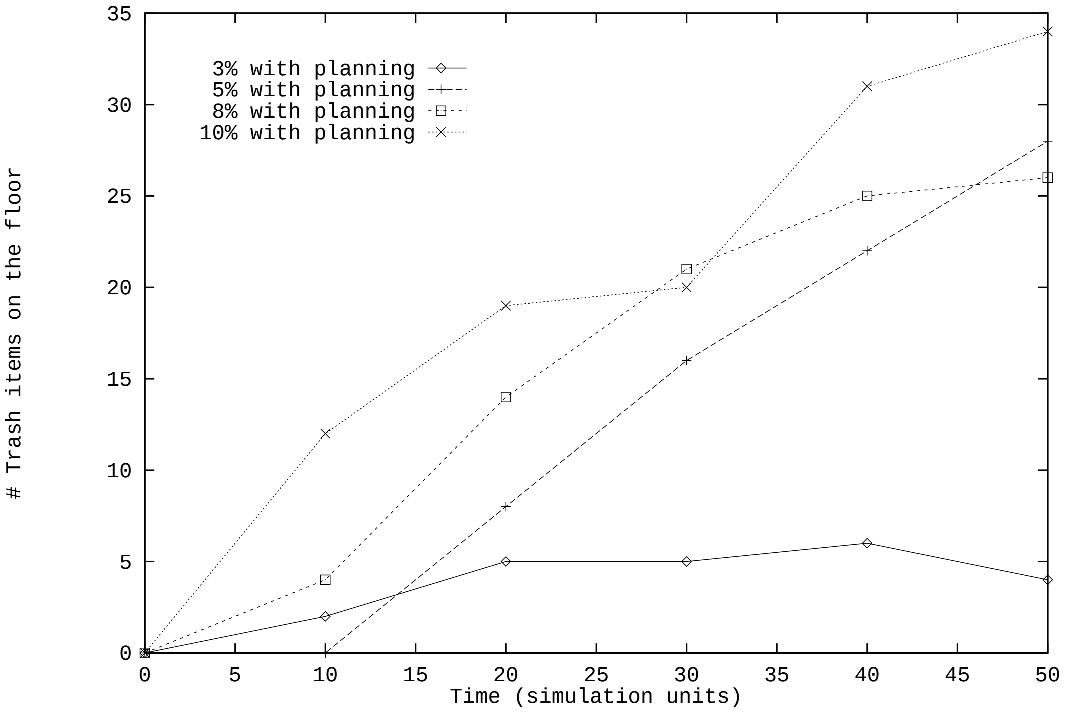

Figure 3: Hybrid robot cleaning up cups.

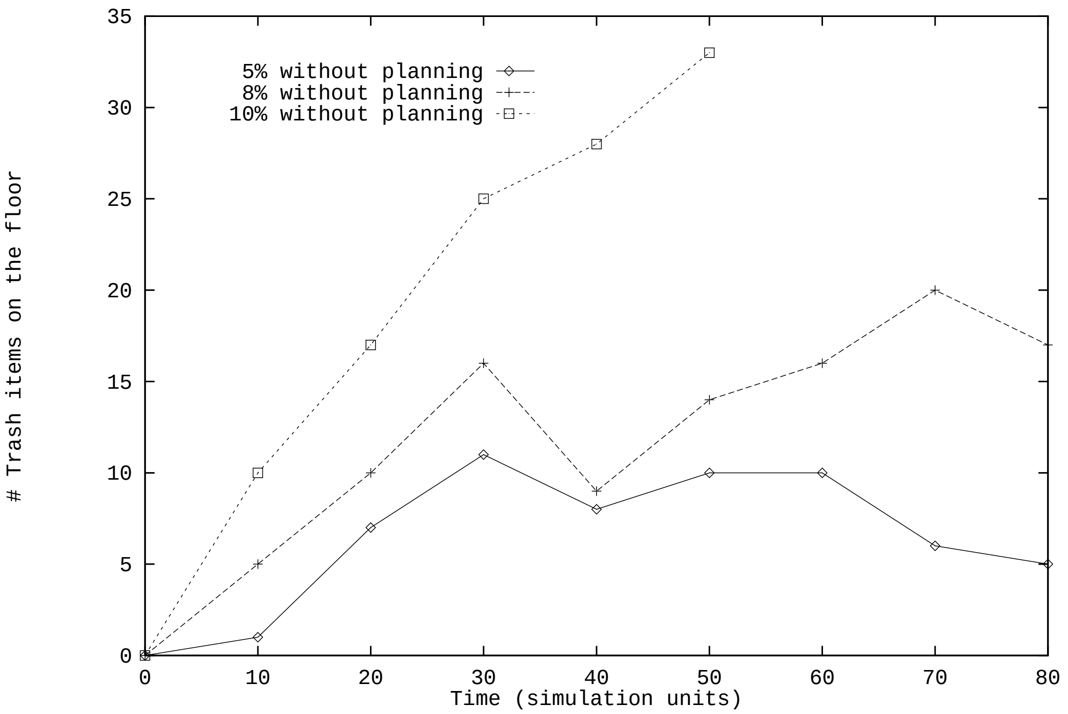

Figure 4: Pure reactive robot cleaning up cups.

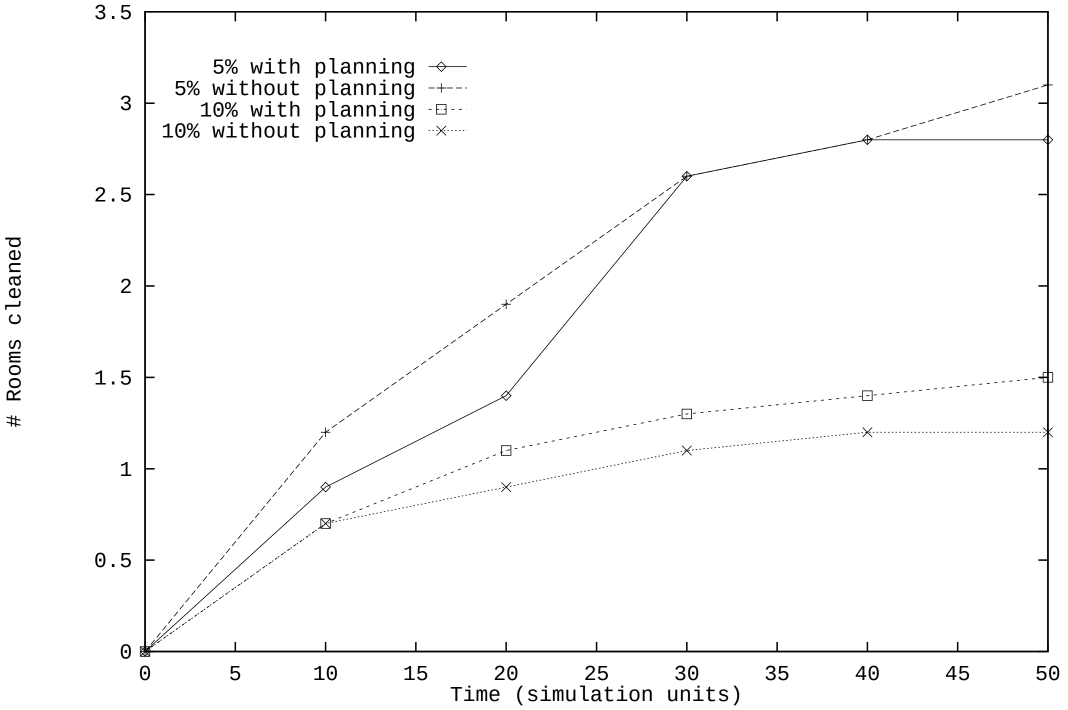

Figure 5: Robots vacuuming rooms.

will often be at the expense of plan quality because the interactions between goals will not be explored. In more stable domains, Prodigy can be given more time to create efficient plans. In fact, in some situations it might be the case that Prodigy is able to solve some resource critical problem that a reactive system might not solve at all.

References

[1] Joseph Bates. Virtual reality, art, and entertainment. *PRESENCE: Teleoperators and Virtual Environments*, 1(1):133–138, 1992.

[2] Jim Blythe, Oren Etzioni, Yolanda Gil, Robert Joseph, Alicia Pérez, Scott Reilly, Manuela Veloso, and Xuemei Wang. Prodigy4.0: The manual and tutorial. Technical report, School of Computer Science, Carnegie Mellon University, 1992.

[3] Jim Blythe and W. Scott Reilly. Integrating reactive and deliberative planning for agents. Technical Report CMU-CS-93-135, Carnegie Mellon University, May 1993.

[4] Mark Boddy. *Solving Time-Dependent Problems: A Decision-Theoretic Approach to Planning in Dynamic Environments*. PhD thesis, Brown University, May 1991.

[5] Rodney Brooks. Integrated systems based on behaviors. In *Proceedings of AAAI Spring Symposium on Integrated Intelligent Architectures*, Stanford University, March 1991. Available in *SIGART Bulletin*, Volume 2, Number 4, August 1991.

[6] David Chapman. Planning for conjunctive goals. *Artificial Intelligence*, 32:333–378, 1987.

[7] Oren Etzioni. *A Structural Theory of Explanation-Based Learning*. PhD thesis, School of Computer Science, Carnegie Mellon University, 1990. Available as technical report CMU-CS-90-185.

[8] James R. Firby. *Adaptive Execution in Complex Dynamic Worlds*. PhD thesis, Department of Computer Science, Yale University, 1989.

[9] Erann Gat. On the role of stored internal state in the control of autonomous mobile robots. *AI Magazine*, 14(1):64–73, Spring 1990.

[10] John Gratch and Gerry DeJong. Composer: A probabilistic solution to the utility problem in speed-up learning. In *AAAI 92*, 1992.

[11] Kristian J. Hammond. *Case-based Planning: An Integrated Theory of Planning, Learning and Memory*. PhD thesis, Yale University, 1986.

[12] Craig A. Knoblock, Josh D. Tenenberg, and Qiang Yang. Characterizing abstraction hierarchies for planning. In *National Conference on Artificial Intelligence*, 1991.

[13] A. Bryan Loyall and Joseph Bates. Hap: A reactive, adaptive architecture for agents. Technical Report CMU-CS-91-147, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, June 1991.

[14] Steven Minton. *Learning Effective Search Control Knowledge: An Explanation-Based Approach*. Kluwer, Boston, MA, 1988.

[15] Alberto Segre and Jennifer Turney. Planning, acting and learning in a dynamic domain. In S. Minton, editor, *Machine Learning Methods for Planning*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[16] Manuela M. Veloso. *Learning by Analogical Reasoning in General Problem Solving*. PhD thesis, Carnegie Mellon University, august 1992.