Using Reactive Rules to Guide a Forward-Chaining Planner

Murray Shanahan
Department of Electrical and Electronic Engineering
Imperial College
Exhibition Road, London SW7 2BT, England
m.shanahan@ic.ac.uk

Abstract

This paper presents a planning technique in which a flawed set of reactive rules is used to guide a stochastic forward-chaining search. A planner based on this technique is shown to perform well on Blocks World problems. But the attraction of the technique is not only its high performance as a straight planner, but also its anytime capability. Using a more dynamic domain, the performance of a resource-bounded version of the planner is shown to degrade gracefully as computational resources are reduced.

Introduction

As Bacchus and Kabanza have demonstrated, the use of domain-specific rules to guide a forward-chaining planner is a promising line of research (Bacchus & Kabanza 2000). Using only a handful of temporal logic formulae as heuristics, their TLPlan system outperforms many state-of-the-art domain-independent planners on hard Blocks World problems. In a similar vein, the present paper describes a forward-chaining planner whose search is controlled by domain-specific rules. But where TLPlan uses formulae of temporal logic to guide the search, the present planner uses a set of reactive rules.

Although the resulting planner is reasonably fast, the aim of the present work isn't the construction of a high-performance planner. Rather, the chief aim is to supply a system that seamlessly integrates reactive behaviour and planning at the same level. This is achieved because the reactive rules enable the planner to behave as an anytime algorithm (Dean & Boddy 1988; Zilberstein & Russell 1993). The planner can always supply a partial solution, including an action to execute right away, no matter how far into its computation it has gone. But the solutions it finds increase in quality with the time available to search for them.

Given sufficient time to respond, the planner generates a complete plan from initial state to goal. In this case, the reactive rules serve to speed up the search. Given insufficient time to find a complete plan, the system responds with a partial plan constructed (mostly) out of actions recommended by the reactive rules, but with the benefit of look-ahead to favour those that are least dangerous and most promising. Given very little time to respond, the system doesn't even have the luxury of look-ahead, and all it can offer is a partial plan comprising a single action recommended by the reactive rules. In other words, it starts to behave as the reactive rules would on their own.

Although some pioneers of the use of reactive rules rejected deliberative planning altogether (Brooks 1986; Agre & Chapman 1987), in the late Eighties and early Nineties, a variety of ways of reconciling planning with reactivity were studied. Prominent examples include the work of Schoppers (1987), Drummond (1989), and Mitchell (1990). In robotics, hybrid architectures combining a low-level reactive layer with a high-level deliberative layer, such as that described by Gat (1992), have become commonplace, and are now to be found in the remotest corners of the Solar System (Pell *et al.* 1997). However, none of this work looks at the possibility of using reactive rules to guide a forward-chaining planner, possibly because the idea only starts to look plausible in the light of Bacchus and Kabanza's more recent work.

Useful but Imperfect Reactive Rules

Consider the following pair of goal-directed reactive rules for solving Blocks World problems. As we'll see, these rules are useful but imperfect.

RULE BW1

Move X onto Y if X is on Y in the goal and everything under Y is correct with respect to the goal

RULE BW2

Move X from Y to Z if anything under Y is incorrect with respect to the goal and everything under Z is correct with respect to the goal.

Let's investigate the performance of these rules on some examples, beginning with the trivial Blocks World problem depicted in Figure 1. Starting in the initial state, the successive application of Rules BW1 and BW2 yields the following sequence of actions leading to the goal: move A to the table, move B to the table, move C to B, move A to C.

In this case, the rules generate a unique sequence of actions. But in general, such rules are non-deterministic. For example, in the initial state of the Blocks World problem in Figure 2, the rules recommend three possible actions – move E to the table, move C to the table, and move I to the table.

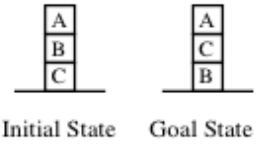

Figure 1: Blocks World Problem bw-small.

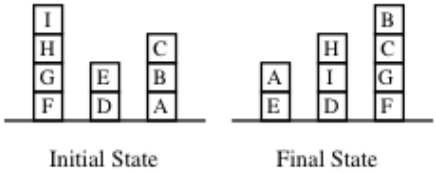

Figure 2: Blocks World Problem bw-large-b.

Rules BW1 and BW2, though simple, are effective with many Blocks World problems. Rule BW1 alone can generate an optimal 6-step solution to the problem of Figure 2 using negligible CPU time. However, these two rules aren't guaranteed to find a solution to *any* Blocks World problem. Problem bw-large-b+ is identical to bw-large-b except that blocks E and A are swapped in the final state. Given this problem, one possible sequence of applications of Rules BW1 and BW2 leads to the state shown in Figure 3. In this state neither Rule BW1 nor Rule BW2 is applicable.

Figure 3: A Stalled State.

Not only can the rules lead to stalled states. They can also recommend actions that are positively harmful. Take the problem in Figure 4, for example. In the initial state, one of the actions recommended by the reactive rules is to move A onto C, where it will prevent B from getting to its final destination.

In general, the fact that a set of reactive rules has been effective on a large class of problems is no guarantee that it will solve the next problem that comes along. If the rules are hand-coded or pre-computed off-line, their correctness and completeness can be proved. But if the rules are learned on the fly, by a reinforcement learning algorithm, for example, there's no guarantee of completeness, and no possibility of off-line validation.

It's easy to see that, in this case, an additional rule would solve the stalled state problem. The new rule would cater for the case when X needs to be moved off Y, even though the blocks below Y are correctly placed. But this is beside the point. The pertinent observation here is that an imperfect set of rules, of the sort that a learning algorithm might generate, can still be useful.

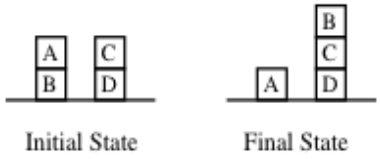

Figure 4: Rule BW2 Goes Wrong.

The system proposed here is designed to exploit the availability of useful but imperfect sets of reactive rules. As long as the rules are effective some of the time, they can be used to efficiently guide the search of a forward-chaining planner. The reactive rules serve a similar function to the temporal logic formulae in TLPlan. But unlike TLPlan's heuristic formulae, the reactive rules can be used in stand-alone mode when a fast response is needed. Moreover, although the examples of reactive rules used in this paper are all indexed on a goal, the system can also function with goal-less reactive rules (of the kind that are prevalent in biologically-inspired approaches to robotics). In the absence of an explicit goal, it isn't an advisable strategy for an intelligent agent to cease activity altogether, which is why goal-less reactive rules that encourage exploration and play are useful. Furthermore, at times when there is no explicit goal, an agent still has to react to ongoing events. In either case, it's advantageous for the agent to look ahead before executing an action. In contrast to conventional approaches to planning and plan execution, the present system can accommodate this requirement.

Linear Iterative Lengthening

Given unbounded time to act, the system under discussion is guaranteed to produce a plan if one exists. Given very little time to act, the system can still suggest the next action to be executed using the reactive rules alone. In between these two extremes, the reactive rules and the search-based planner operate together, producing a high-quality partial plan with an immediately executable action. Moreover, as we'll see, in this intermediate region, the system can discover plans that the reactive rules would miss altogether and that a conventional search-based planner would fail to find within an acceptable time.

The basic idea of using reactive rules to guide a forward-chaining planner can be implemented in a variety of ways. During the work carried out for this paper, several different approaches were tried, on a variety of Blocks World problems, using Rules BW1 and BW2 from previous Section as the reactive guide. Only the most promising technique is described here. This also happens to be the simplest of the implementations.

The most successful planner uses an algorithm we'll call *linear iterative lengthening*. In the following pseudo-code, the algorithm is invoked by a call to ILPlan(G,S,P), where G is the goal state, S is the initial state, and P is the returned plan. A state is a set of fluents, to which the closed world

assumption is applicable. The variable L is the length bound.

```
1 Procedure ILPlan(G,S,P)
2   L := 1
3   FPlan(G,S,L,P)
4   While plan not found
5       L := L + 1
6       FPlan(G,S,L,P)
```

The FPlan procedure tries to construct a plan, and exits if it finds one or if the plan it's working on exceeds the current length bound. In this case, the length bound is incremented, and the planner goes round the loop again, starting with an empty plan.

```
7 Procedure FPlan(G,S,L,P)
8   P := []
9   While Length(P) < L and S ≠ G
10      Let RL be the set of actions recommended
11          by the reactive rules for state S
12      Let EL be the set of remaining actions
13          executable in S
14      If RL is empty
15      Then randomly select A from EL
16      Else randomly select A from either
17          EL or RL with a bias towards RL
18      P := [A|P]
19      S := result of executing A in S
```

The FPlan procedure uses forward chaining to construct a single candidate plan in progression order. Successive actions are randomly selected, but with a heavy bias towards those recommended by the reactive rules.

While this planner is sound, it obviously isn't complete. No attempt is made to systematically explore the search tree. It is, however, complete in a stochastic sense: as the execution time tends to infinity, the probability of finding a solution, if one exists, tends to one. (In the reported experiments, though, the bias towards recommended actions was made 100%, sacrificing completeness even in this stochastic sense.) In practise, theoretical completeness is a less interesting property than how likely it is that the planner will actually find a solution in an acceptable timeframe. On this score, how good can an algorithm be that burrows apparently blindly down a single branch of the search tree at a time?

The answer is that everything depends on the quality of the reactive rules guiding the planner. At one extreme, if the reactive rules are very poor, the planner really is working blindly and its performance is appalling. At the other extreme, if the reactive rules are extremely good, they can solve any problem directly, and the planner is redundant. The planner's performance is most impressive in the intermediate cases, where the reactive rules are effective on a small class of problems, but are unable to solve all problems by themselves. As we'll see in the next section, the Blocks World rules of previous section fall into this intermediate category, and the planner's performance is correspondingly impressive.

But for the planner to be efficient, even in these intermediate cases, it has to be able to make effective guesses when the reactive rules are inapplicable. The algorithm above simply makes a random choice. This works fine with the Blocks World, because it's just enough to jog the planner out of a stalled state. But in other domains, it will be necessary to employ a little more sophistication. One way to do this is to introduce an evaluation function for scoring potential successor states, and to use this to select the most promising (and least dangerous) next action. This extension to the basic algorithm will be discussed in a later section.

Blocks World Experiments

As already stated, the thrust of this paper is *not* the design of a stand-alone high-speed planner. The issue at hand is architecture rather than performance. In designing a whole agent, such as a robot, we need to integrate features such as deliberative planning, reactive behaviour, and heuristics in a single control system that also interleaves computation and action. Accordingly, the experimental results presented in this section should be taken solely as a "proof of concept".

The result of applying the planner to a number of Blocks World problems are presented, using the two reactive rules discussed previously to guide the search. Problems bw-small, bw-large-b, and bw-large-b+ were described in a previous section. Problems bw-large-c and bw-large-d were taken from the BlackBox test suite (Kautz & Selman 1999). Problem bw-large-c+ is a modification of bw-large-c. Each problem was submitted to the planner 10 times, first using only Rule BW1, and then using both Rules BW1 and BW2. The results using only Rule BW1 are presented in Table 1.

The planner was implemented in LPA Prolog 32 on an Apple iMac with a 233MHz G3 processor. Note that, in a sense, the true metric here is not the time taken to find a solution, but whether or not the planner can find an answer *at all* within a reasonable timeframe. If a planner can deliver an answer to a problem in a matter of seconds, then turning this into milliseconds is simply a matter of optimisation and computing power. On the other hand, when the system fails to present an answer having been left to run overnight, we suspect its practical limits have been reached.

Problem	Av. Time	Av. Length
bw-small	0.02s	4.0
bw-large-b	0.12s	6.0
bw-large-b+	0.19s	7.0
bw-large-c	1.60s	14.9
bw-large-c+	7.13s	32.9
bw-large-d	33.88s	18.5

Table 1: Using Rule BW1 Only

The planner consistently solves bw-large-b in around one tenth of a second. The plan is always 6 steps long, which is optimal. Recall that the reactive rules alone fail to solve this problem altogether. Moreover, 9-block problems such as bw-large-b and bw-large-b+, though soluble in seconds by the current generation of domain independent planners, exemplified by BlackBox (Kautz & Selman 1999),

were large enough to overwhelm earlier planners, such as UCPOP (Penberthy & Weld 1992).

The present planner consistently solves the 15-block problem bw-large-c in less than 2 seconds. However, bw-large-c is a relatively "easy" problem for its size. A solution can be found just by repeatedly following Rule BW1. In other words, there's always a block that can be moved to its final destination, and the planner never has to dismantle a tower just to get at a particular block.

Problem bw-large-c+ is designed to be more difficult. The initial state is that of bw-large-c. But the goal state is the same as the initial state with only the bottom two blocks of each tower swapped over. This requires the planner to take each tower apart, swap the bottom blocks, and then rebuild it. The planner also performed satisfactorily on this problem, consistently finding a solution in under 10 seconds.

Finally, the planner was run on a 19-block problem, bw-large-d. Until recently, Blocks World problems of this size were beyond the capability of domain-independent planners. On average, the present planner can solve bw-large-d in around 30 seconds.

Table 2 presents the results of running the planner on the same problems, but using both Rules BW1 and BW2. The performance turns out to be worse than with Rule BW2 absent, with respect to both time and length. This is because of Rule BW2's tendency to recommend the occasional harmful action. Since the recommendations of reactive rules are given such weight, their mistakes are very costly.

Problem	Av. Time	Av. Length
bw-large-a	0.02s	4.0
bw-large-b	0.72s	8.5
bw-large-b+	0.96s	9.5
bw-large-c	14.27s	20.0
bw-large-c+	21.52s	24.0
bw-large-d	58.36s	28.4

Table 2: Using Rules BW1 and BW2.

The Kids World Domain

The results reported in the previous show that the planner outperforms fast domain-independent planners in the Blocks World, using only a single guiding reactive rule, one that is incapable of solving Blocks World problems on its own. This is an encouraging result. But it doesn't show off one of the most attractive features of a forward-chaining planner guided by reactive rules, which is its anytime capability.

In this section, a new domain is introduced, called Kids World, which is adjustably dynamic, in the sense that unexpected events can occur with a preset probability. (In other respects, Kids World resembles the Logistics domain used to assess planners in planning competitions.) Kids World problems will be tackled with a resource-bounded version of the planner previously presented, which is embedded in a sense-plan-act cycle that executes plans and reacts to ongoing events.

Two properties of this system are demonstrated using Kids World problems. First, it's demonstrated that the planner's performance degrades gracefully as its resource bound is decreased. Second, it's shown that the system can solve Kids World problems in real time, even with the frequent occurrence of unexpected events.

The Kids World domain comprises the four fluents and five actions summarised in Table 3 below. The action Move(x) affects the location of both the parent and any child they are carrying. The Open(d) and Close(d) actions have the precondition that the parent isn't carrying anything. To move from one location to another, the connecting door has to be open.

The particular Kids World problem used in the experiments reported here involves two children, Kerry and Liam, and three locations — the house, the street, and the car. These locations are connected by doors in the obvious way. In the initial state, the parent and both children are in the house and the doors are all closed. In the final state, everyone is in the car and both the children are happy. A crucial additional twist to the problem is that Kerry becomes unhappy if Liam is put in the car first.

Fluent/Action	Meaning
Location(x)	Parent is in location x
Location(c,x)	Child c is in location x
Carrying(c)	Parent is carrying child c
Happy(c)	Child c is happy
IsOpen(d)	Door d is open
Move(x)	Go to location x
PickUp(c)	Pick child c up
PutDown(c)	Put child c down
Open(d)	Open door d
Close(d)	Close door d

Table 3: Kids World Fluents and Actions.

Planning in Kids World would be much easier if children stayed where they were put. Then, planning could be studied without having to consider plan execution. But to make Kids World problems realistic, after every action the parent performs, there's a certain probability that one or other of the children will run off somewhere completely unexpected. In Experiments kw-1 and kw-2 described in the next section, this probability is set to zero, but in Experiment kw-3, it is set to 0.1.

A set of seven reactive rules were used in the experiments reported here. Two examples follow. Note their nondeterminism. Suppose the parent is in the car carrying Liam, while Kerry is still in the street. Then Rule KW1 recommends putting the Liam down, while Rule KW2 recommends going back for Kerry.

RULE KW1

If you're carrying a child and you're in the location the child has to end up, then put the child down

RULE KW2

If there's a child in another room who needs to be moved, then move towards that room

As with the Blocks World example, these seven reactive rules are deliberately imperfect. On their own, they can solve some Kids World problems. But they cannot solve the problem described above on their own, as they quickly lead to a stalled state. Even with the injection of the occasional random action to jog the system out of a stalled state, the rules still frequently make the fatal mistake of putting Liam in the car first. However, the rules can be used to effectively guide a forward chaining planner.

Planning with Bounded Computation

This section describes the resource-bounded version of the planner. With a low resource bound, the system amounts to the use of reactive rules with look-ahead. But with higher values, the system doesn't commit to an action early by executing it as reactive rules do, even with look-ahead. Instead, it carries out search — backtracking "in the head" rather than in the world.

The linear iterative lengthening algorithm in a previous section can easily be made resource-bounded by incorporating a count of the number of times the body of the while loop in the FPlan procedure is executed. The ILPlanR procedure below maintains a resource bound V. The planning process is terminated if V exceeds a predefined bound F.

```
1 Procedure ILPlanR(G,S,P)
2   L := 1; V := 0
3   FPlanR(G,S,L,P,V)
4   While plan not found and V < Φ
5       L := L + 1
6       FPlanR(G,S,L,P,V)
```

If V exceeds F, the system returns the best partial plan it has found so far. The best partial plan so far is maintained as B in the FPlanR procedure below, and is determined using an evaluation function, Score, on partial plan (see (Korf 1990)).

```
7 Procedure FPlanR(G,S,L,P,V)
8   P := []; B := []
9   While Length(P) < L and S ≠ G and V < Φ
10      Let RL be the set of actions recommended
11          by the reactive rules for state S
12      Let EL be the set of remaining actions
13          executable in S
14      If RL is empty
15      Then randomly select A from EL
16      Else randomly select A from RL
17      P := [A|P]
18      S := result of executing A in S
19      If Score(P) ≤ Score(B) Then B := P
20      V := V + 1
21  If V ≥ Φ Then P := B
```

When this planner is embedded in a sense-plan-act cycle, it exhibits anytime properties even *without* a carefully designed evaluation function. Table 4 shows the results of Experiment kw-1, a Kids World experiment in which Score(P) = 0 for all P. In other words, the planner simply returns the partial plan it's currently working on when the resource bound is exceeded.

In Experiment kw-1, the planner has to plan and carry out a sequence of actions that solves the Kids World problem described above. The probability of an unexpected event after a parental action is set to zero. In other words, the kids stay put. After each action the system executes, it *replans from scratch*. This is perfectly feasible, as the planner solves Kids World planning problems very quickly.

Resource Bound	Av. Actions	Aborts
1000	17.4	0
500	17.0	1
200	17.9	5
100	22.5	10
50	23.0	16
10	24.9	15
2	–	30

Table 4: Results of Experiment kw-1.

The system was run 30 times for each value of the resource bound. A run was aborted after 50 actions, as this indicates that the planner has made Kerry irreversibly unhappy by putting Liam in the car first. The number of actions executed was averaged over all the successful runs. As Table 4 shows, the performance of the system degrades gracefully as the resource bound is reduced, both in terms of the number of aborted runs and the length of the successful runs.

When the resource bound is 1000, the planner can always find a solution, so there are no aborted runs. When the resource bound is 2, the system is behaving almost as if it were using the reactive rules on their own, and all runs are aborted. For intermediate values of the resource bound, we get intermediate levels of success. When the resource bound falls below 200, we start to see an increase in the length of the successful runs as well as a continuing increase in the number of aborted runs.

To see why this graceful degradation occurs, in spite of the fact that the planner doesn't evaluate intermediate, partial plans, consider how the system behaves as it gets closer to the goal. When the goal is too far away, the planner's resource bound will be exceeded before it finds a plan, and the first action in the partial plan it returns will be one the reactive rules would have chosen on their own. But the closer the goal gets, the more likely it becomes that the planner will find a complete plan before the resource bound is exceeded. The higher the resource bound, the more rapidly this likelihood increases.

Although the planner exhibits anytime properties even when the evaluation function returns a constant, the degradation in the planner's performance can be made more graceful still if a less trivial evaluation function is used. Without such an evaluation function, the partial plans returned when the planner fails to find a complete plan are of uniform quality whatever the resource bound. With a proper evaluation function, a high resource bound means the planner is more likely to find a complete plan, just as before. But it will also produce better quality partial plans when it can't find a complete plan.

In Experiment kw-2, a very simple evaluation function was used to demonstrate this. This function simply gives a score of –1 to any partial plan that leaves a child unhappy, and a score of 0 to all other partial plans. The rest of the details of Expriment kw-2 are as for kw-1. The results of this experiment are presented in Table 5, for low values of the resource bound.

Resource Bound	Av. Actions	Aborts
100	20.9	6
50	23.1	11
10	29.0	13
2	32.4	13

Table 5: Results of Experiment kw-2.

Table 5 shows a clear improvement in performance over that obtained with the constant evaluation function, in terms of aborted runs. Even when the resource bound is 2, the planner is still able to find solutions. The increase in average run length at resource bound 10 is misleading, because it's due simply to the presence of a number of long runs that would have led to abortion with the constant evaluation function. When the resource bound dips below about 50, the number of aborted runs levels off at around 12, although the number of actions per run continues to increase as the resource bound goes down. With such a low resource bound, the planner will rarely find a complete plan, so this gradual degradation can only be put down to a correspondingly gradual reduction in the quality of its partial plans, which is what we were aiming for.

In Experiments kw-1 and kw-2, there were no surprise interventions by the children themselves. But in Experiment kw-3, the dynamic potential of the domain is explored, and the probability of a child unexpectedly moving somewhere by itself is increased from zero to 0.1. Sometimes these unexpected events make the planner's job easier, but most of the time, they make it harder.

Table 6 presents the results of Experiment kw-3. The setup is essentially the same as in Experiment kw-2, except that the system is given up to 100 actions before a run is aborted. As before, 30 runs were carried out for each value of the resource bound. In addition to the number of actions averaged over the successful runs, the mean response time per action is given, to indicate that planning is taking place in a realistic timeframe for robotic applications.

The results of Experiment kw-3 show that the system is consistently able to solve the problem in spite of interference in plan execution. As before, we see a steady deterioration in performance as the resource bound is decreased. The similar response times for resource bounds of 1000 and 500 suggest that the planner doesn't generally require as many as 1000 resource units to find a plan, and this is confirmed by the total absence of aborted runs even with a resource bound of 200.

Resource Bound	Av. Actions	Aborts	Mean Response Time
1000	25.8	0	599ms
500	26.5	0	627ms
200	28.1	0	521ms
100	40.9	1	347ms
50	46.6	7	199ms
10	51.6	12	44ms
2	55.1	10	11ms

Table 6: Results of Experiment kw-3.

Concluding Remarks

How does the system described here differ from Bacchus and Kabanza's TLPlan, from which it takes its inspiration? First, TLPlan's temporal logic formulae have to be correct for all plans in order to preserve the completeness of the planner. Reactive rules, on the other hand, only supply recommendations. They can be completely wrong without threatening the planner's (stochastic) completeness. Second, TLPlan's heuristic formulae don't have a standalone role, and can't form the basis of a planner with anytime capability, like the one presented here.

The forward chaining approach to planning has many benefits that haven't been explored in this paper. For example, it's much easier to implement safety and maintenance goals in a forward-chaining mechanism than in a backward-chaining one. Similarly, it's easier to extend the planner to accept a rich input language, permitting concurrent actions, actions with indirect effects, and so on. These topics are the subject of ongoing work.

The forward-chaining approach to planning is further vindicated in the recent work of Hoffmann and Nebel (2001), whose forward-chaining FF planner outperforms other state-of-the-art domain-independent planners without relying on pre-defined domain-specific rules. Instead, the FF planner automatically extracts heuristics from the domain description to guide the forward search. A promising line of research would be to see how such automatically generated heuristics could be used in combination with reactive rules in a system of the sort described here.

Acknowledgments

This work was funded by EPSRC project GR/N13104, "Cognitive Robotics II". Thanks to Paulo Santos.

References

Agre, P., and Chapman, D. 1987. Pengi: An Implementation of a Theory of Activity. In *Proceedings AAAI 87*, pp. 268–272.

Bacchus, F. and Kabanza, F. 2000. Using Temporal Logics to Express Search Control Knowledge for Planning. *Artificial Intelligence*, vol 116, pp. 123–191.

Brooks, R. 1986. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, Vol.2, pp. 14–23.

Dean, T. and Boddy, M. 1988. An Analysis of Time-Dependent Planning. In *Proceedings AAAI 88*, pp. 49–54.

Drummond, M. 1989. Situated Control Rules. In *Proceedings KR 89*, pp. 103–113.

Gat, E. 1992. Integrating Planning and Reacting in a Heterogenous Asynchronous Architecture for Controlling Real-World Mobile Robots. In *Proceedings AAAI 92*, pp. 809–815.

Hoffmann, J. and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*, 14:253–302.

Kautz, H. and Selman, B. Unifying SAT-Based and Graph-Based Planning. In *Proceedings IJCAI 99*, pp. 318–325.

Korf, R. 1990. Real-Time Heuristic Search. *Artificial Intelligence*, Vol. 42, pp. 189–211.

Mitchell, T. M. 1990. Becoming Increasingly Reactive. In *Proceedings AAAI 90*, pp. 1051–1058.

Pell, B.; Gat, E.; Keesing, R.; Muscettola, N. and Smith, B. 1997. Robust Periodic Planning and Execution for Autonomous Spacecraft. In *Proceedings IJCAI 97*, pp. 1234–1239.

Penberthy, J. S. and Weld, D.S. 1992. UCPOP: A Sound, Complete, Partial Order Planner for ADL. In *Proceedings KR 92*, pp. 103–114.

Schoppers, M.J. 1987. Universal Plans for Reactive Robots in Unpredictable Environments. In *Proceedings IJCAI 87*, pp. 1039-1046.

Zilberstein, S. and Russell, S. J. 1993. Anytime Sensing, Planning and Action: A Practical Model for Robot Control. In *Proceedings IJCAI 93*, pp. 1402–1407.